Red Hat Process Automation Manager 7.13

使用红帽流程自动化管理器中的 OptaPlanner 构建解决者

Last Updated: 2024-03-20

Red Hat Process Automation Manager 7.13 使用红帽流程自动化管理器中的 OptaPlanner 构建解决者

法律通告

摘要

本文档论述了如何通过 Red Hat Process Automation Manager 中的 OptaPlanner 构建解决者找到规划问题的最佳解决方案。

目录

前言

作为业务决策和流程的开发人员，您可以使用红帽构建的 OptaPlanner 构建来开发可决定最佳解决方案以规划问题的最佳解决方案。OptaPlanner 是 Red Hat Process Automation Manager 的内置组件。您可以使用 solution 作为 Red Hat Process Automation Manager 中的服务的一部分，以根据特定限制优化有限资源。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始： master、slave、blacklist 和 whitelist。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看 *CTO Chris Wright* 的信息。

部分 I. 将 OPTAPLANNER 项目的红帽构建升级到 OPTAPLANNER 8

如果您有使用 OptaPlanner 7 或更早的 pubic API 创建的 OptaPlanner 项目，并希望将项目代码升级到 OptaPlanner 8，请参阅本指南中的信息。本指南还包括对在 pubic API 之外的实施类的更改。

OptaPlanner 公共 API 是 OptaPlanner 源代码的子集，可让您通过 Java 代码与 OptaPlanner 进行交互。因此，您可以在同一主发行版本中将 OptaPlanner 版本升级到更高的 OptaPlanner 版本，OptaPlanner 会遵循语义版本。这意味着，您可以在不破坏使用 OptaPlanner public API 的代码的情况下，从 OptaPlanner 7.44 升级到 OptaPlanner 7.48。OptaPlanner 公共 API 类在主版本的 OptaPlanner 发行版本中兼容。但是，当红帽发布一个新的主发行版本时，可能会破坏对公共 API 的更改。

OptaPlanner 8 是一个新的主发行版本，对公共 API 的一些更改与早期版本的 OptaPlanner 不兼容。OptaPlanner 8 将成为未来几年 8.x 系列的基础。对于这个项目的长期优势，对与这个版本要求的早期版本不兼容的公共 API 的更改。

表 1. Red Hat Process Automation Manager 和红帽构建的 OptaPlanner 版本

进程自动化管理器	OptaPlanner
7.7	7.33
7.8	7.39
7.9	7.44
7.10	7.48
7.11	8.5

每个升级注意事项都有一个标签，表明您的代码会受到该更改的影响。下表描述了每个标签：

表 2. 升级影响标签

标签	影响
主要	可能会影响您的代码。
次	不太可能影响您的代码，特别是如果您进行了大量自定义代码，尤其是在遵循示例时。

任何与早期版本的 OptaPlanner 不兼容的更改都会使用 **Public API** 标签进行注解。

第 1 章 与 OPTAPLANNER 7.X 或更早版本不兼容的更改

本节中列出的更改与 OptaPlanner 7.x 或早期版本的 OptaPlanner 不兼容。

需要 Java 11 或更高版本

主要,Public API

如果您使用 JRE 或 JDK 8，升级到 JDK 11 或更高版本。

- 在 Linux 上，从您的 Linux 软件存储库获取 OpenJDK。在 Fedora 和 Red Hat Enterprise Linux 中输入以下命令：

```
sudo dnf install java-11-openjdk-devel
```

- 在 Windows 和 macOS 中，从 AdoptOpenJDK 网站下载 OpenJDK。

SolverFactory 和 PlannerBenchmarkFactory 不再支持 KIE 容器

主要,Public API

因为 OptaPlanner 现在与 Kogito 一致，所以 KIE 容器概念不再适用。因此，**SolverFactory** 不再允许从 KIE 容器创建 **Solver** 实例。这也适用于 **PlannerBenchmarkFactory** 和 基准。

已删除 OSGi 元数据

主要,Public API

由于 OSGi 有限的使用及其带来的维护负担，OptaPlanner 8.x 系列中的 OptaPlanner JAR 文件不再包括在 **META-INF/MANIFEST.MF** 文件中。

使用 Java 序列化

小、公共 API

在 OptaPlanner 8 中，大多数使用 **Serializable** 标记接口已从公共 API 中删除。考虑使用 JSON 或其他格式序列化。

SolverFactory.getScoreDirectorFactory() 替换为 ScoreManager

主要,Public API

在 OptaPlanner 版本 7 中，需要使用 **ScoreDirectorFactory** 来解释分数。在 OptaPlanner 版本 8 中，新的功能被添加到 **ScoreManager** 中，因此不再有理由创建 **ScoreDirector** 的新实例。

OptaPlanner 7 中的 ***.java** 文件示例：

```
ScoreDirectorFactory<CloudBalance> scoreDirectorFactory =
solverFactory.getScoreDirectorFactory();
try (ScoreDirector<CloudBalance> scoreDirector = scoreDirectorFactory.buildScoreDirector()) {
    scoreDirector.setWorkingSolution(solution);
    Score score = scoreDirector.calculateScore();
}
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
ScoreManager<CloudBalance> scoreManager = ScoreManager.create(solverFactory);
Score score = scoreManager.updateScore(solution);
```

允许您检索 **ScoreDirector** 和 **ScoreDirectorFactory** 实例的方法已从公共 API 中删除，无需替换。缩减了 **ScoreDirector** 接口版本被提升到公共 API，以将 **ProblemFactChange** 接口提升到公共 API。

SolverFactory:getSolverConfig（） 删除
小、公共 API

SolverFactory.getSolverConfig（） 方法已弃用，并被 **SolverFactory.create(SolverConfig)** 方法替代。**SolverConfig** 实例现在在实例化 **SolverFactory** 实例之前被实例化，这是更自然的。以上顺序已被删除。

OptaPlanner 7 中的 ***.java** 文件示例：

```
SolverFactory<MySolution> solverFactory =
SolverFactory.createFromXmlResource(".../mySolverConfig.xml");
SolverConfig solverConfig = solverFactory.getSolverConfig();
...
Solver<MySolution> solver = solverFactory.buildSolver();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource(".../mySolverConfig.xml");
...
SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
Solver<MySolution> solver = solverFactory.buildSolver();
```

如果您还通过了类 **Loader**，请将其传递到 **SolverConfig.createFromXmlResource（）** 和 **SolverFactory.create（）**。

SolverConfig:buildSolver（） 删除
小、公共 API

SolverConfig.buildSolver（） 方法是不属于公共 API 的内向方法。改为使用 **SolverFactory.buildSolver（）** 方法。

OptaPlanner 7 中的 ***.java** 文件示例：

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource(".../mySolverConfig.xml");
...
Solver<MySolution> solver = solverConfig.buildSolver();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource(".../mySolverConfig.xml");
...
SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
Solver<MySolution> solver = solverFactory.buildSolver();
```

PlannerBenchmarkConfig:buildPlannerBenchmark（） 删除
小、公共 API

PlannerBenchmarkConfig.buildPlannerBenchmark（） 方法是不属于公共 API 的内建方法。使用 **PlannerBenchmarkFactory.buildPlannerBenchmark（）** 方法。

OptaPlanner 7 中的 ***.java** 文件示例：

```
PlannerBenchmarkConfig benchmarkConfig = PlannerBenchmarkConfig.createFromXmlResource(
        ".../cloudBalancingBenchmarkConfig.xml");
...
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
PlannerBenchmarkConfig benchmarkConfig = PlannerBenchmarkConfig.createFromXmlResource(
        ".../cloudBalancingBenchmarkConfig.xml");
...
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.create(benchmarkConfig);
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();
```

solverFactory:cloneSolverFactory（） 已删除
小、公共 API

SolverFactory.cloneSolverFactory（） 方法已弃用，并被新的 **SolverConfig(SolverConfig)** copyors 和 **SolverFactory.cloneSolverFactory（）** 方法替代。

OptaPlanner 7 中的 ***.java** 文件示例：

```
private SolverFactory<MySolution> base;

public void userRequest(..., long userInput) {
    SolverFactory<MySolution> solverFactory = base.cloneSolverFactory();
    solverFactory.getSolverConfig()
            .getTerminationConfig()
            .setMinutesSpentLimit(userInput);
    Solver<MySolution> solver = solverFactory.buildSolver();
    ...
}
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
private SolverConfig base;

public void userRequest(..., long userInput) {
    SolverConfig solverConfig = new SolverConfig(base); // Copy it
    solverConfig.getTerminationConfig()
            .setMinutesSpentLimit(userInput);
    SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
    Solver<MySolution> solver = solverFactory.buildSolver();
    ...
}
```

SolverFactory:createEmpty（） 删除
小、公共 API

SolverFactory.createEmpty（） 方法已弃用，并被 **新的 SolverConfig（）** 方法替代。**SolverFactory.createEmpty（）** 方法已被删除。

OptaPlanner 7 中的 ***.java** 文件示例：

```
SolverFactory<MySolution> solverFactory = SolverFactory.createEmpty();
SolverConfig solverConfig = solverFactory.getSolverConfig()
...
Solver<MySolution> solver = solverFactory.buildSolver();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
SolverConfig solverConfig = new SolverConfig();
...
SolverFactory<MySolution> solverFactory = SolverFactory.create(solverConfig);
Solver<MySolution> solver = solverFactory.buildSolver();
```

XML <solver/> root 元素现在属于 http://www.optaplanner.org/xsd/solver 命名空间

主要,Public API

OptaPlanner 现在为解决器配置提供了一个 XML 模式定义。虽然 OptaPlanner 保留与现有 XML 配置的早期版本的兼容性，但强烈建议迁移到 XSD，因为 OptaPlanner 可能在以后只支持有效的配置 XML。

OptaPlanner 7 中的 ***SolverConfig.xml** 文件中的示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<solver>
  ...
</solver>
```

OptaPlanner 8 中的 ***SolverConfig.xml** 文件示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  ...
</solver>
```

使用 XSD 可能需要重新排序配置的一些 XML 元素。在 IDE 中使用代码完成迁移到有效的 XML。

移动选择器配置中的属性 subPillarEnabled 已被删除

小、公共 API

PillarSwapMoveSelector 和 **PillarChangeMoveSelector** 中的 **subPillarEnabled** 属性已弃用，并被一个新的属性 **subPillarType** 替代。**subPillarEnabled** 属性已被删除。

OptaPlanner 7 中的 ***SolverConfig.xml** 和 ***BenchmarkConfig.xml** 文件中的示例：

```
<pillar...MoveSelector>
  ...
  <pillarSelector>
    <subPillarEnabled>false</subPillarEnabled>
    ...
  </pillarSelector>
  ...
</pillar...MoveSelector>
```

OptaPlanner 8 中的 ***SolverConfig.xml** 和 ***BenchmarkConfig.xml** 文件中的示例：

```
<pillar...MoveSelector>
  <subPillarType>NONE</subPillarType>
  <pillarSelector>
    ...
  </pillarSelector>
  ...
</pillar...MoveSelector>
```

solver:getScoreDirectorFactory（） 已删除
主要,Public API

getScoreDirectorFactory（） 方法已被弃用，现已从 Solver 和 **Solver Factory** 类中删除。

您不再需要创建一个 **Solver** 实例来计算或解释 UI 中的分数。使用 **ScoreManager** API 替代。

OptaPlanner 7 中的 ***.java** 文件示例：

```
SolverFactory<VehicleRoutingSolution> solverFactory = SolverFactory.createFromXmlResource(...);
Solver<VehicleRoutingSolution> solver = solverFactory.buildSolver();
uiScoreDirectorFactory = solver.getScoreDirectorFactory();
...
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
SolverFactory<VehicleRoutingSolution> solverFactory = SolverFactory.createFromXmlResource(...);
ScoreManager<VehicleRoutingSolution> scoreManager = ScoreManager.create(solverFactory);
...
```

ScoreDirectorFactory 不再应当使用，因为它始终不在公共 API 外部，且其功能都由公共 API 的不同部分公开。

solver.explainBestScore（） 已被删除
主要,Public API

Solver 接口上的 **explainBestScore（）** 方法已在 7.x 中弃用，现在已被删除。您可以通过新的 **ScoreManager** API 获取相同的信息。

红帽建议您不要以任何方式解析这个方法调用的结果。

OptaPlanner 7 中的 ***.java** 文件示例：

```
solver = ...;
scoreExplanation = solver.explainBestScore();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
MySolution solution = ...;
ScoreManager<MySolution> scoreManager = ...;
scoreExplanation = scoreManager.explainScore(solution);
```

Solver 接口方法 getBestSolution （）、getBestScore（） 和 getTimeMillisSpent（） 已被删除

小、公共 API

Solver 接口中的几种方法已在 7.x 中弃用，并已被删除。您可以通过通过 **Solver.add EventListener (...)** 注册 EventListener 来获取相同的信息。

OptaPlanner 7 中的 ***.java** 文件示例：

```
solver = ...;
solution = solver.getBestSolution();
score = solver.getBestScore();
timeMillisSpent = solver.getTimeMillisSpent();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
solver = ...;
solver.addEventListener(event -> {
    solution = event.getNewBestSolution();
    score = event.getNewBestScore();
    timeMillisSpent = event.getTimeMillisSpent();
});
```

已删除注解扫描

主要,Public API

solver 配置中的 **<scanAnnotatedClasses** /> 指令已在 7.x 中弃用，现已被删除。

OptaPlanner 7 中的 ***.xml** 文件示例：

```
<solver>
  ...
  <scanAnnotatedClasses/>
  ...
</solver>
```

OptaPlanner 8 中的 ***.xml** 文件示例：

```
<solver>
  ...
  <solutionClass>...</solutionClass>
  <entityClass>...</entityClass>
  ...
</solver>
```

用于 @PlanningFactProperty 和 @PlanningFactCollection的新软件包

主要,Public API

@PlanningFactProperty 和 **@PlanningFactCollectionProperty** 注释现在与其他类似注释（如 **@PlanningSolution** ）共享相同的软件包。旧注解在 7.x 中弃用并被删除。

OptaPlanner 7 中的 ***.java** 文件示例：

```
import org.optaplanner.core.api.domain.solution.drools.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.solution.drools.ProblemFactProperty;
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.solution.ProblemFactProperty;
```

filterClassList 替换为单个过滤器类

小、公共 API

EntitySelector 的配置**ValueSelector** 和 **MoveSelector** 现在在配置 API 和 addressr 配置 XML 中均有一个过滤器类。

在实践中，您不需要多个选择过滤类，您可以使用单一选择过滤器类替换它们，该类实施所有这些逻辑。现在，传递单个选择类需要较少的样板代码。

OptaPlanner 7 中的 ***.java** 文件示例：

```
ValueSelectorConfig valueSelectorConfig = new ValueSelectorConfig();
valueSelectorConfig.setFilterClassList(Collections.singletonList(MySelectionFilterClass.class));
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
ValueSelectorConfig valueSelectorConfig = new ValueSelectorConfig();
valueSelectorConfig.setFilterClass(MySelectionFilterClass.class);
```

使用单个选择过滤器类替换多个选择过滤器类

OptaPlanner 7 中的 ***.xml** 文件示例：

```
<swapMoveSelector>
  <entitySelector>
    <filterClass>com.example.FilterA</filterClass>
    <filterClass>com.example.FilterB</filterClass>
  </entitySelector>
</swapMoveSelector>
```

OptaPlanner 7 中的 ***.java** 文件示例：

```
package com.example;
...
public class FilterA implements SelectionFilter<MySolution, MyPlanningEntity> {

    @Override
    public boolean accept(ScoreDirector<MySolution> scoreDirector, MyPlanningEntity selection) {
        return selection.getValue() < 500;
    }
}
```

```
package com.example;
...
public class FilterB implements SelectionFilter<MySolution, MyPlanningEntity> {
```

```
    @Override
    public boolean accept(ScoreDirector<MySolution> scoreDirector, MyPlanningEntity selection) {
        return selection.getOrder() == Order.ASC;
    }
}
```

OptaPlanner 8 中的 ***.xml** 文件示例：

```
<swapMoveSelector>
  <entitySelector>
    <filterClass>com.example.SingleEntityFilter</filterClass>
  </entitySelector>
</swapMoveSelector>
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
package com.example;
...
public class SingleEntityFilter implements SelectionFilter<MySolution, MyPlanningEntity> {

    @Override
    public boolean accept(ScoreDirector<MySolution> scoreDirector, MyPlanningEntity selection) {
        return selection.getValue() < 500 && selection.getOrder() == Order.ASC;
    }
}
```

AcceptorConfig 重命名为 LocalSearchAcceptorConfig
次

这只会影响配置 API。解决器配置 XML 文件保持不变。

实施了与其他本地搜索特定配置类的命名一致性。

OptaPlanner 7 中的 ***.java** 文件示例：

```
LocalSearchPhaseConfig localSearchPhaseConfig = new LocalSearchPhaseConfig()
        .withAcceptorConfig(new AcceptorConfig().withEntityTabuSize(5));
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
LocalSearchPhaseConfig localSearchPhaseConfig = new LocalSearchPhaseConfig()
        .withAcceptorConfig(new LocalSearchAcceptorConfig().withEntityTabuSize(5));
```

自定义属性 **XML** 配置格式更改
小、公共 API

此问题只影响解决器配置 XML，特别是 < **scoreDirectorFactory/>, < moveIteratoronnectionFactoryy/>, < moveListFactory/> , &** lt; **partitionedSearch/> 和 &** lt **;customPhase/>**。

这个更改的目的是在构建时强制配置 XML 的结构。

OptaPlanner 7 中的 ***.xml** 文件示例：

```
<partitionedSearch>
  <solutionPartitionerClass>com.example.MySolutionPartitioner</solutionPartitionerClass>
  <solutionPartitionerCustomProperties>
    <partCount>4</partCount> <!-- a custom property -->
    <minimumProcessListSize>300</minimumProcessListSize> <!-- a custom property -->
  </solutionPartitionerCustomProperties>
</partitionedSearch>
```

OptaPlanner 8 中的 ***.xml** 文件示例：

```
<partitionedSearch>
  <solutionPartitionerClass>com.example.MySolutionPartitioner</solutionPartitionerClass>
  <solutionPartitionerCustomProperties>
    <property name="partCount" value="4"/> <!-- a custom property -->
    <property name="minimumProcessListSize" value="300"/> <!-- a custom property -->
  </solutionPartitionerCustomProperties>
</partitionedSearch>
```

<variableNameInclude/& gt; 元素现在被 < variableNameIncludes/> 元素嵌套
小、公共 API

这个版本只影响解决器配置 XML，特别是 < **swapMoveSelector/> 和 &** lt **;pillarSwapMoveSelector/>** 元素。

这个更改的目的是在构建时强制配置 XML 的结构。

OptaPlanner 7 中的 ***.xml** 文件示例：

```
<swapMoveSelector>
  <variableNameInclude>variableA</variableNameInclude>
  <variableNameInclude>variableB</variableNameInclude>
</swapMoveSelector>
```

OptaPlanner 8 中的 ***.xml** 文件示例：

```
<swapMoveSelector>
  <variableNameIncludes>
    <variableNameInclude>variableA</variableNameInclude>
    <variableNameInclude>variableB</variableNameInclude>
  </variableNameIncludes>
</swapMoveSelector>
```

删除 **解决方案** 接口
小、公共 API

Solution 接口已弃用并删除。同时还删除了 **AbstractSolution** 接口（仅适用于 Business Central）。

删除 **Solution** 接口，将 **getScore（）** 方法标上 **@PlanningScore**，并将 **getProblemFacts（）** 方法替换为 **@ProblemFactCollectionProperty** 注释（或字段）。

OptaPlanner 7 中的 ***.java** 文件示例：

```
@PlanningSolution
public class CloudBalance implements Solution<HardSoftScore> {
```

```
    private List<CloudComputer> computerList;
    ...

    private HardSoftScore score;

    @ValueRangeProvider(id = "computerRange")
    public List<CloudComputer> getComputerList() {...}

    public HardSoftScore getScore() {...}
    public void setScore(HardSoftScore score) {...}

    public Collection<? extends Object> getProblemFacts() {
        List<Object> facts = new ArrayList<Object>();
        facts.addAll(computerList);
        ...
        return facts;
    }

}
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
@PlanningSolution
public class CloudBalance {

    private List<CloudComputer> computerList;
    ...

    private HardSoftScore score;

    @ValueRangeProvider(id = "computerRange")
    @ProblemFactCollectionProperty
    public List<CloudComputer> getComputerList() {...}

    @PlanningScore
    public HardSoftScore getScore() {...}
    public void setScore(HardSoftScore score) {...}

}
```

对于没有嵌套在 **Collection** 中的单个问题事实，请使用 **@ProblemFactProperty** 注释，如下例所示，其中字段注解如下：

OptaPlanner 7 中的 ***.java** 文件示例：

```
@PlanningSolution
public class CloudBalance implements Solution<HardSoftScore> {

    private CloudParametrization parametrization;
    private List<CloudBuilding> buildingList;
    @ValueRangeProvider(id = "computerRange")
    private List<CloudComputer> computerList;
    ...
```

```
    public Collection<? extends Object> getProblemFacts() {
        List<Object> facts = new ArrayList<Object>();
        facts.add(parametrization); // not a Collection
        facts.addAll(buildingList);
        facts.addAll(computerList);
        ...
        return facts;
    }

}
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
@PlanningSolution
public class CloudBalance {

    @ProblemFactProperty
    private CloudParametrization parametrization;
    @ProblemFactCollectionProperty
    private List<CloudBuilding> buildingList;
    @ValueRangeProvider(id = "computerRange")
    @ProblemFactCollectionProperty
    private List<CloudComputer> computerList;
    ...

}
```

不要在 getters（或字段）上添加 **@ProblemFactCollectionProperty** 注释，该注释具有 **@PlanningEntityCollectionProperty** 注释。

BestSolutionChangedEvent:isNewBestSolutionInitialized（） 已删除

小、公共 API

best **SolutionChangedEvent.isNewBestSolutionInitialized（）** 方法已弃用，并替换为 **BestSolutionChangedEvent.getNewBestSolution（）.getScore（）.isSolutionInitialized（）** 方法。best **SolutionChangedEvent.isNewBestSolutionInitialized（）** 方法已被删除。

OptaPlanner 7 中的 ***.java** 文件示例：

```
    public void bestSolutionChanged(BestSolutionChangedEvent<CloudBalance> event) {
        if (event.isEveryProblemFactChangeProcessed()
                && event.isNewBestSolutionInitialized()) {
            ...
        }
    }
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
    public void bestSolutionChanged(BestSolutionChangedEvent<CloudBalance> event) {
        if (event.isEveryProblemFactChangeProcessed()
                && event.getNewBestSolution().getScore().isSolutionInitialized()) {
            ...
        }
    }
```

如果您检查 **是否可行（）**，它会检查是否初始化解决方案。

OptaPlanner 8 中的 ***.java** 文件示例：

```
public void bestSolutionChanged(BestSolutionChangedEvent<CloudBalance> event) {
    if (event.isEveryProblemFactChangeProcessed()
            // isFeasible() checks isSolutionInitialized() too
            && event.getNewBestSolution().getScore().isFeasible()) {
        ...
    }
}
```

<valueSelector > : variableName 现在是一个属性
小、公共 API

当 power-tweaking 选择器（如 &lt **;changeMoveSelector&** gt;）时，带有多个规划变量的用例中，&lt **;variableName&** gt; XML 元素已被一个 **variableName="..."** XML 属性替代。这个变化减少了解决器配置详细程度。在整个 7.x 系列中被弃用后，旧方法现已被删除。

OptaPlanner 7 中的 ***SolverConfig.xml** 和 ***BenchmarkConfig.xml** 文件中的示例：

```
<valueSelector>
  <variableName>room</variableName>
</valueSelector>
```

OptaPlanner 8 中的 ***SolverConfig.xml** 和 ***BenchmarkConfig.xml** 文件中的示例：

```
<valueSelector variableName="room"/>
```

分区搜索： 已删除 **threadFactoryClass**
小、公共 API

因为 **<solver&** gt; 已在某些情况下支持 **<threadFactoryClass** > 元素，所以 & **lt; partitionedSearch>** 下的 **<threadFactoryClass &** gt; 元素已被删除。

OptaPlanner 7 中的 ***SolverConfig.xml** 和 ***BenchmarkConfig.xml** 文件中的示例：

```
<solver>
  ...
  <partitionedSearch>
    <threadFactoryClass>...MyAppServerThreadFactory</threadFactoryClass>
    ...
  </partitionedSearch>
</solver>
```

OptaPlanner 8 中的 ***SolverConfig.xml** 和 ***BenchmarkConfig.xml** 文件中的示例：

```
<solver>
  <threadFactoryClass>...MyAppServerThreadFactory</threadFactoryClass>
  ...
  <partitionedSearch>
    ...
  </partitionedSearch>
</solver>
```

SimpleDoubleScore 和 HardSoftDoubleScore 已删除

小、公共 API

不建议使用基于双倍的分数类型，因为它们可能会造成损坏。它们已被删除。

OptaPlanner 7 中的 ***.java** 文件示例：

```
@PlanningSolution
public class MyPlanningSolution {

    private SimpleDoubleScore score;

    ...

}
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
@PlanningSolution
public class MyPlanningSolution {

    private SimpleLongScore score;

    ...

}
```

Score.toInitializedScore() removed

小、公共 API

Score.toInitializedScore（） 方法已弃用，并替换为 7.x 中的 **Score.withInitScore(int)** 方法，现已被删除。

OptaPlanner 7 中的 ***.java** 文件示例：

```
score = score.toInitializedScore();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
score = score.withInitScore(0);
```

各种 验证工具 被删除

小、公共 API

在 7.x 中弃用了以下 **Comparator** 实现：

- **org.optaplanner.core.api.score.comparator.NaturalScoreComparator**
- **org.optaplanner.core.api.score.constraint.ConstraintMatchScoreComparator**
- **org.optaplanner.core.api.score.constraint.ConstraintMatchTotalScoreComparator**
- **org.optaplanner.core.api.score.constraint.IndictmentScoreComparator**

OptaPlanner 7 中的 ***.java** 文件示例：

```
NaturalScoreComparator comparator = new NaturalScoreComparator();
ConstraintMatchScoreComparator comparator2 = new ConstraintMatchScoreComparator();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
Comparator<Score> comparator = Comparable::compareTo;
Comparator<ConstraintMatch> comparator2 = Comparator.comparing(ConstraintMatch::getScore);
```

移除了 FeasibilityScore

小、公共 API

在 7.x 中弃用了 **FeasibilityScore** 接口，它 **的唯一方法是Feasible（）** 移到 **Score** 超级类型。接口现已被删除。

您应该按最终类型引用 **分数**，例如 **HardSoftScore** 而不是 **分数**。

@PlanningEntity.movableEntitySelectionFilter removed

小、公共 API

@PlanningEntity 注解上的 **movableEntitySelectionFilter** 字段已在 7.x 中弃用，一个新的字段 **pinningFilter** 已被引入，显示与 **@PlanningPin** 注解相关的名称。此过滤器实施新的 **PinningFilter** 接口，如果实体被固定，则返回 true；如果有禁止。因此，与旧过滤器相比，这个新过滤器的逻辑已被验证。

您应该通过提供新过滤器而不是旧的过滤器来更新 **@PlanningEntity** 注解。旧过滤器现已被删除。

OptaPlanner 7 中的 ***.java** 文件示例：

```
@PlanningEntity(movableEntitySelectionFilter = MyMovableEntitySelectionFilter.class)
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
@PlanningEntity(pinningFilter = MyPinningFilter.class)
```

@PlanningVariable.reinitializeVariableEntityFilter removed

小、公共 API

@Planning Variable 注解上的 reinitializeVariableFilter 字段已在 7.x 中被弃用，现已被删除。

* 被设置为接口（ScoreHolder 类）

小、公共 API

在 OptaPlanner 7 中，**ScoreHolder** 类专门用于 Drools 分数计算，公开一些公共方法（如果使用），允许用户意外破坏或对它们的分数造成负面影响。

在 OptaPlanner 8 中，这些方法已被删除，并将类转换为接口。大多数用户不使用移除且有害的方法。

但是，如果您使用这些方法，您可以在分数解释和约束配置方面在公共 API 中找到合适的替换。

ValueRangeFactory 类现在最后

次

ValueRangeFactory 类是仅具有静态方法的工厂类。您不需要扩展这个类，因此最终是 **最终的**。

OptaPlanner 7 中的 ***.java** 文件示例：

```
class MyValueRangeFactory extends ValueRangeFactory {
    ...
}
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
class MyValueRangeFactory {
    ...
}
```

现在，**ConstraintMatchTotal** 和 **Indictment** 是接口
小、公共 API

ConstraintMatchTotal 和 **Indictment** 类已转换为接口。因此，他们的实施过程从公共 API 中移出，并附带允许它们修改其状态的方法。这些方法绝不能用于公共 API，因此没有替换它们。

如果您选择实施 **ConstraintMatchAwareIncrementalScoreCalculator**，您可能仍需要实例本身：

```
ConstraintMatchTotal maximumCapacityMatchTotal = new ConstraintMatchTotal(...);
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
ConstraintMatchTotal maximumCapacityMatchTotal = new DefaultConstraintMatchTotal(...);
```

分数管理器 ：添加了通用类型 **Score**
主要,Public API

ScoreManager 和 **ScoreExplanation** API 现在具有通用类型 **Score**，以避免在您的代码中进行停机，例如：从 **Score** 到 **HardSoftScore**。

OptaPlanner 7 中的 ***.java** 文件示例：

```
@Inject // or @Autowired
ScoreManager<TimeTable> scoreManager;
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
@Inject // or @Autowired
ScoreManager<TimeTable, HardSoftScore> scoreManager;
```

OptaPlanner 7 中的 ***.java** 文件示例：

```
ScoreExplanation<TimeTable> explanation = scoreManager.explainScore(timeTable);
HardSoftScore score = (HardSoftScore) explanation.getScore();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
    ScoreExplanation<TimeTable, HardSoftScore> explanation =
scoreManager.explainScore(timeTable);
    HardSoftScore score = explanation.getScore();
```

ConstraintMatchTotal、 ConstraintMatch 和 Indictment: 通用类型 Score 添加
主要

与 **ScoreManager** 和 **ScoreExplanation** 类似， **ConstraintMatchTotal ConstraintMatch**, 和 **Indictment** API 现在有一个通用类型 **Score** 来避免代码中的广播，例如：从 **Score** 到 **HardSoftScore**。

OptaPlanner 7 中的 ***.java** 文件示例：

```
    ScoreExplanation<TimeTable> explanation = scoreManager.explainScore(timeTable);
    Map<String, ConstraintMatchTotal> constraintMatchTotalMap =
scoreExplanation.getConstraintMatchTotalMap();
    ConstraintMatchTotal constraintMatchTotal = constraintMatchTotalMap.get(contraintId);
    HardSoftScore totalScore = (HardSoftScore) constraintMatchTotal.getScore();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
    ScoreExplanation<TimeTable, HardSoftScore> explanation =
scoreManager.explainScore(timeTable);
    Map<String, ConstraintMatchTotal<HardSoftScore>> constraintMatchTotalMap =
scoreExplanation.getConstraintMatchTotalMap();
    ConstraintMatchTotal<HardSoftScore> constraintMatchTotal =
constraintMatchTotalMap.get(contraintId);
    HardSoftScore totalScore = constraintMatchTotal.getScore();
```

OptaPlanner 7 中的 ***.java** 文件示例：

```
    ScoreExplanation<TimeTable> explanation = scoreManager.explainScore(timeTable);
    Map<Object, Indictment> indictmentMap = scoreExplanation.getIndictmentMap();
    Indictment indictment = indictmentMap.get(lesson);
    HardSoftScore totalScore = (HardSoftScore) indictment.getScore();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
    ScoreExplanation<TimeTable, HardSoftScore> explanation =
scoreManager.explainScore(timeTable);
    Map<Object, Indictment<HardSoftScore>> indictmentMap = scoreExplanation.getIndictmentMap();
    Indictment<HardSoftScore> indictment = indictmentMap.get(lesson);
    HardSoftScore totalScore = indictment.getScore();
```

ConstraintMatchAwareIncrementalScoreCalculator: generic type Score added
次

接口 **ConstraintMatchAwareIncrementalScoreCalculator** 还有一个通用类型参数，用于 **Score** 以避免 **ConstraintMatchTotal** 和 **Indictment** 的原始类型使用。

OptaPlanner 7 中的 ***.java** 文件示例：

```
public class MachineReassignmentIncrementalScoreCalculator
        implements ConstraintMatchAwareIncrementalScoreCalculator<MachineReassignment> {
```

```
    @Override
    public Collection<ConstraintMatchTotal> getConstraintMatchTotals() {
        ...
    }


    @Override
    public Map<Object, Indictment> getIndictmentMap() {
        ...
    }

}
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
public class MachineReassignmentIncrementalScoreCalculator
        implements ConstraintMatchAwareIncrementalScoreCalculator<MachineReassignment,
HardSoftLongScore> {

    @Override
    public Collection<ConstraintMatchTotal<HardSoftLongScore>> getConstraintMatchTotals() {
        ...
    }


    @Override
    public Map<Object, Indictment<HardSoftLongScore>> getIndictmentMap() {
        ...
    }

}
```

AbstractCustomPhaseCommand 被删除

小、公共 API

抽象类 **AbstractCustomPhaseCommand** 被删除。任何扩展该类都应直接实施 **CustomPhaseCommand** 接口。

OptaPlanner 7 中的 ***.java** 文件示例：

```
public class DinnerPartySolutionInitializer extends AbstractCustomPhaseCommand<DinnerParty> {

    @Override
    public void changeWorkingSolution(ScoreDirector<DinnerParty> scoreDirector) {
        ...
    }

}
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
public class DinnerPartySolutionInitializer implements CustomPhaseCommand<DinnerParty> {

    @Override
```

```
    public void changeWorkingSolution(ScoreDirector<DinnerParty> scoreDirector) {
        ...
    }

}
```

分数计算器移到公共 API
主要

接口 **EasyScoreCalculator**,**IncrementalScoreCalculator** 和 **ConstraintMatchAwareIncrementalScoreCalculator** 已移至公共 API 中的新软件包。它们已弃用的对应项已被删除。弃用的类 **org.optaplanner.core.impl.score.director.incremental.AbstractIncrementalScoreCalculator** 已被删除。使用删除的接口和类替换其在公共 API 中的对应接口。

OptaPlanner 7 中的 **EasyScoreCalculator.java** 文件中的示例：

```
...
import org.optaplanner.core.impl.score.director.easy.EasyScoreCalculator;
...

public class CloudBalancingEasyScoreCalculator implements EasyScoreCalculator<CloudBalance>
{
    ...
}
```

OptaPlanner 8 中的 **EasyScoreCalculator.java** 文件中的示例：

```
...
import org.optaplanner.core.api.score.calculator.EasyScoreCalculator;
...

public class CloudBalancingEasyScoreCalculator implements EasyScoreCalculator<CloudBalance,
HardSoftScore> {
    ...
}
```

OptaPlanner 7 中的 **IncrementalScoreCalculator.java** 文件中的示例：

```
...
import org.optaplanner.core.impl.score.director.incremental.AbstractIncrementalScoreCalculator;
...

public class CloudBalancingIncrementalScoreCalculator extends
AbstractIncrementalScoreCalculator<CloudBalance> {
    ...
}
```

OptaPlanner 8 中的 **IncrementalScoreCalculator.java** 文件中的示例：

```
...
import org.optaplanner.core.api.score.calculator.IncrementalScoreCalculator;
...
```

```
  public class CloudBalancingIncrementalScoreCalculator implements
IncrementalScoreCalculator<CloudBalance, HardSoftScore> {
    ...
  }
```

OptaPlanner 7 中的 **ConstraintMatchAwareIncrementalScoreCalculator.java** 文件中的示例：

```
  ...
  import org.optaplanner.core.impl.score.director.incremental.AbstractIncrementalScoreCalculator;
  import
org.optaplanner.core.impl.score.director.incremental.ConstraintMatchAwareIncrementalScoreCalculator
;
  ...

  public class CheapTimeConstraintMatchAwareIncrementalScoreCalculator
      extends AbstractIncrementalScoreCalculator<CheapTimeSolution>
      implements ConstraintMatchAwareIncrementalScoreCalculator<CheapTimeSolution> {
    ...
  }
```

OptaPlanner 8 中的 **ConstraintMatchAwareIncrementalScoreCalculator.java** 文件中的示例：

```
  ...
  import org.optaplanner.core.api.score.calculator.ConstraintMatchAwareIncrementalScoreCalculator;
  ...

  public class CheapTimeConstraintMatchAwareIncrementalScoreCalculator
      implements ConstraintMatchAwareIncrementalScoreCalculator<CheapTimeSolution,
HardMediumSoftLongScore> {
    ...
  }
```

PlannerBenchmarkFactory:createFromSolverFactory（） 已删除
主要,Public API

PlannerBenchmarkFactory.createFromSolverFactory（） 方法已弃用，并被 PlannerBenchmarkFactory.createFromSolverConfigXmlResource(String) 方法替代。PlannerBenchmarkFactory.createFromSolverFactory（） 方法已被删除。

OptaPlanner 7 中的 ***.java** 文件示例：

```
SolverFactory<CloudBalance> solverFactory = SolverFactory.createFromXmlResource(
      ".../cloudBalancingSolverConfig.xml");
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.createFromSolverFactory(solverFactory);
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.createFromSolverConfigXmlResource(
      ".../cloudBalancingSolverConfig.xml");
```

如果您以编程方式调整了解决器配置，您可以使用 **PlannerBenchmarkConfig.createFromSolverConfig(SolverConfig)**，然后运行 **PlannerBenchmarkFactory.create(PlannerBenchmarkConfig)**。

PlannerBenchmarkFactory:getPlannerBenchmarkConfig（） 已删除

小、公共 API

PlannerBenchmarkFactory.getPlannerBenchmarkConfig（） 方法已弃用，并被 **PlannerBenchmarkFactory.create(PlannerBenchmarkConfig)** 方法替代。现在，**PlannerBenchmarkConfig** 实例在 **PlannerBenchmark factory** 实例被实例化前。这个顺序更为逻辑。**PlannerBenchmarkFactory.getPlannerBenchmarkConfig（）** 已被删除。

OptaPlanner 7 中的 ***.java** 文件示例：

```
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.createFromXmlResource(
        ".../cloudBalancingBenchmarkConfig.xml");
PlannerBenchmarkConfig benchmarkConfig = benchmarkFactory.getPlannerBenchmarkConfig();
...
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();
```

OptaPlanner 8 中的 ***.java** 文件示例：

```
PlannerBenchmarkConfig benchmarkConfig = PlannerBenchmarkConfig.createFromXmlResource(
        ".../cloudBalancingBenchmarkConfig.xml");
...
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.create(benchmarkConfig);
PlannerBenchmark benchmark = benchmarkFactory.buildPlannerBenchmark();
```

XML <plannerBenchmark/> root 元素现在属于 http://www.optaplanner.org/xsd/benchmark 命名空间

小、公共 API

OptaPlanner 现在为基准配置提供一个 XML 架构定义(XSD)。虽然 OptaPlanner 与现有 XML 配置的早期版本保持兼容性，但强烈建议迁移到 XSD，因为 OptaPlanner 可能在以后仅支持有效的配置 XML。

OptaPlanner 7 中的 ***BenchmarkConfig.xml** 文件中的示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<plannerBenchmark>
  ...
</plannerBenchmark>
```

OptaPlanner 8 中的 ***BenchmarkConfig.xml** 文件中的示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<plannerBenchmark xmlns="https://www.optaplanner.org/xsd/benchmark"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.optaplanner.org/xsd/benchmark
https://www.optaplanner.org/xsd/benchmark/benchmark.xsd">
  ...
</plannerBenchmark>
```

使用 XSD 可能需要重新排序配置的一些 XML 元素。在 IDE 中使用代码完成迁移到有效的 XML。

ProblemBenchmarksConfig:xStreamAnnotatedClass 删除
主要,Public API

{mAnnotatedClas} 已从 < **problemBenchmarks/&gt**; 配置以及对应的 **getXStreamAnnotatedClassList（）** 和 **setXStreamAnnotatedClassList（）** 方法中删除。

OptaPlanner 7 中的 *.java 文件示例：

```
ProblemBenchmarksConfig problemBenchmarksConfig = new ProblemBenchmarksConfig();
problemBenchmarksConfig.setXStreamAnnotatedClassList(MySolution.class);
```

OptaPlanner 8 中的 *.java 文件示例：

```
package com.example;
...
public class MySolutionFileIO extends XStreamSolutionFileIO<MySolution> {
    public MySolutionFileIO() {
        super(MySolution.class);
    }
}

...

ProblemBenchmarksConfig problemBenchmarksConfig = new ProblemBenchmarksConfig();
problemBenchmarksConfig.setSolutionFileIOClass(MySolutionFileIO.class);
```

OptaPlanner 7 中的 *BenchmarkConfig.xml 文件中的示例：

```
<plannerBenchmark>
...
  <solverBenchmark>
    <problemBenchmarks>
      <xStreamAnnotatedClass>com.example.MySolution</xStreamAnnotatedClass>
      ...
    </problemBenchmarks>
    ...
  </solverBenchmark>
...
</plannerBenchmark>
```

OptaPlanner 8 中的 *BenchmarkConfig.xml 文件中的示例：

```
<plannerBenchmark>
```

```
...
  <solverBenchmark>
    <problemBenchmarks>
      <!-- See the "After in *.java" section to create the MySolutionFileIO. -->
      <solutionFileIOClass>com.example.MySolutionFileIO</solutionFileIOClass>
      ...
    </problemBenchmarks>
    ...
  </solverBenchmark>
...
</plannerBenchmark>
```

BenchmarkAggregatorFrame.createAndDisplay(PlannerBenchmark factory) removed

次

BenchmarkAggregatorFrame.createAndDisplay(PlannerBenchmarkFactory) 方法已弃用，并替换为 BenchmarkAggregatorFrame.createAndDisplayFromXmlResource(String) 方法。BenchmarkAggregatorFrame.createAndDisplay(PlannerBenchmark factory) 方法已被删除。

OptaPlanner 7 中的 *.java 文件示例：

```
PlannerBenchmarkFactory benchmarkFactory =
PlannerBenchmarkFactory.createFromXmlResource(
        ".../cloudBalancingBenchmarkConfig.xml");
BenchmarkAggregatorFrame.createAndDisplay(benchmarkFactory);
```

OptaPlanner 8 中的 *.java 文件示例：

```
BenchmarkAggregatorFrame.createAndDisplayFromXmlResource(
        ".../cloudBalancingBenchmarkConfig.xml");
```

如果您以编程方式调整基准配置，可以使用 BenchmarkAggregatorFrame.createAndDisplay(PlannerBenchmarkConfig)。

删除了配置中的 JavaScript 表达式支持

次

解决程序配置和基准配置的不同元素不再支持嵌套的 JavaScript 表达式。您必须使用自动配置或整数常量替换它们。

OptaPlanner 7 中的 solverConfig.xml 文件中的示例：

```
<solver>
  ...
  <moveThreadCount>availableProcessorCount - 1</moveThreadCount>
  ...
</solver>
```

OptaPlanner 8 中的 'solverConfig.xml'file 示例：

```
<solver>
  ...
  <moveThreadCount>1</moveThreadCount> <!-- Alternatively, use "AUTO" or omit entirely. -
->
  ...
</solver>
```

OptaPlanner 7 中的 benchmarkConfig.xml 文件示例：

```
<plannerBenchmark>
  ...
  <parallelBenchmarkCount>availableProcessorCount - 1</parallelBenchmarkCount>
  ...
</plannerBenchmark>
```

OptaPlanner 8 中的 benchmarkConfig.xml 文件示例：

```
<plannerBenchmark>
  ...
  <parallelBenchmarkCount>1</parallelBenchmarkCount> <!-- Alternatively, use "AUTO" or omit
entirely. -->
  ...
</plannerBenchmark>
```

删除已弃用的变量监听程序

主要,Public API

来自软件包 org.optaplanner.core.impl.domain.variable.listener 已弃用的接口 VariableListener 已被删除，以及已弃用的 interface StatefulVariableListener 以及同一软件包中的已弃用的类 VariableListenerAdapter。改为使用 org.optaplanner.core.api.domain.variable 软件包中的 VariableListener 接口。

OptaPlanner 7 中的 VariableListener.java 文件示例：

```
...
import org.optaplanner.core.impl.domain.variable.listener.VariableListenerAdapter;
...

public class MyVariableListener extends VariableListenerAdapter<Object> {

  ...

  @Override
  void afterEntityRemoved(ScoreDirector scoreDirector, Object entity);
    ...
  }

  ...
}
```

OptaPlanner 8 中的 VariableListener.java 文件示例：

```
...
import org.optaplanner.core.api.domain.variable.VariableListener;
...

public class MyVariableListener extends VariableListener<MySolution, Object> {

  ...

  @Override
  void afterEntityRemoved(ScoreDirector<MySolution> scoreDirector, Object entity);
    ...
  }

  ...
}
```

OptaPlanner 7 中的 StatefulVariableListener.java 文件示例：

```
...
import org.optaplanner.core.impl.domain.variable.listener.StatefulVariableListener;
...

public class MyStatefulVariableListener implements StatefulVariableListener<Object> {

  ...

  @Override
  public void clearWorkingSolution(ScoreDirector scoreDirector) {
    ...
```

```
  }

  ...
}
```

OptaPlanner 8 中的 StatefulVariableListener.java 文件示例：

```
...
import org.optaplanner.core.api.domain.variable.VariableListener;
...

public class MyStatefulVariableListener implements VariableListener<MySolution, Object> {

  ...

  @Override
  public void close() {
    ...
  }

  ...
}
```

第 2 章 OPTAPLANNER 8.2.0 和 OPTAPLANNER 8.3.0 间的更改

本节中列出的更改在 OptaPlanner 8.2.0 和 OptaPlanner 8.3.0 之间进行。

ConstraintMatch.compareTo（） 与 equals（）不一致

次

ConstraintMatch 中的 equals（） 覆盖已被删除。因此，两个不同的 ConstraintMatch 实例永远不会被视为相等。这与 compareTo（） 方法相对，如果所有字段值都相等，它会继续考虑两个实例相等。

部分 II. 开始使用红帽构建的 OPTAPLANNER

作为业务规则开发人员，您可以使用红帽构建的 OptaPlanner 来查找最佳解决方案，以根据一组有限的资源以及在特定限制约束下规划问题。

使用本文档开始使用 OptaPlanner 解决问题。

第 3 章 红帽构建的 OPTAPLANNER 简介

OptaPlanner 是一个轻量级、可嵌入的规划引擎，可优化计划问题。它帮助普通 Java 编程人员有效地解决规划问题，它可将优化性与风险计算相结合。

例如，OptaPlanner 帮助解决各种用例：

- *staff/Patient Ros* ters ：它还有助于为 nurses 创建定时表并跟踪病人管理。
- *教育*时间表：它有助于安排课程、课程、考试和会议演示。
- *shop Schedules* ：它跟踪车装行、计算机队列计划和 workforce 任务规划。
- *危机* ：通过减少资源消耗（如文章和发证）来最小化浪费。

每个组织都面临规划问题；也就是说，它们提供有限的资源（员工、资产、时间和金钱）。

OptaPlanner 是 Apache 软件许可证 2.0 下的开源软件。它是 100% 的纯 Java，在大多数 Java 虚拟机(JVM)上运行。

3.1. 计划问题

根据有限的资源及特定限制，*计划问题*是一种最佳目标。最佳目标可以是任何数量，例如：

- 最大化利润 - 实现最大利润最佳目标。
- 最小化原则占用 - 最佳目标具有最低程度的环境影响。
- 最大化员工或客户的满意度 - 最佳目标将优先考虑员工或客户的需求。

实现这些目标的能力取决于可用的资源数量。例如，以下资源可能会受限制：

- 人员数量
- 时间量
- budget
- 物理资产，如机器、车、车、计算机、构建

您还必须考虑与这些资源相关的特定限制，如个人工作小时数、使用某些机器或其它设备间的兼容性。

红帽构建的 OptaPlanner 帮助 Java 编程人员有效地解决约束满意度问题。它将优化 heuristics 和 metaheuristics 合并起来，且有有效的分数计算。

3.2. 规划问题中的 NP 完整性

提供的用例 *可能是* NP-complete 或 NP-hard，这意味着应用以下语句：

- 在合理时间，轻松地验证特定解决方案以解决问题。
- 在合理时间无法找到问题的最佳解决方案。

含义在于解决您的问题可能比您预期的问题更难，因为两个常见的技术并不知道：

- brute 强制算法（即使更高级的变体）过长。
- 例如，一个快速算法，例如在 组合问题 *中，首先放入最大项目*，这是目前从最佳效果最远的解决方案。

通过使用高级优化算法，OptaPlanner 在解决此类计划问题的适当解决方案。

3.3. 用于规划问题的解决方案

计划问题有很多解决方案。

多种解决方案类别如下：

可能的解决方案

可能的解决方法是任何解决方案，无论它是否会破坏任何数量的限制。规划问题通常存在大量可能的解决方案。其中很多解决方案都非常有用。

可行的解决方案

可行的解决方案是不会破坏任何（负）硬约束的解决方案。可行的解决方案数量相对于可能的解决方案。有时没有可行的解决方案。每个可行的解决方案都是一个可能的解决方案。

最佳解决方案

最佳解决方案是具有最高分数的解决方案。规划问题通常具有较少的最佳解决方案。即使没有可行的解决方案，它们始终只有一个最佳解决方案，且最佳解决方案并不可行。

找到最佳解决方案

最好的解决方法是解决方案，在指定时间内按实施获得的最高分值。找到的最佳解决方案可能可行，并有足够的时间，这是最佳的解决方案。

Counterintuively，可能的解决方案数量非常大（如果被正确计算），即使是很小的数据集。

在 planner-engine 分发文件夹中提供的示例中，大多数实例都有大量可能的解决方案。由于无法获得最佳解决方案，因此任何实施都强制评估所有可能的解决方案的子集。

OptaPlanner 支持一些优化算法，以便高效地通过这些算法来有效地处理大量可能的解决方案。

根据用例，某些优化算法会比其他性能更好，但无法预先了解。使用 OptaPlanner，您可以在 XML 或代码的几行中更改解决器配置来切换优化算法。

3.4. 有关规划问题的约束

通常，计划问题至少有两个级别限制：

- *（负）硬约束* 不能出现问题。

  例如，一个老师无法同时教授两个不同的课时。

- 如果可以避免，则应该损坏 *（负）软约束*。

  例如，Teacher A 不希望在星期五下午进行教学。

有些问题也有正的约束：

- 如果可能，应达到 *正的软约束（或好处）*。

  例如，Teacher B 喜欢在上星期一早上教学。

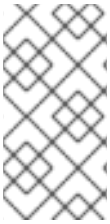
一些基本问题只具有硬限制。有些问题有三个或更多限制级别，如 hard、medium 和 soft 约束。

这些限制定义了计划问题 *的分数计算*（也称为适合性 *功能*）。规划问题的每个解决方案都会以分数为准。使用 OptaPlanner 时，分数限制以面向对象的语言编写，如 Java，或者在 Drools 规则中使用。

这种类型的代码非常灵活，可扩展。

3.5. 红帽构建的 OPTAPLANNER 示例

Red Hat Process Automation Manager 提供了几个 OptaPlanner 示例。您可以检查示例代码，并根据需要对其进行修改，以满足您的需要。



注意

红帽不提供对 Red Hat Process Automation Manager 发行版本中包含的示例代码的支持。

某些 OptaPlanner 示例可以解决问题，这些问题在学术 contests中呈现。下表中的 Contest 列列出了 contests。它还标识了示例，*作为 一项测试的目的或不切实际的。真实的竞争* 测试是满足以下标准的官方独立测试：

- 明确定义实际用例
- 实际限制
- 多个真实数据集
- 在特定硬件的特定时间限制内可重复生成的结果
- 来自学术和/或企业运营研究社区的严重参与。

真实的 Contests 提供了 OptaPlanner 与竞争软件和学术研究的目标比较。

表 3.1. 示例概述

示例	域	大小	Contest	目录名称
N queens	1 个实体类 （1 变量）	**256**个实体 值 **256** 搜索空间 **^616**	无得分 (cheatable)	**nqueens**
云平衡	1 个实体类 （1 变量）	entity abrt **2400** 值 IFL **800** 搜索空间 Warning **10^6967**	否（由我们定义）	**Cloudbalancing**

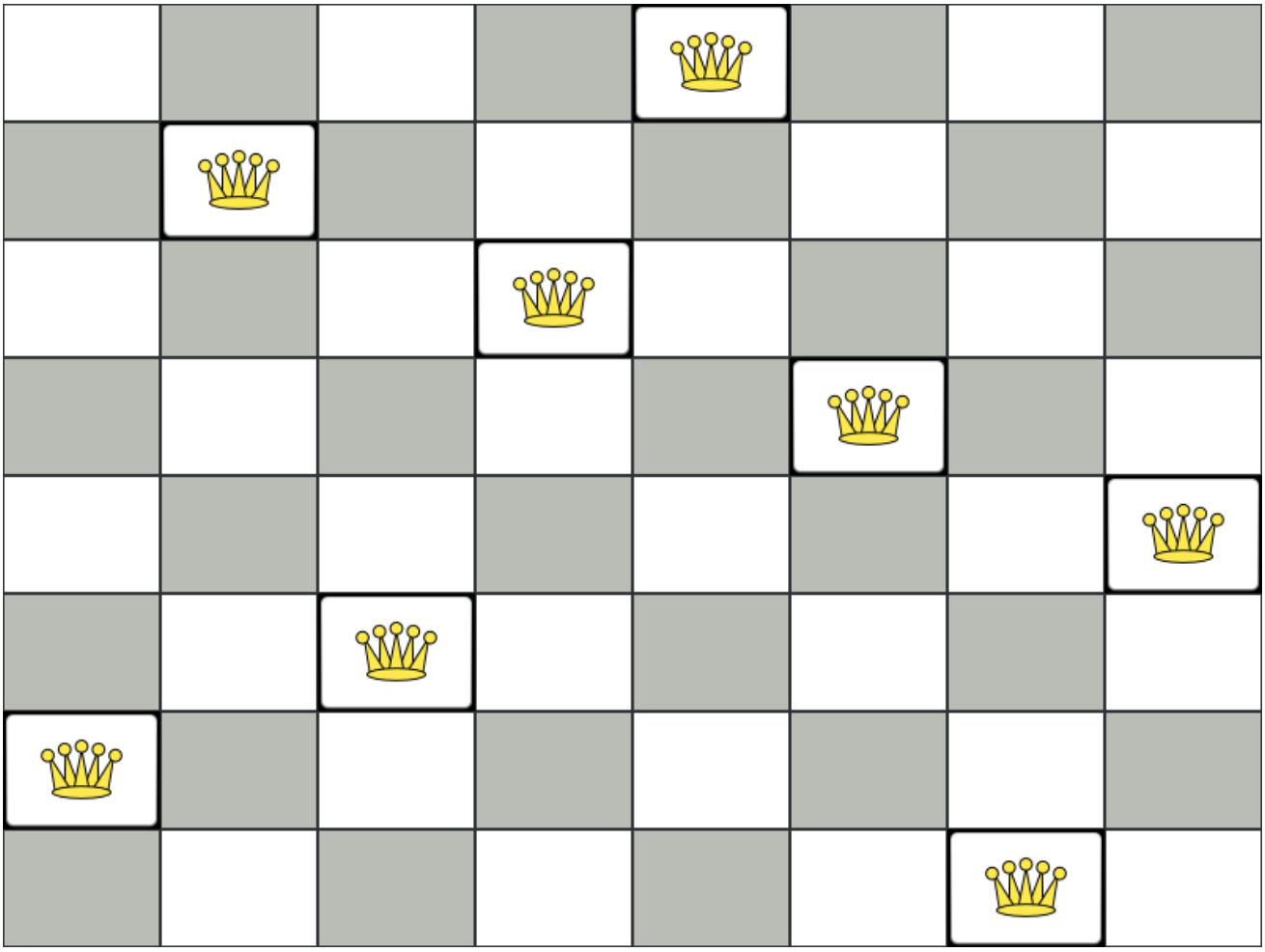
示例	域	大小	Contest	目录名称
traveling salesman	1 个实体类 （1链的变量）	entity dropped **980** 值 InventoryService **980** 搜索空间 768 **10^2504**	不切实际的 TSP web	**tsp**
Tennis club 调度	1 个实体类 （1 变量）	entity abrt **72** 值 InventoryService **7** 搜索空格 Warning **10^60**	否（由我们定义）	**十nis**
会议调度	1 个实体类 （2 变量）	entity abrt **10** 值 abrt **320** 和 时间 **5** 搜索空间 768 **10^320**	否（由我们定义）	**会议阶段**
课程时间表	1 个实体类 （2 变量）	entity Equal **434** 值 abrt **25** 和 时间 **20** 搜索空格 Warning **10^1171**	现实 ITC 2007 年跟踪 3	**curriculumCourse**
机器重新分配	1 个实体类 （1 变量）	entity Equal **50000** 值 **5000** 搜索空间 **^184948**	2012 年几乎真实的 ROADEF	**machineReassignment**
载体路由	1 个实体类 （1链的变量） 1 个影子实体类 （1 个自动影子变量）	entity abrt **2740** 值 InventoryService **2795** 搜索空间 768 **10^8380**	无切实际的 VRP 网络	**vehiclerouting**
使用时间窗 的载设备路由	所有 Vehicle 路由 （1 个影子变量）	entity abrt **2740** 值 InventoryService **2795** 搜索空间 768 **10^8380**	无切实际的 VRP 网络	**vehiclerouting**

示例	域	大小	Contest	目录名称
项目作业调度	1 个实体类 （2 变量） （1 个影子变量）	entity Equal **640** 值 fsanitize **?** 和 sHistoryLimit **?** 搜索空格 **?**	2013 年几乎真实的 MISTA	**projectjobscheduling**
任务分配	1 个实体类 （1链的变量） （1 个影子变量） 1 个影子实体类 （1 个自动影子变量）	entity Equal **500** 值 InventoryService **520** 搜索空间 768 **10^1168**	没有被我们定义	**taskassigning**
考试时间表	2 个实体类（层次结构） （2 变量）	entity abrt **1096** 值 InventoryService **80** 和 时间 **49** 搜索空格 >_< **10^3374**	真实的 ITC 2007 年跟踪 1	**考试项目**
Nurse rostering	1 个实体类 （1 变量）	entity abrt **752** 值 IFL **50** 搜索空间 768 **10^1277**	现实 INRC 2010	**nurserostering**
traveling tournament	1 个实体类 （1 变量）	entity abrt **1560** 值 sHistoryLimit **78** 搜索空间 Warning **10^2301**	不切实际的 TTP	**travelingtournament**
更低的时间调度	1 个实体类 （2 变量）	entity Equal **500** 值 InventoryService **100** 和 时间 **288** 搜索空间 768 **10^20078**	几乎真实的 ICON Energy	**cheaptimescheduling**
投资	1 个实体类 （1 变量）	entity abrt **11** 值 = **1000** 搜索空间 768 **10^4**	没有被我们定义	**投资**

示例	域	大小	Contest	目录名称
会议调度	1 个实体类 （2 变量） 	IFL **216** 值 abrt **18** 和 时间 **20** 搜索空间 768 **10^552**	没有被我们定义	**会议阶段**
sttour	1 个实体类 （1链的变量） （4 个影子变量） 1 个影子实体类 （1 个自动影子变量）	entity dropped **47** 值 abrt **48** 搜索空格 768 **10^59**	没有被我们定义	**sttour**
flight crew 调度	1 个实体类 （1 变量） 1 个影子实体类 （1 个自动影子变量）	entity dropped **4375** 值 abrt **750** 搜索空间 2.4 **10^12578**	没有被我们定义	**flightcrewschedulin g**

3.6. N QUEENS

在 *n* 个大小小板上放置 *n* queens，以便无法互相攻击。最常见的 *n* queens puzzle 是 8 个 queuzzle，有 *n = 8* ：



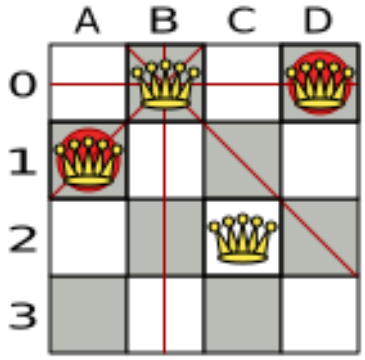
约束：

- 使用主板的 *n* 列和 *n* 行。
- 在主板上放置 *n* queens。
- 无法相互攻击两个频率。queen 可攻击同一横向、垂直或部门其他任何频率的其他频率。

本文档主要使用四个不同点。

建议的解决方案可能是：

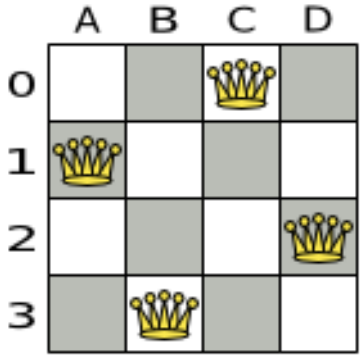
图 3.1. 四点欺诈解决方案



以上解决方案是错误的，因为 queens A1 和 B0 可以相互攻击（因此可以说 B0 和 D0）。删除 queen B0 会尊重"不两种 queens"约束，但会破坏 "place *n* queens" 约束。

以下是一个正确的解决方案：

图 3.2. Four quele queuzzle 的正确解决方案



所有约束都已满足，因此解决方案正确。

请注意，大多数 *n* 个词汇都有多个正确的解决方案。我们将重点介绍查找特定 *n* 的单一正确的解决方案，而不是为特定 *n* 查找可能的正确解决方案。

问题大小

```
4queens   has   4 queens with a search space of    256.
8queens   has   8 queens with a search space of   10^7.
16queens  has  16 queens with a search space of  10^19.
32queens  has  32 queens with a search space of  10^48.
64queens  has  64 queens with a search space of 10^115.
256queens has 256 queens with a search space of 10^616.
```

N queens 示例的实现还没有优化，因为它作为新手示例的功能。*然而，它可以轻松地处理* ***64 queens****。出现了一些变化，它已被显示，可轻松处理* ***5000 queens*** *等等。*

3.6.1. N queens 的域模型

这个示例使用域模型解决四条问题。

- *创建域模型*

  好的域模型可以方便理解和解决您的规划问题。

  这是 ***n queens*** *示例的域模型：*

```
public class Column {

    private int index;

    // ... getters and setters
}
```

```
public class Row {

    private int index;

    // ... getters and setters
}
```

```
public class Queen {

    private Column column;
    private Row row;

    public int getAscendingDiagonalIndex() {...}
    public int getDescendingDiagonalIndex() {...}

    // ... getters and setters
}
```

- *计算搜索空间.*

Queen 实例有一个 Column （例如：0 为列 A， 1 为列 B, ...）和一个 Row （its 行， 例如 0 代表行 0， 1 是行 1, ...）。

可以根据列和行计算升序行和降序行。

列和行索引从主板的左上角开始。

```
public class NQueens {

    private int n;
    private List<Column> columnList;
    private List<Row> rowList;

    private List<Queen> queenList;

    private SimpleScore score;

    // ... getters and setters
}
```

- *查找解决方案*

单个 NQueens 实例包含所有 Queen 实例的列表。它是 Solution 实施， 它将提供给、被解决并从 Solver 中检索。

请注意， 在四个 queens 示例中， NQueens getN（） 方法总是返回 4。

图 3.3. Four Queens 的解决方案

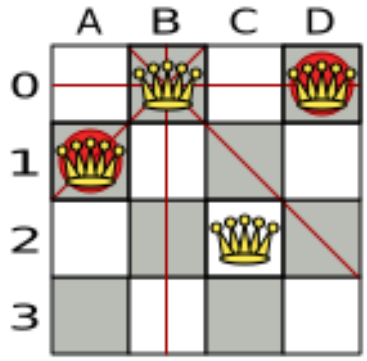


表 3.2. 域模型中的解决方案详情

	columnIndex	rowIndex	ascendingDiagonalIndex (columnIndex + rowIndex)	descendingDiagonalIndex (columnIndex - rowIndex)
A1	0	1	**1 (**)**	-1
B0	1	**0 (*)**	**1 (**)**	1
C2	2	2	4	0
D0	3	**0 (*)**	3	3

*当两个 **queens** 共享同一列时，行或横线，如(*)和(**)，它们可以相互攻击。*

3.7. 云平衡

*有关本示例的详情，请参阅 **Red Hat build of OptaPlanner quick start Guide**。*

3.8. TRAVELING SALESMAN（TSP - TRAVELING SALESMAN 问题）

*给定城市列表，找到一个只访问每个城市的 **salesman** 最短导览。*

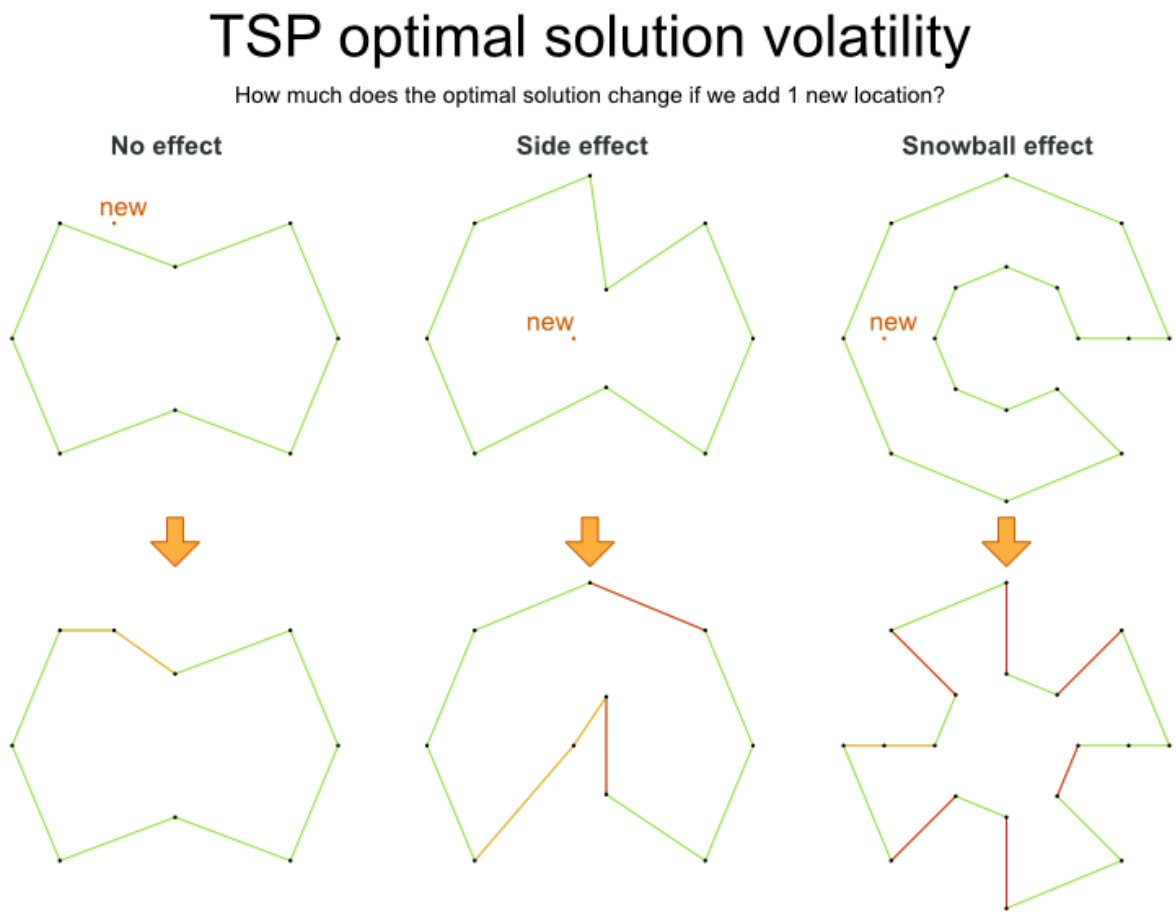
*该问题由 **Wikipedia** 定义。它是计算 计算中最广泛调查的问题之一。然而，在现实世界中，这通常只是规划问题的一部分，以及其他约束，如员工切换限制。*

问题大小

> *dj38 has 38 cities with a search space of 10^43.*
> *europe40 has 40 cities with a search space of 10^46.*
> *st70 has 70 cities with a search space of 10^98.*
> *pcb442 has 442 cities with a search space of 10^976.*
> *lu980 has 980 cities with a search space of 10^2504.*

问题困难

*尽管 **TSP** 的简单定义，但问题很难解决。因为这是一个 **NP-hard** 问题（如大多数规划问题），当问题数据集中稍有改变时，特定问题 **dataset** 的最佳解决方案可能会改变：*



3.9. TENNIS CLUB 调度

每周 **10nis club** 都有四个团队相互推断。将这四个点分配给团队。

硬约束：

- 冲突：团队每天只能每周一次。
- 不可用：一些团队在一些日期不可用。

Medium 约束：

- 公平分配：所有团队都应扮演（几乎）相同的时间。

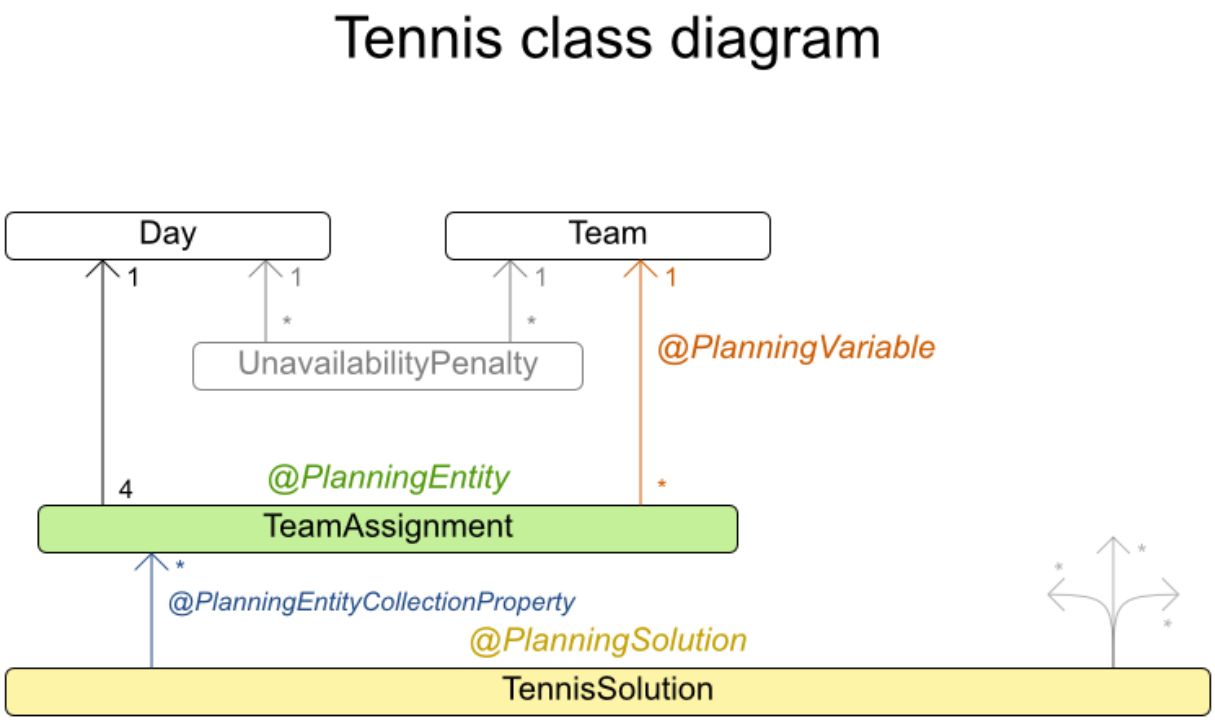
软限制：

- *均匀的前端：每个团队应针对其他每个团队都有一个相同的次数。*

问题大小

> *munich-7teams has 7 teams, 18 days, 12 unavailabilityPenalties and 72 teamAssignments with a search space of 10^60.*

*图 **3.4.** 域模型*



***3.10.** 会议调度*

为启动时间和房间分配每个会议。会议有不同的持续时间。

硬约束：

- *间冲突：两个会议不能同时使用同一空间。*
- *所需参与者：一个人在同时无法进行两个所需的会议。*
- *必需的房间容量：会议不能适合于所有会议的参与者。*
- *在同一天开始和结束：无法安排在多天的会议。*

***Medium** 约束：*

- *首选参与人：一个人不能同时拥有两个首选会议，而且是首选，而且是需要同时进行的会议。*

软限制：

- *更快而不是之后：尽快计划所有会议。*
- *会议间的休息：任何两个会议至少应在这两间中断。*
- *重叠会议：为了尽量减少并行会议的数量，用户不必选择另外一种会议。*
- *首先分配较大的空间：如果有一个大的房间可用，应该分配给该会议，以便适应尽可能多的人，即使他们尚未签到该会议。*
- *房间稳定性：如果一个人连续有两个或小时间会议，它们之间的时间较差在同一房里。*

问题大小

> *50meetings-160timegrains-5rooms has 50 meetings, 160 timeGrains and 5 rooms with a search space of 10^145.*
> *100meetings-320timegrains-5rooms has 100 meetings, 320 timeGrains and 5 rooms with a search space of 10^320.*
> *200meetings-640timegrains-5rooms has 200 meetings, 640 timeGrains and 5 rooms with a search space of 10^701.*
> *400meetings-1280timegrains-5rooms has 400 meetings, 1280 timeGrains and 5 rooms with a search space of 10^1522.*
> *800meetings-2560timegrains-5rooms has 800 meetings, 2560 timeGrains and 5 rooms with a search space of 10^3285.*

3.11. 课程时间表（ITC 2007 年跟踪 3 - 日程表课程安排）

安排每个授课内容进入一个时间，并入一个房间。

硬约束：

- *老师冲突：老师不能在同一期限内没有两个讲义。*
- *课程冲突：课程不能在同一时间内拥有两个讲义。*
- *房间：两个讲话不能在同一时间段内。*
- *不可用周期（为每个数据集指定）：必须将特定的课程分配给特定的时间段。*

软限制：

- *房间容量：房间容量不应少于学员在演讲中的数量。*
- *最小工作日：在同一课程的讲义应分发至最少的天数。*

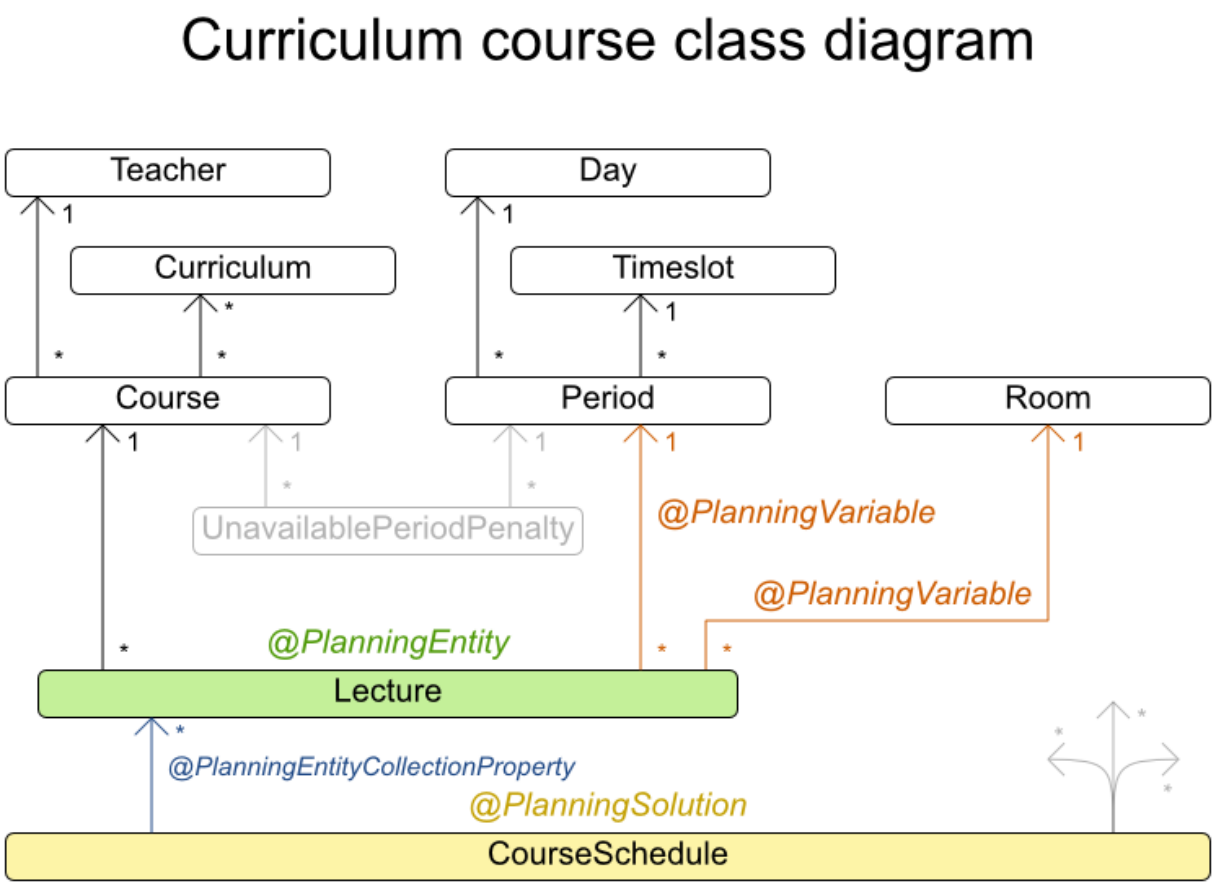
- 课程紧凑：属于相同课程的演讲应相互相邻（在连续的期间内）。
- 房间稳定性：指示相同课程的演讲应该分配到相同的房间。

这个问题由 国际时间选项卡 2007 年跟踪 3 定义。

问题大小

> comp01 has 24 teachers, 14 curricula, 30 courses, 160 lectures, 30 periods, 6 rooms and 53 unavailable period constraints with a search space of 10^360.
> comp02 has 71 teachers, 70 curricula, 82 courses, 283 lectures, 25 periods, 16 rooms and 513 unavailable period constraints with a search space of 10^736.
> comp03 has 61 teachers, 68 curricula, 72 courses, 251 lectures, 25 periods, 16 rooms and 382 unavailable period constraints with a search space of 10^653.
> comp04 has 70 teachers, 57 curricula, 79 courses, 286 lectures, 25 periods, 18 rooms and 396 unavailable period constraints with a search space of 10^758.
> comp05 has 47 teachers, 139 curricula, 54 courses, 152 lectures, 36 periods, 9 rooms and 771 unavailable period constraints with a search space of 10^381.
> comp06 has 87 teachers, 70 curricula, 108 courses, 361 lectures, 25 periods, 18 rooms and 632 unavailable period constraints with a search space of 10^957.
> comp07 has 99 teachers, 77 curricula, 131 courses, 434 lectures, 25 periods, 20 rooms and 667 unavailable period constraints with a search space of 10^1171.
> comp08 has 76 teachers, 61 curricula, 86 courses, 324 lectures, 25 periods, 18 rooms and 478 unavailable period constraints with a search space of 10^859.
> comp09 has 68 teachers, 75 curricula, 76 courses, 279 lectures, 25 periods, 18 rooms and 405 unavailable period constraints with a search space of 10^740.
> comp10 has 88 teachers, 67 curricula, 115 courses, 370 lectures, 25 periods, 18 rooms and 694 unavailable period constraints with a search space of 10^981.
> comp11 has 24 teachers, 13 curricula, 30 courses, 162 lectures, 45 periods, 5 rooms and 94 unavailable period constraints with a search space of 10^381.
> comp12 has 74 teachers, 150 curricula, 88 courses, 218 lectures, 36 periods, 11 rooms and 1368 unavailable period constraints with a search space of 10^566.
> comp13 has 77 teachers, 66 curricula, 82 courses, 308 lectures, 25 periods, 19 rooms and 468 unavailable period constraints with a search space of 10^824.
> comp14 has 68 teachers, 60 curricula, 85 courses, 275 lectures, 25 periods, 17 rooms and 486 unavailable period constraints with a search space of 10^722.

*图 **3.5.** 域模型*



3.12. 机器重新分配(GOOGLE ROADEF 2012)

*为机器分配每个进程。所有进程已经有原始（未优化）分配。每个进程需要每个资源（如 **CPU** 或 **RAM**）的数量。这是 **Cloud Balancing** 示例的一个更为复杂的版本。*

硬约束：

- *最大容量：不能超过每台机器的每个资源的最大容量。*
- *冲突：同一服务的进程必须在不同的机器上运行。*
- *分布：必须将同一服务的进程分散到各个位置。*
- *依赖项：取决于其他服务的进程必须在其他服务的邻居中运行。*

- 临时使用：有些资源是临时的，可以计入原始机器作为新分配机器的最大容量。

软限制：

- **load**：不应超过每台机器的每个资源的安全容量。
- **balances**：通过平衡每台机器上的可用资源，为以后分配空间。
- 流程移动成本：流程具有移动成本。
- 服务迁移成本：服务具有移动成本。
- 机器迁移成本：将进程从机器 **A** 移到机器 **B** 还有另一个特定于 **A-B** 的移动成本。

这个问题由 **Google ROADEF/EURO Challenge** 定义，**2012** 年。

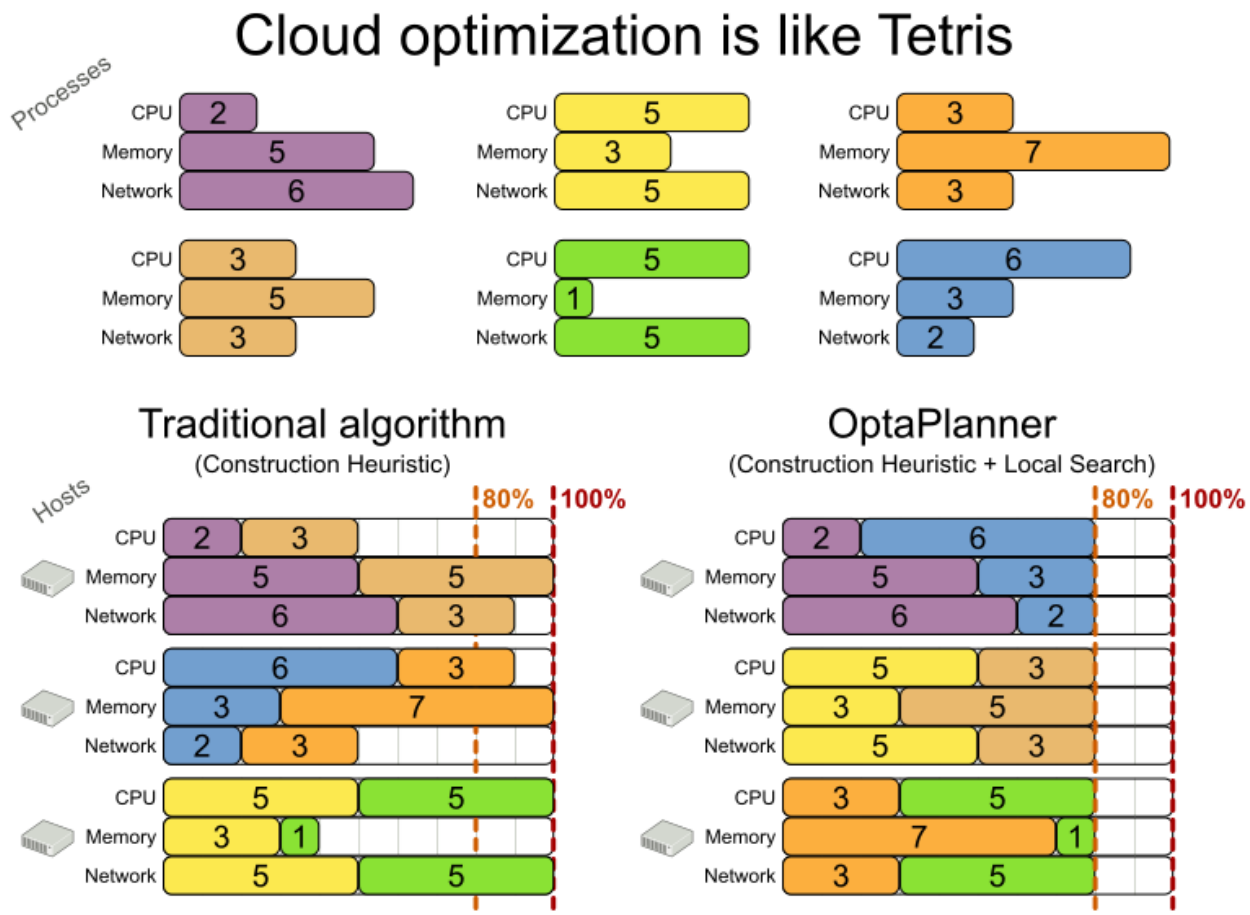
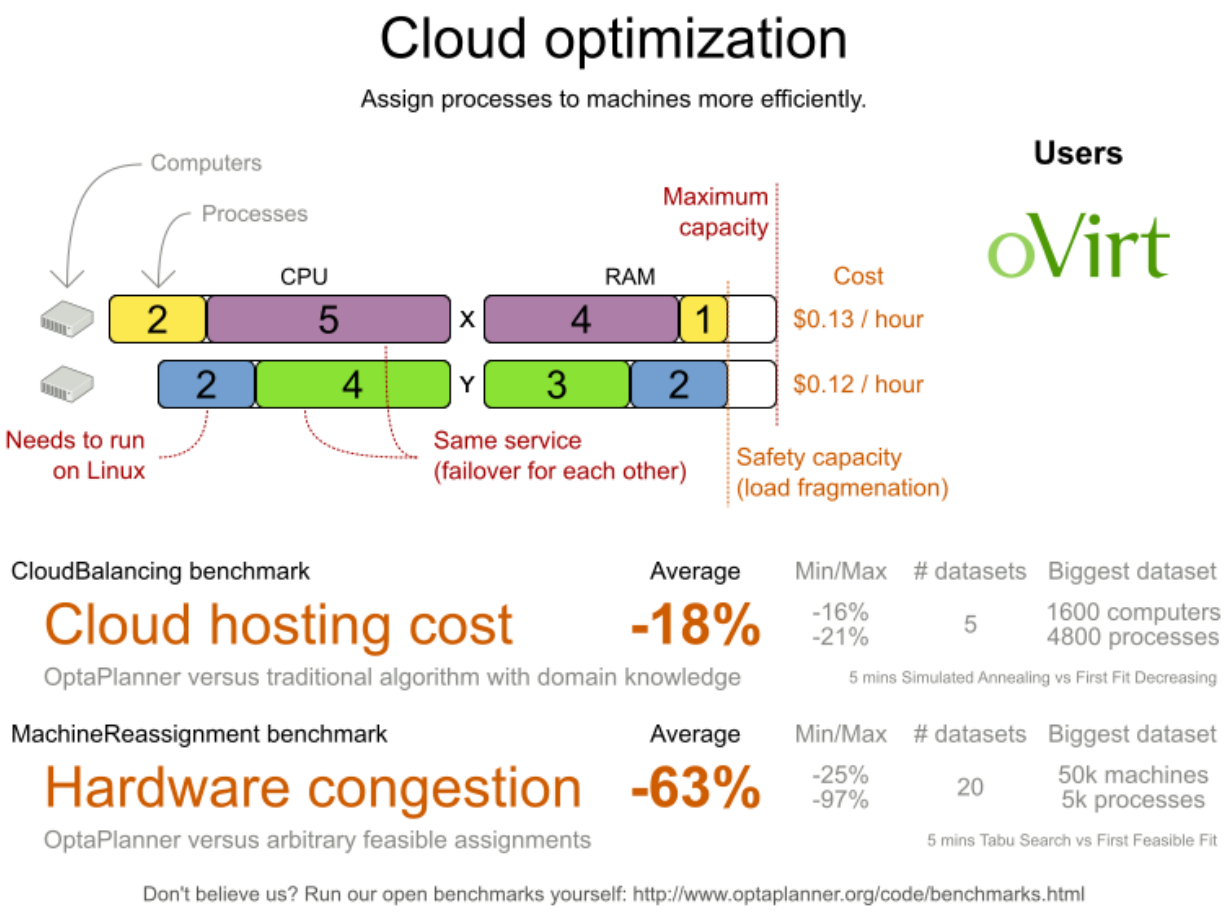

Cloud optimization is like Tetris
Processes
CPU 2
Memory 5
Network 6
CPU 5
Memory 3
Network 5
CPU 3
Memory 7
Network 3
CPU 3
Memory 5
Network 3
CPU 5
Memory 1
Network 5
CPU 6
Memory 3
Network 2
Traditional algorithm
(Construction Heuristic)
Hosts
80%
100%
CPU 2 3
Memory 5 5
Network 6 3
CPU 6 3
Memory 3 7
Network 2 3
CPU 5 5
Memory 3 1
Network 5 5
OptaPlanner
(Construction Heuristic + Local Search)
80%
100%
CPU 2 6
Memory 5 3
Network 6 2
CPU 5 3
Memory 3 5
Network 5 3
CPU 3 5
Memory 7 1
Network 3 5

*图 **3.6.** 价值主张*

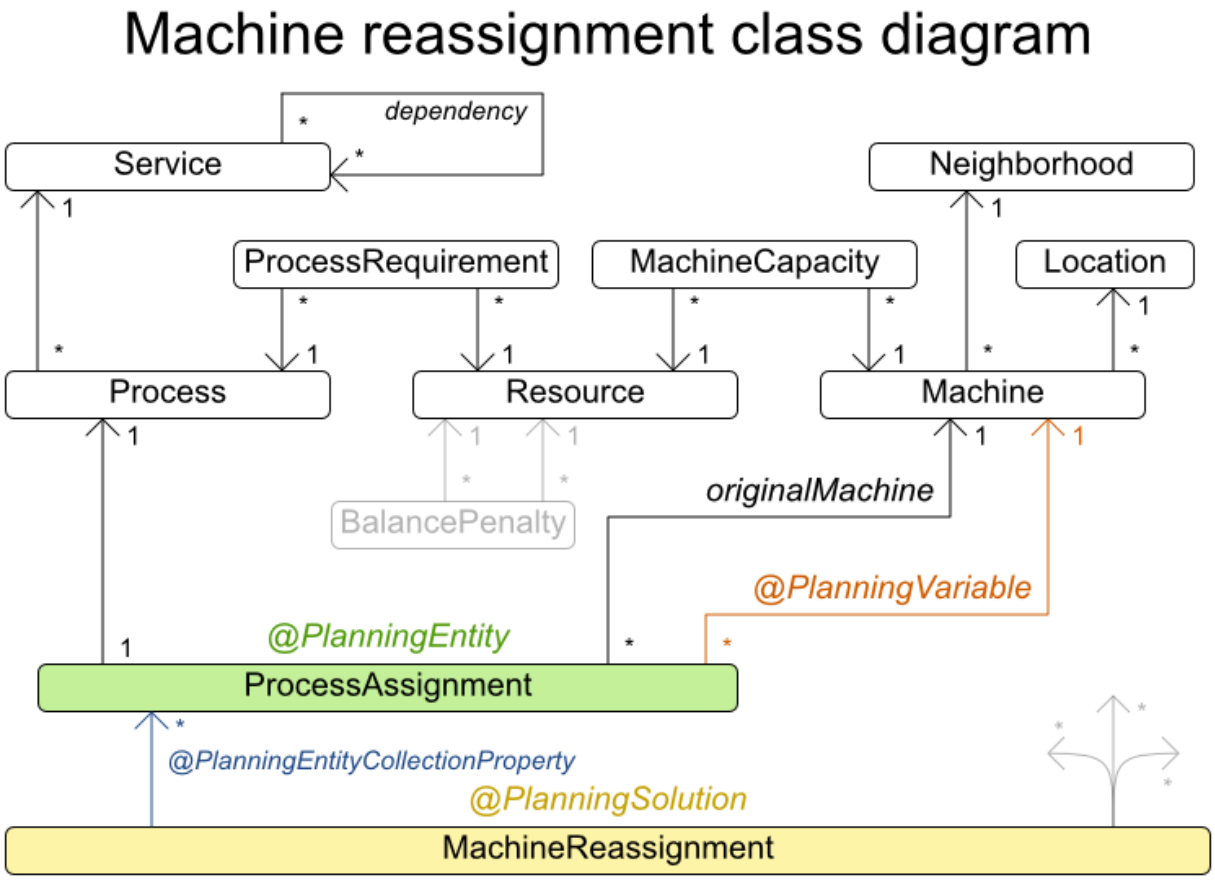


问题大小

> *model_a1_1 has 2 resources, 1 neighborhoods, 4 locations, 4 machines, 79 services, 100 processes and 1 balancePenalties with a search space of 10^60.*
> *model_a1_2 has 4 resources, 2 neighborhoods, 4 locations, 100 machines, 980 services, 1000 processes and 0 balancePenalties with a search space of 10^2000.*
> *model_a1_3 has 3 resources, 5 neighborhoods, 25 locations, 100 machines, 216 services, 1000 processes and 0 balancePenalties with a search space of 10^2000.*
> *model_a1_4 has 3 resources, 50 neighborhoods, 50 locations, 50 machines, 142 services, 1000 processes and 1 balancePenalties with a search space of 10^1698.*
> *model_a1_5 has 4 resources, 2 neighborhoods, 4 locations, 12 machines, 981 services, 1000 processes and 1 balancePenalties with a search space of 10^1079.*
> *model_a2_1 has 3 resources, 1 neighborhoods, 1 locations, 100 machines, 1000 services, 1000 processes and 0 balancePenalties with a search space of 10^2000.*
> *model_a2_2 has 12 resources, 5 neighborhoods, 25 locations, 100 machines, 170 services, 1000 processes and 0 balancePenalties with a search space of 10^2000.*
> *model_a2_3 has 12 resources, 5 neighborhoods, 25 locations, 100 machines, 129 services, 1000 processes and 0 balancePenalties with a search space of 10^2000.*
> *model_a2_4 has 12 resources, 5 neighborhoods, 25 locations, 50 machines, 180 services, 1000 processes and 1 balancePenalties with a search space of 10^1698.*
> *model_a2_5 has 12 resources, 5 neighborhoods, 25 locations, 50 machines, 153 services, 1000 processes and 0 balancePenalties with a search space of 10^1698.*
> *model_b_1 has 12 resources, 5 neighborhoods, 10 locations, 100 machines, 2512 services, 5000*

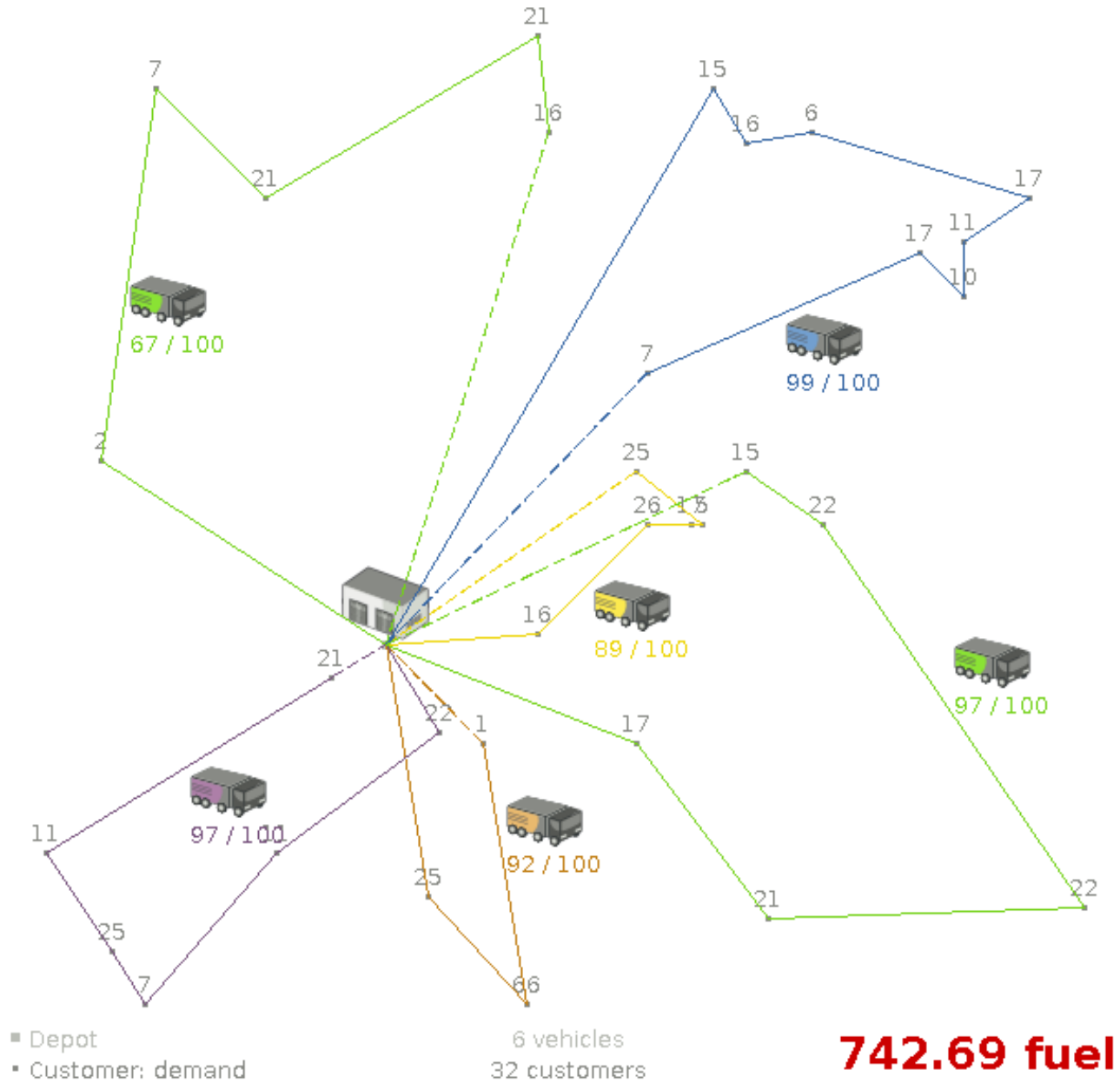
processes and 0 balancePenalties with a search space of 10^10000.
model_b_2 has 12 resources, 5 neighborhoods, 10 locations, 100 machines, 2462 services, 5000 processes and 1 balancePenalties with a search space of 10^10000.
model_b_3 has 6 resources, 5 neighborhoods, 10 locations, 100 machines, 15025 services, 20000 processes and 0 balancePenalties with a search space of 10^40000.
model_b_4 has 6 resources, 5 neighborhoods, 50 locations, 500 machines, 1732 services, 20000 processes and 1 balancePenalties with a search space of 10^53979.
model_b_5 has 6 resources, 5 neighborhoods, 10 locations, 100 machines, 35082 services, 40000 processes and 0 balancePenalties with a search space of 10^80000.
model_b_6 has 6 resources, 5 neighborhoods, 50 locations, 200 machines, 14680 services, 40000 processes and 1 balancePenalties with a search space of 10^92041.
model_b_7 has 6 resources, 5 neighborhoods, 50 locations, 4000 machines, 15050 services, 40000 processes and 1 balancePenalties with a search space of 10^144082.
model_b_8 has 3 resources, 5 neighborhoods, 10 locations, 100 machines, 45030 services, 50000 processes and 0 balancePenalties with a search space of 10^100000.
model_b_9 has 3 resources, 5 neighborhoods, 100 locations, 1000 machines, 4609 services, 50000 processes and 1 balancePenalties with a search space of 10^150000.
model_b_10 has 3 resources, 5 neighborhoods, 100 locations, 5000 machines, 4896 services, 50000 processes and 1 balancePenalties with a search space of 10^184948.

*图 **3.7.** 域模型*



***3.13.** 载体路由*

使用车队车队，选择每个客户的对象并将其带给它。每个车块都可以为多个客户提供服务，但其容量有限。



除了基本情况(CVRP)外，还有一个包含时间窗(CVRPTW)的变体。

硬约束：

- *载体容量：一种车块无法获取更多项目，然后它的容量。*
- *时间窗（仅在 CVRPTW 中）：*

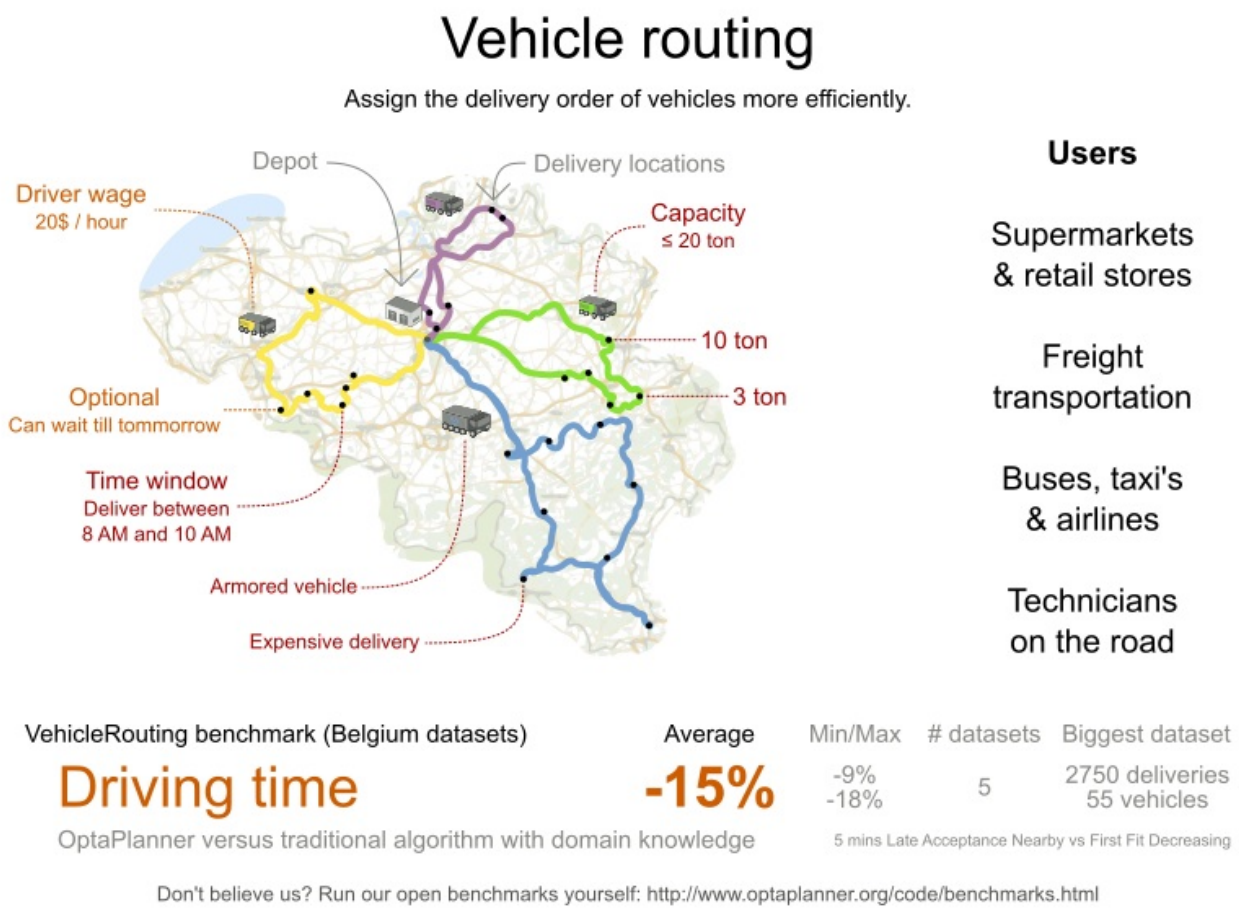
- 差时：从一个位置到另一个位置的差差时间。
- 客户服务持续时间：消费者必须保持在服务期间的长度。
- 客户准备时间：在客户准备时间之前，车辆可能已经进入，但必须等到提供时间后再提供服务。
- 客户经过时限：在客户因时间而进行前，车辆必须经过时间。

软限制：

- 总距离：让所有车上的总距离驱动（精简资源占用率）

VRP Web 定义了容量设施设施路由问题**(CVRP)**及其时间序列**(CVRPTW)**。

*图 **3.8.** 价值主张*



问题大小

***CVRP** 实例（没有时间窗）：*

> *belgium-n50-k10 has 1 depots, 10 vehicles and 49 customers with a search space of 10^74.*
> *belgium-n100-k10 has 1 depots, 10 vehicles and 99 customers with a search space of 10^170.*
> *belgium-n500-k20 has 1 depots, 20 vehicles and 499 customers with a search space of 10^1168.*
> *belgium-n1000-k20 has 1 depots, 20 vehicles and 999 customers with a search space of 10^2607.*
> *belgium-n2750-k55 has 1 depots, 55 vehicles and 2749 customers with a search space of 10^8380.*
> *belgium-road-km-n50-k10 has 1 depots, 10 vehicles and 49 customers with a search space of 10^74.*
> *belgium-road-km-n100-k10 has 1 depots, 10 vehicles and 99 customers with a search space of 10^170.*
> *belgium-road-km-n500-k20 has 1 depots, 20 vehicles and 499 customers with a search space of 10^1168.*
> *belgium-road-km-n1000-k20 has 1 depots, 20 vehicles and 999 customers with a search space of 10^2607.*
> *belgium-road-km-n2750-k55 has 1 depots, 55 vehicles and 2749 customers with a search space of 10^8380.*

belgium-road-time-n50-k10 has 1 depots, 10 vehicles and 49 customers with a search space of 10^74.
belgium-road-time-n100-k10 has 1 depots, 10 vehicles and 99 customers with a search space of 10^170.
belgium-road-time-n500-k20 has 1 depots, 20 vehicles and 499 customers with a search space of 10^1168.
belgium-road-time-n1000-k20 has 1 depots, 20 vehicles and 999 customers with a search space of 10^2607.
belgium-road-time-n2750-k55 has 1 depots, 55 vehicles and 2749 customers with a search space of 10^8380.
belgium-d2-n50-k10 has 2 depots, 10 vehicles and 48 customers with a search space of 10^74.
belgium-d3-n100-k10 has 3 depots, 10 vehicles and 97 customers with a search space of 10^170.
belgium-d5-n500-k20 has 5 depots, 20 vehicles and 495 customers with a search space of 10^1168.
belgium-d8-n1000-k20 has 8 depots, 20 vehicles and 992 customers with a search space of 10^2607.
belgium-d10-n2750-k55 has 10 depots, 55 vehicles and 2740 customers with a search space of 10^8380.

A-n32-k5 has 1 depots, 5 vehicles and 31 customers with a search space of 10^40.
A-n33-k5 has 1 depots, 5 vehicles and 32 customers with a search space of 10^41.
A-n33-k6 has 1 depots, 6 vehicles and 32 customers with a search space of 10^42.
A-n34-k5 has 1 depots, 5 vehicles and 33 customers with a search space of 10^43.
A-n36-k5 has 1 depots, 5 vehicles and 35 customers with a search space of 10^46.
A-n37-k5 has 1 depots, 5 vehicles and 36 customers with a search space of 10^48.
A-n37-k6 has 1 depots, 6 vehicles and 36 customers with a search space of 10^49.
A-n38-k5 has 1 depots, 5 vehicles and 37 customers with a search space of 10^49.
A-n39-k5 has 1 depots, 5 vehicles and 38 customers with a search space of 10^51.
A-n39-k6 has 1 depots, 6 vehicles and 38 customers with a search space of 10^52.
A-n44-k7 has 1 depots, 7 vehicles and 43 customers with a search space of 10^61.
A-n45-k6 has 1 depots, 6 vehicles and 44 customers with a search space of 10^62.
A-n45-k7 has 1 depots, 7 vehicles and 44 customers with a search space of 10^63.
A-n46-k7 has 1 depots, 7 vehicles and 45 customers with a search space of 10^65.
A-n48-k7 has 1 depots, 7 vehicles and 47 customers with a search space of 10^68.
A-n53-k7 has 1 depots, 7 vehicles and 52 customers with a search space of 10^77.
A-n54-k7 has 1 depots, 7 vehicles and 53 customers with a search space of 10^79.
A-n55-k9 has 1 depots, 9 vehicles and 54 customers with a search space of 10^82.
A-n60-k9 has 1 depots, 9 vehicles and 59 customers with a search space of 10^91.
A-n61-k9 has 1 depots, 9 vehicles and 60 customers with a search space of 10^93.
A-n62-k8 has 1 depots, 8 vehicles and 61 customers with a search space of 10^94.
A-n63-k9 has 1 depots, 9 vehicles and 62 customers with a search space of 10^97.
A-n63-k10 has 1 depots, 10 vehicles and 62 customers with a search space of 10^98.
A-n64-k9 has 1 depots, 9 vehicles and 63 customers with a search space of 10^99.
A-n65-k9 has 1 depots, 9 vehicles and 64 customers with a search space of 10^101.
A-n69-k9 has 1 depots, 9 vehicles and 68 customers with a search space of 10^108.
A-n80-k10 has 1 depots, 10 vehicles and 79 customers with a search space of 10^130.
F-n45-k4 has 1 depots, 4 vehicles and 44 customers with a search space of 10^60.
F-n72-k4 has 1 depots, 4 vehicles and 71 customers with a search space of 10^108.
F-n135-k7 has 1 depots, 7 vehicles and 134 customers with a search space of 10^240.

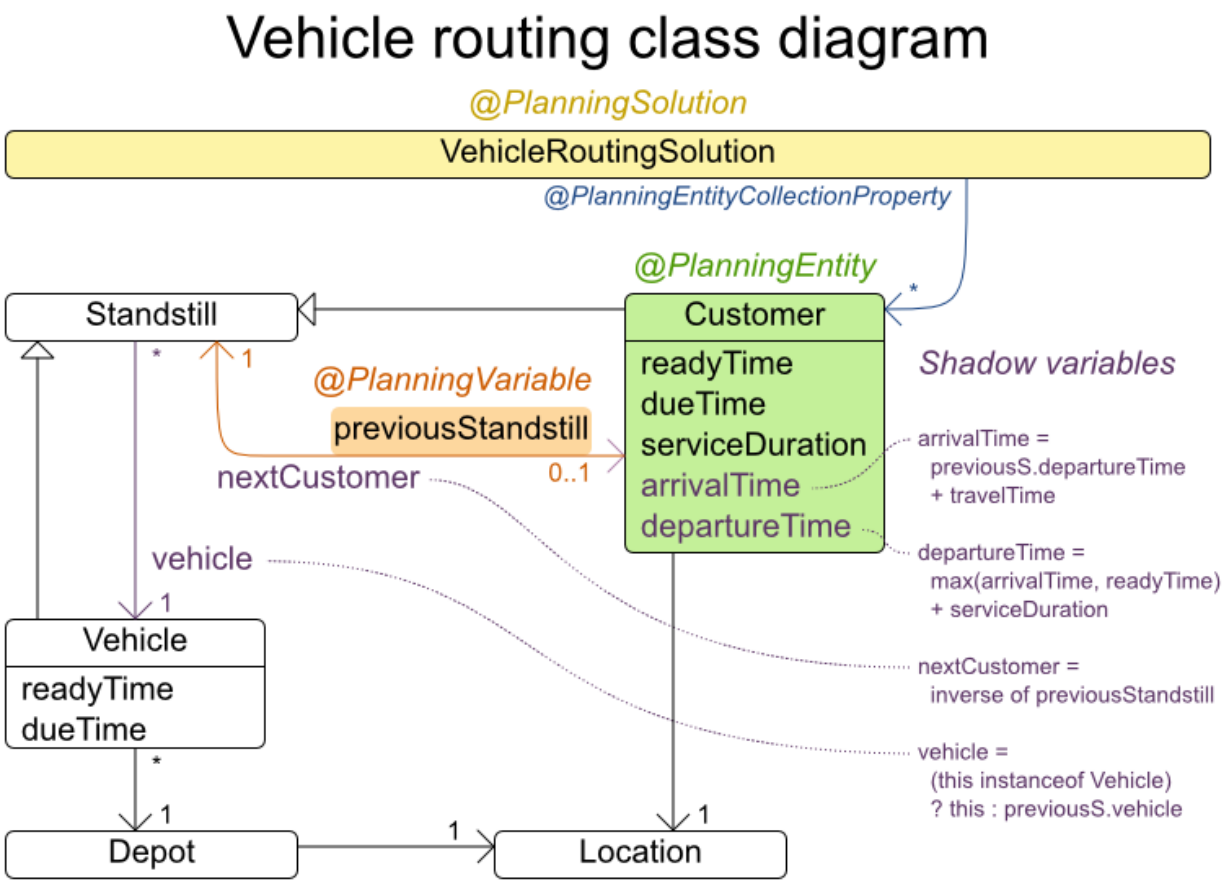
***CVRP** 两个实例（带时间窗）：*

belgium-tw-d2-n50-k10 has 2 depots, 10 vehicles and 48 customers with a search space of 10^74.
belgium-tw-d3-n100-k10 has 3 depots, 10 vehicles and 97 customers with a search space of 10^170.
belgium-tw-d5-n500-k20 has 5 depots, 20 vehicles and 495 customers with a search space of 10^1168.
belgium-tw-d8-n1000-k20 has 8 depots, 20 vehicles and 992 customers with a search space of 10^2607.
belgium-tw-d10-n2750-k55 has 10 depots, 55 vehicles and 2740 customers with a search space of 10^8380.
belgium-tw-n50-k10 has 1 depots, 10 vehicles and 49 customers with a search space of 10^74.
belgium-tw-n100-k10 has 1 depots, 10 vehicles and 99 customers with a search space of 10^170.
belgium-tw-n500-k20 has 1 depots, 20 vehicles and 499 customers with a search space of 10^1168.
belgium-tw-n1000-k20 has 1 depots, 20 vehicles and 999 customers with a search space of 10^2607.
belgium-tw-n2750-k55 has 1 depots, 55 vehicles and 2749 customers with a search space of 10^8380.

Solomon_025_C101 has 1 depots, 25 vehicles and 25 customers with a search space of 10^40.
Solomon_025_C201 has 1 depots, 25 vehicles and 25 customers with a search space of 10^40.
Solomon_025_R101 has 1 depots, 25 vehicles and 25 customers with a search space of 10^40.
Solomon_025_R201 has 1 depots, 25 vehicles and 25 customers with a search space of 10^40.
Solomon_025_RC101 has 1 depots, 25 vehicles and 25 customers with a search space of 10^40.
Solomon_025_RC201 has 1 depots, 25 vehicles and 25 customers with a search space of 10^40.
Solomon_100_C101 has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Solomon_100_C201 has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Solomon_100_R101 has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Solomon_100_R201 has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Solomon_100_RC101 has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Solomon_100_RC201 has 1 depots, 25 vehicles and 100 customers with a search space of 10^185.
Homberger_0200_C1_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.
Homberger_0200_C2_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.
Homberger_0200_R1_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.
Homberger_0200_R2_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.
Homberger_0200_RC1_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of 10^429.
Homberger_0200_RC2_2_1 has 1 depots, 50 vehicles and 200 customers with a search space of

10^429.
Homberger_0400_C1_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.
Homberger_0400_C2_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.
Homberger_0400_R1_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.
Homberger_0400_R2_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.
Homberger_0400_RC1_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.
Homberger_0400_RC2_4_1 has 1 depots, 100 vehicles and 400 customers with a search space of 10^978.
Homberger_0600_C1_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.
Homberger_0600_C2_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.
Homberger_0600_R1_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.
Homberger_0600_R2_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.
Homberger_0600_RC1_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.
Homberger_0600_RC2_6_1 has 1 depots, 150 vehicles and 600 customers with a search space of 10^1571.
Homberger_0800_C1_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.
Homberger_0800_C2_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.
Homberger_0800_R1_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.
Homberger_0800_R2_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.
Homberger_0800_RC1_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.
Homberger_0800_RC2_8_1 has 1 depots, 200 vehicles and 800 customers with a search space of 10^2195.
Homberger_1000_C110_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.
Homberger_1000_C210_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.
Homberger_1000_R110_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.
Homberger_1000_R210_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.
Homberger_1000_RC110_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.
Homberger_1000_RC210_1 has 1 depots, 250 vehicles and 1000 customers with a search space of 10^2840.

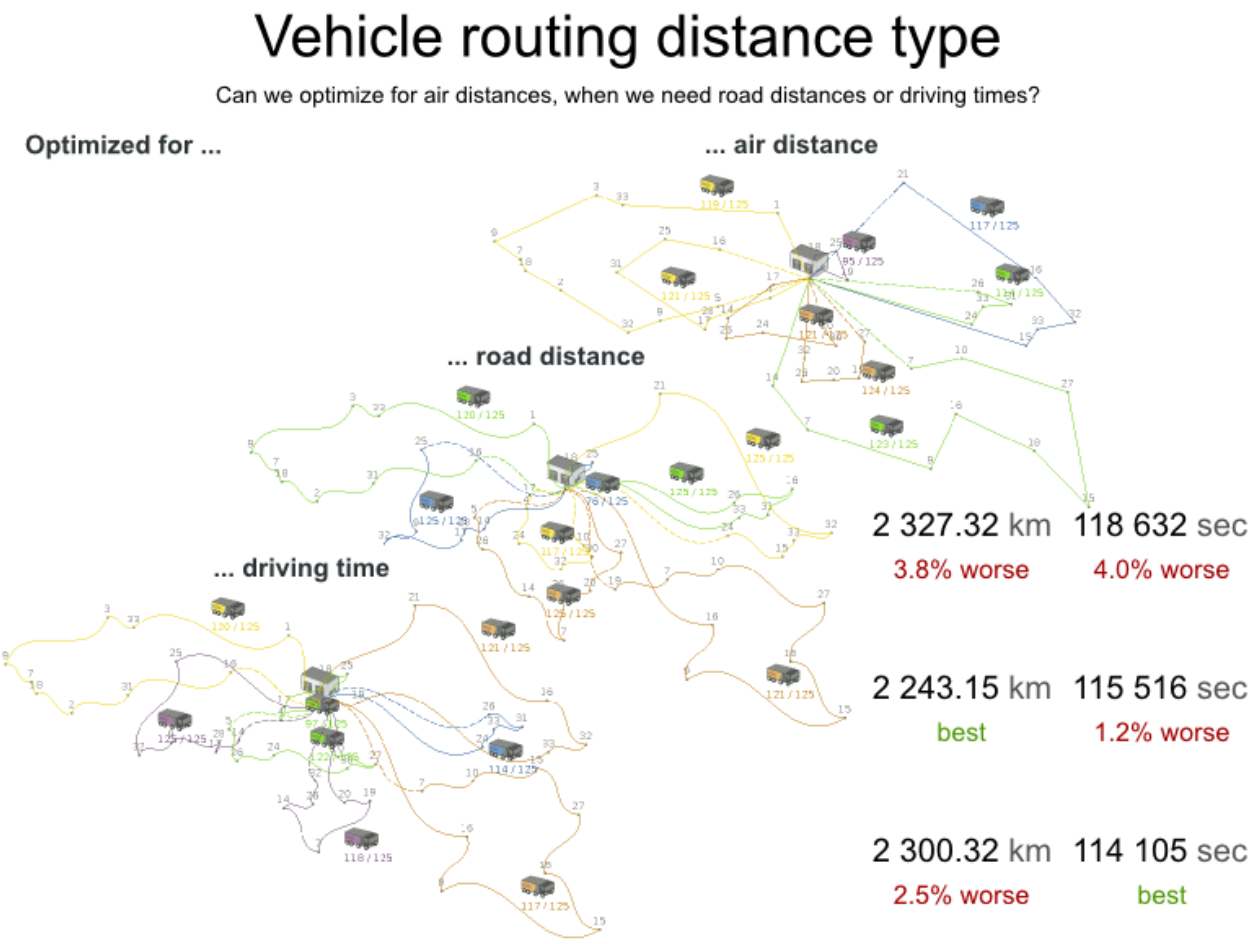
3.13.1. Vehicle 路由的域模型



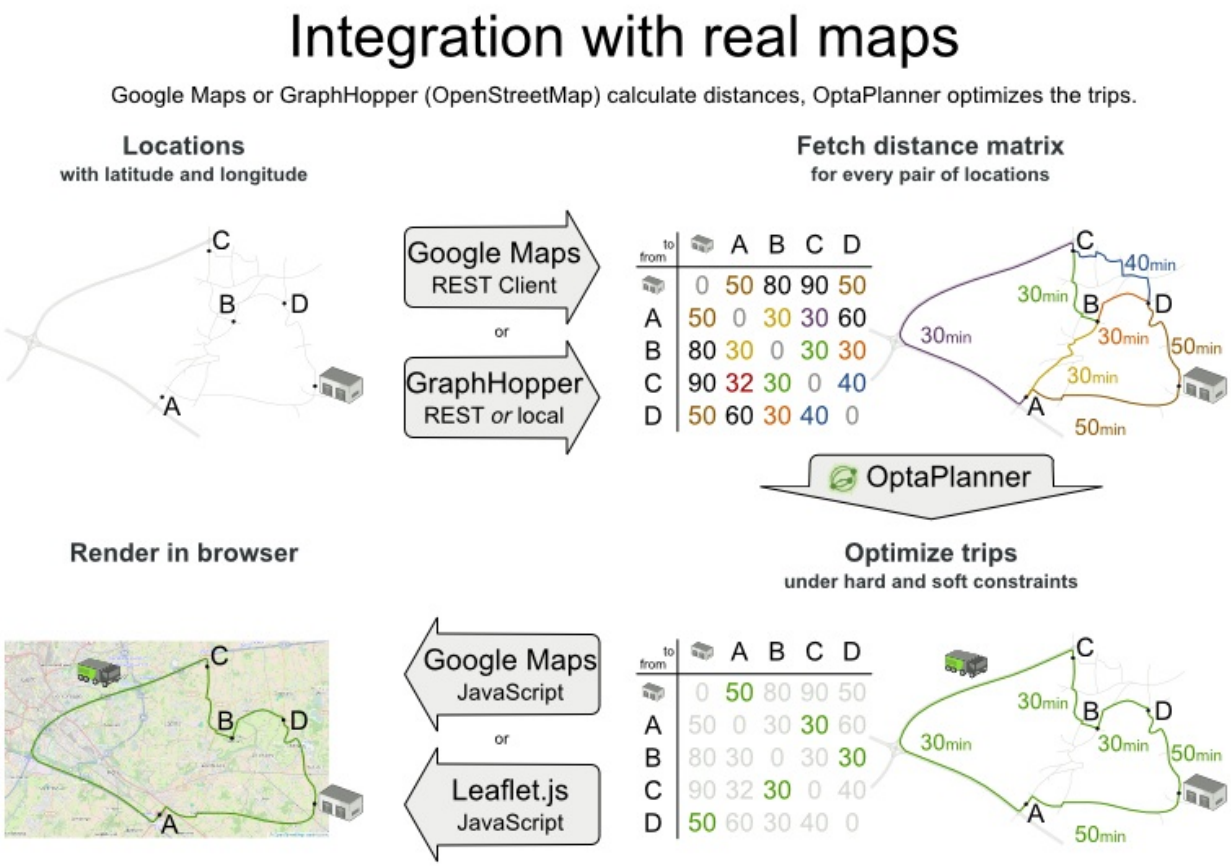
*使用时间窗域模型的载体路由大量使用 **shadow** 变量功能。这使得它能够更自然地表达其限制，因为 **arrivalTime** 和 **departureTime** 等属性可以在域模型上直接可用。*

巴林林特利斯特利斯特区

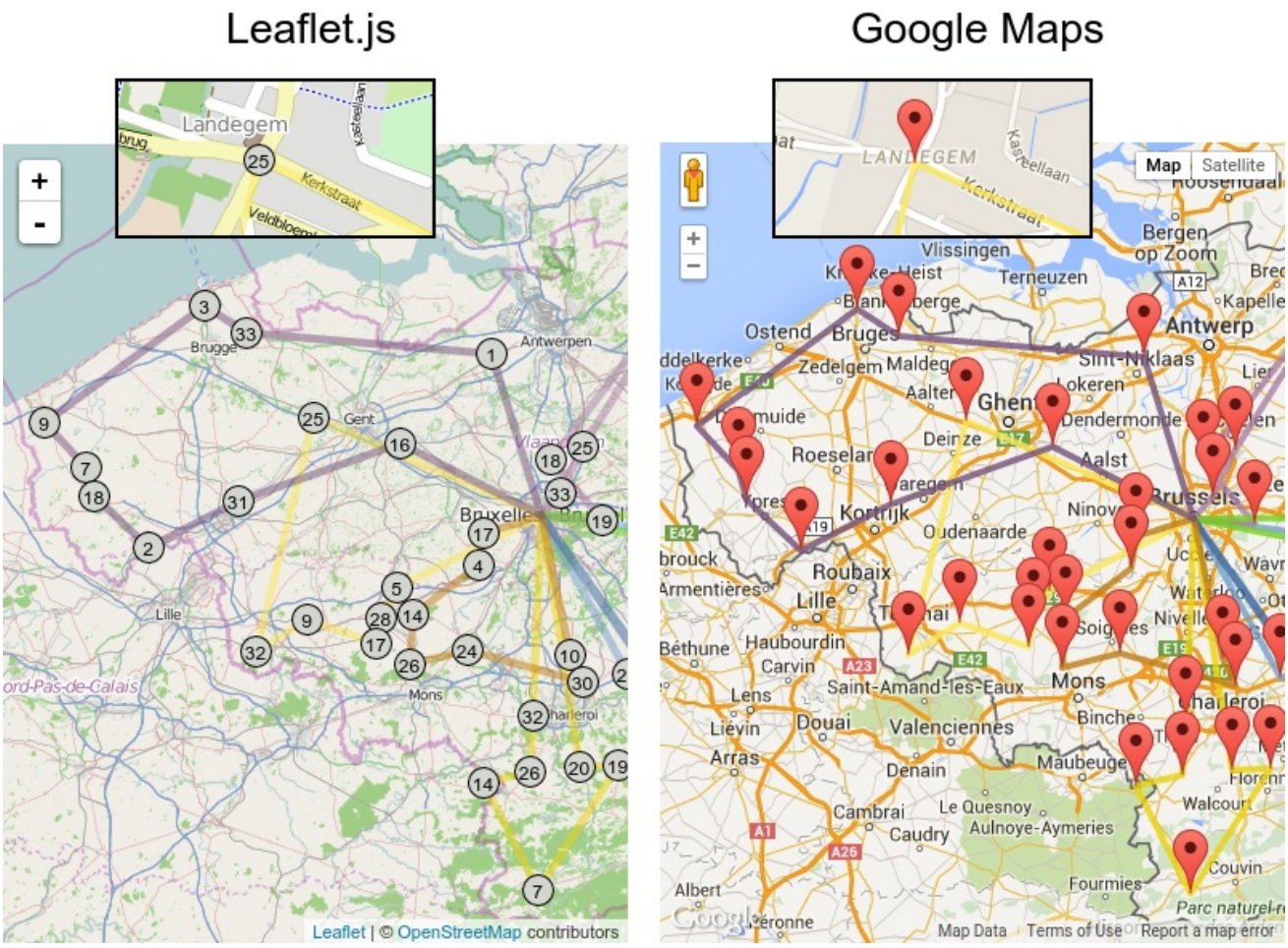
在现实世界中，车辆不能直接跟随位置到位置：他们必须使用路路和高路。从业务的角度来说，这很重要：



对于优化算法，这并不重要，只要可以查找两个点之间的距离（并最好预先计算）。未来成本甚至不需要成为距离。它还可以是差时间、增加成本或更加权的功能。有几种方法可以预先计算出您的成本，如 ***GraphHopper*** *（嵌入式、离线* ***Java*** *引擎）、* ***Open MapQuest*** *（****Web*** *服务）和* ***Google Maps*** *客户端* ***API*** *（****Web*** *服务）。*

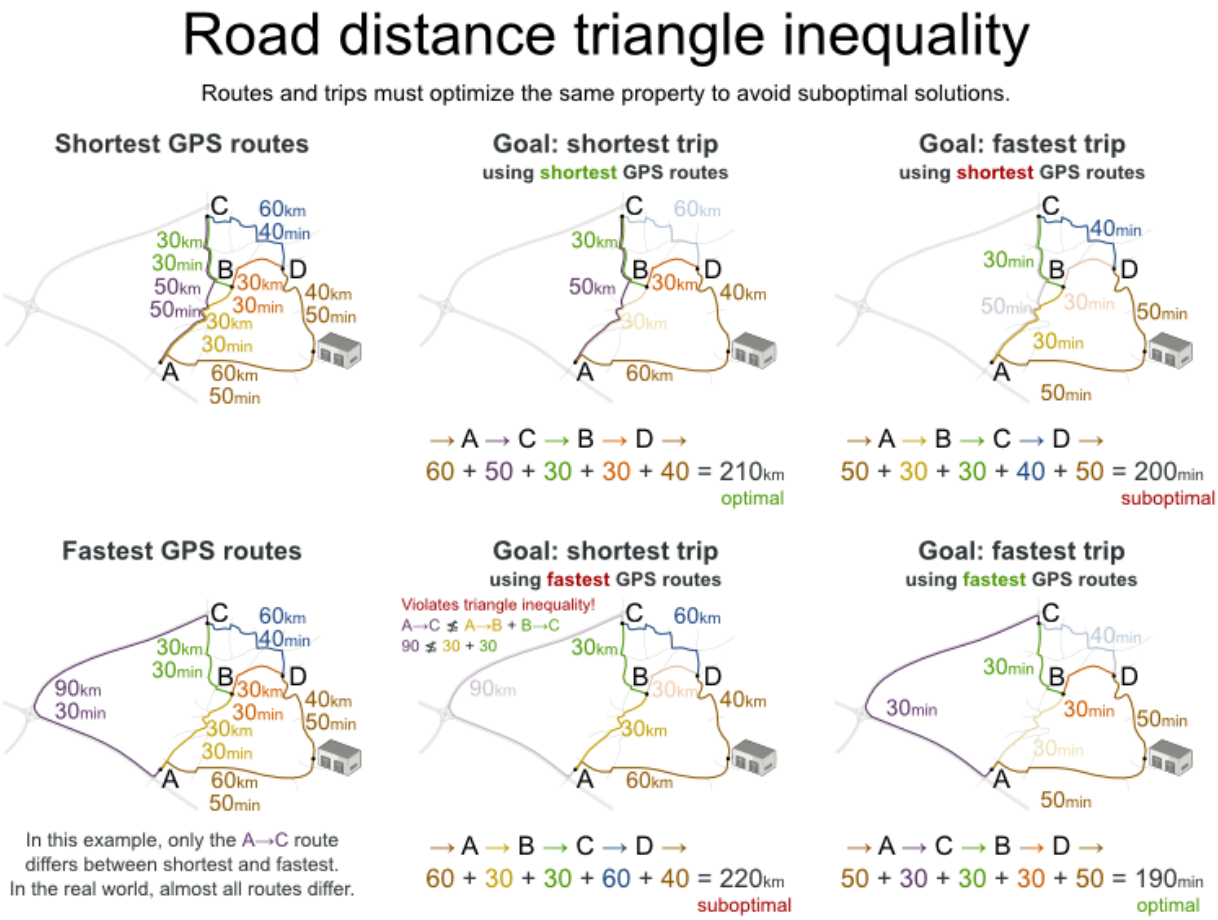


*还有一些技术 可以呈现它，例如针对开发人员的 **Leaflet** 和 **Google Maps**。*



*甚至可以使用 **GraphHopper** 或 **Google Map Directions** 呈现实际规划路由，但是由于路由在高路上的重叠，它可能会变得难以看到反引号：*

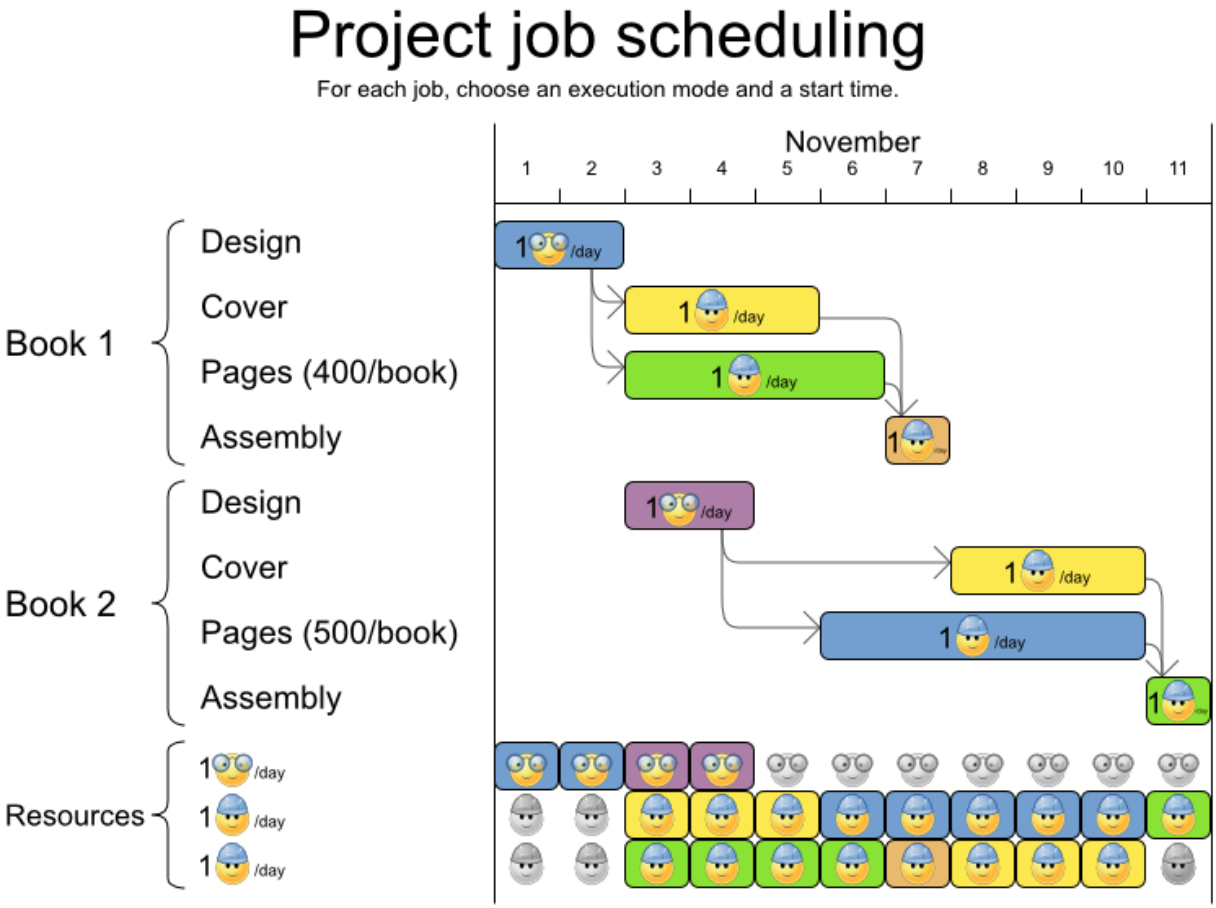
*请特别小心，在两个点之间采用与 **OptaPlanner** 中使用的相同优化标准。例如： **GraphHopper** 默认返回最快的路由，而不是最短的路由。不要使用 **km**（或 **miles**）最快的 **GPS** 路由来优化 **OptaPlanner** 中最短的差差：*



与热门相对应，大多数用户都不希望最短的路由：他们想要使用最快的路由。它们比一般的路高。它们优先于有变化的障碍。在现实世界中，最快且最短的路由很少是相同的。

3.14. 项目作业调度

以时间和执行模式调度所有作业，以最小化项目延迟。每个作业都是项目的一部分。作业可以通过不同的方式执行：每个方法都是一种执行模式，它代表了不同的持续时间，但也可以使用不同的资源。这是一种灵活的 作业商店调度 的一种形式。



硬约束：

- *作业优先级：作业只能在其所有前身作业完成后启动。*
- *资源容量：不要使用超过可用的资源。*
  - *资源是本地（在同一项目中的作业之间共享）或全局（在所有作业间共享）*
  - *资源是可续订（每天可用容量）或不可续订（所有天可用的容量）*

***Medium** 约束：*

- *项目延迟总量：尽量减少每个项目的持续时间**(makespan)**。*

软限制：

- ***Total makespan**：最小化整个多项目调度的持续时间。*

*该问题由 **MISTA 2013 挑战** 定义。*

问题大小

> *Schedule A-1 has 2 projects, 24 jobs, 64 execution modes, 7 resources and 150 resource requirements.*
> *Schedule A-2 has 2 projects, 44 jobs, 124 execution modes, 7 resources and 420 resource requirements.*
> *Schedule A-3 has 2 projects, 64 jobs, 184 execution modes, 7 resources and 630 resource requirements.*
> *Schedule A-4 has 5 projects, 60 jobs, 160 execution modes, 16 resources and 390 resource requirements.*
> *Schedule A-5 has 5 projects, 110 jobs, 310 execution modes, 16 resources and 900 resource requirements.*
> *Schedule A-6 has 5 projects, 160 jobs, 460 execution modes, 16 resources and 1440 resource requirements.*
> *Schedule A-7 has 10 projects, 120 jobs, 320 execution modes, 22 resources and 900 resource requirements.*
> *Schedule A-8 has 10 projects, 220 jobs, 620 execution modes, 22 resources and 1860 resource requirements.*
> *Schedule A-9 has 10 projects, 320 jobs, 920 execution modes, 31 resources and 2880 resource requirements.*
> *Schedule A-10 has 10 projects, 320 jobs, 920 execution modes, 31 resources and 2970 resource requirements.*
> *Schedule B-1 has 10 projects, 120 jobs, 320 execution modes, 31 resources and 900 resource requirements.*
> *Schedule B-2 has 10 projects, 220 jobs, 620 execution modes, 22 resources and 1740 resource requirements.*
> *Schedule B-3 has 10 projects, 320 jobs, 920 execution modes, 31 resources and 3060 resource requirements.*
> *Schedule B-4 has 15 projects, 180 jobs, 480 execution modes, 46 resources and 1530 resource requirements.*
> *Schedule B-5 has 15 projects, 330 jobs, 930 execution modes, 46 resources and 2760 resource requirements.*
> *Schedule B-6 has 15 projects, 480 jobs, 1380 execution modes, 46 resources and 4500 resource requirements.*
> *Schedule B-7 has 20 projects, 240 jobs, 640 execution modes, 61 resources and 1710 resource requirements.*
> *Schedule B-8 has 20 projects, 440 jobs, 1240 execution modes, 42 resources and 3180 resource requirements.*
> *Schedule B-9 has 20 projects, 640 jobs, 1840 execution modes, 61 resources and 5940 resource*

> *requirements.*
> *Schedule B-10 has 20 projects, 460 jobs, 1300 execution modes, 42 resources and 4260 resource requirements.*

3.15. 任务分配

*将每个任务分配到员工队列中的 **spot**。每个任务都有一个持续时间，它受到员工的关联性级别的影响。*

硬约束：

- *技能：每个任务都需要一个或多个技能。员工必须拥有所有这些技能。*

*软级别 **0** 约束：*

- ***Critical** 任务：首先完成关键的任务，比主要任务和次要任务快。*

*软级别 **1** 的限制：*

- *最小化 **makespan**：缩短完成所有任务的时间。*
  - *首先，首先从工作最多的工作员工开始，即员工数量最长，从而创建公平和负载平衡。*

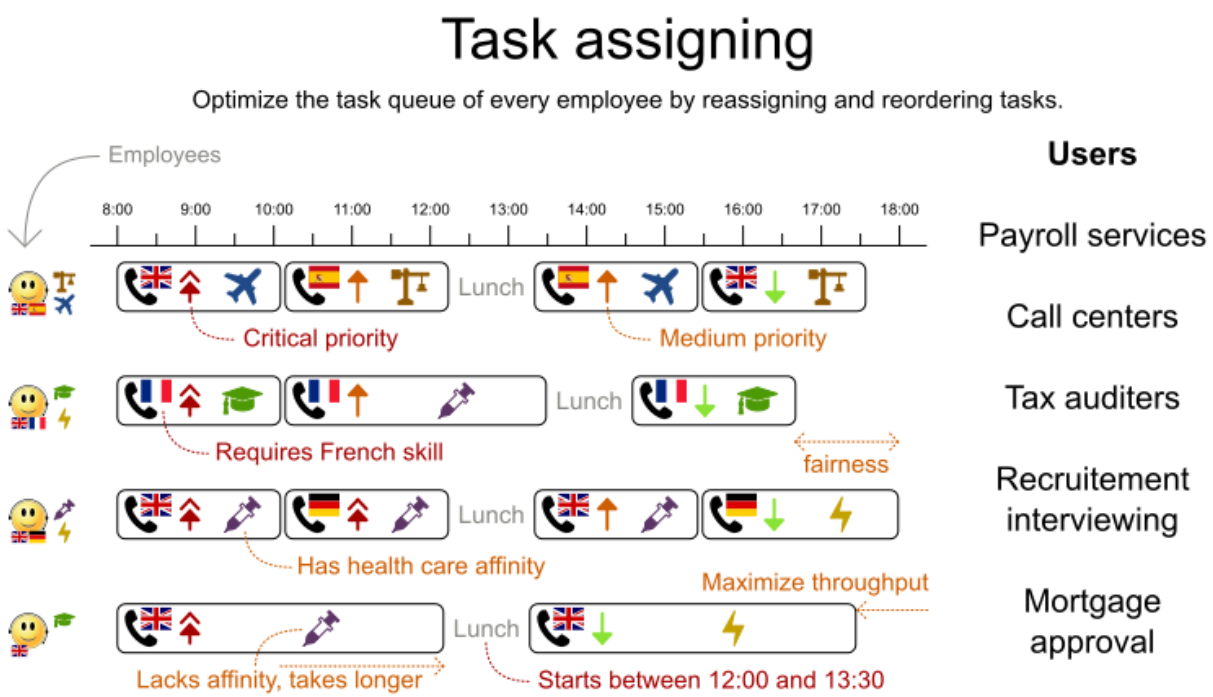
*软级别 **2** 约束：*

- *主要任务：在可能的情况下尽快完成主要任务，比小任务更快。*

*软级别 **3** 的限制：*

- *次要任务：尽快完成次要任务。*

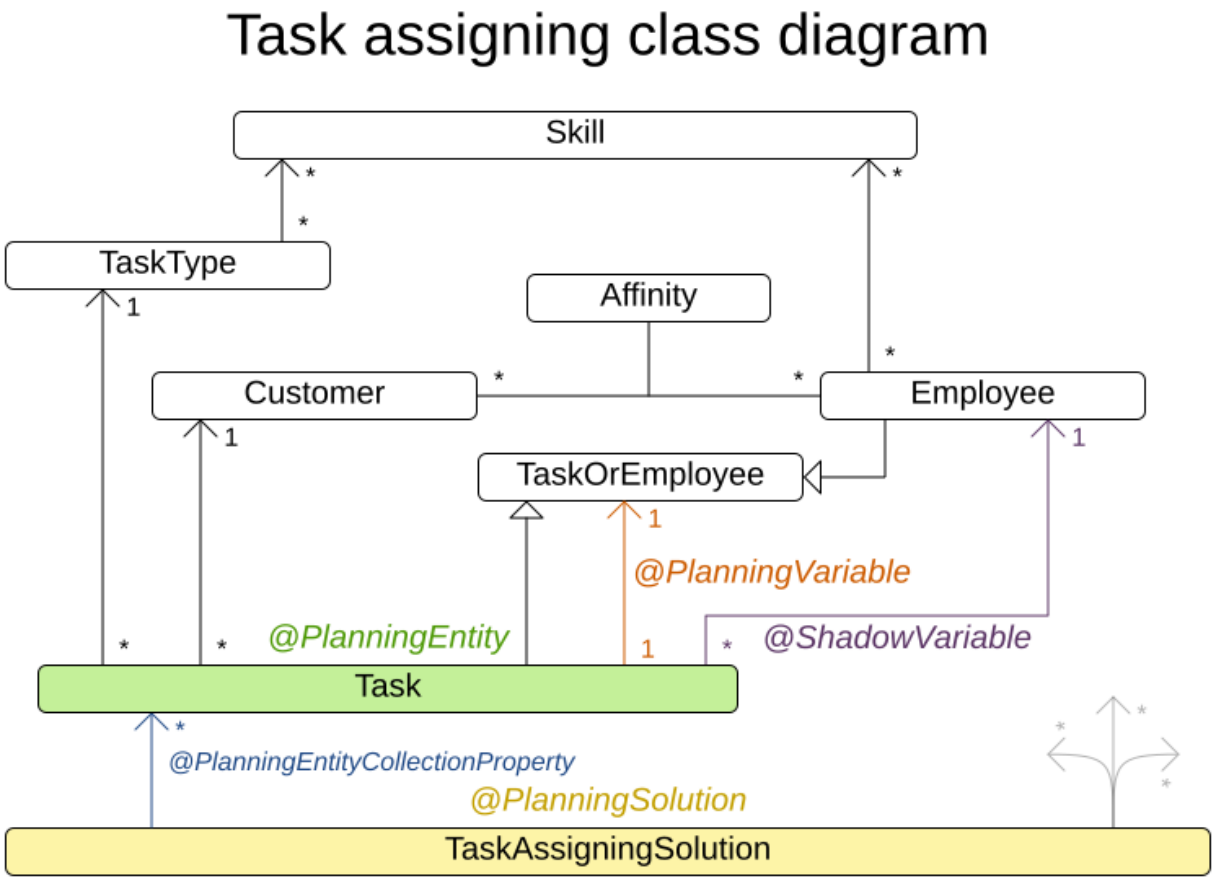
*图 **3.9.** 价值主张*



问题大小

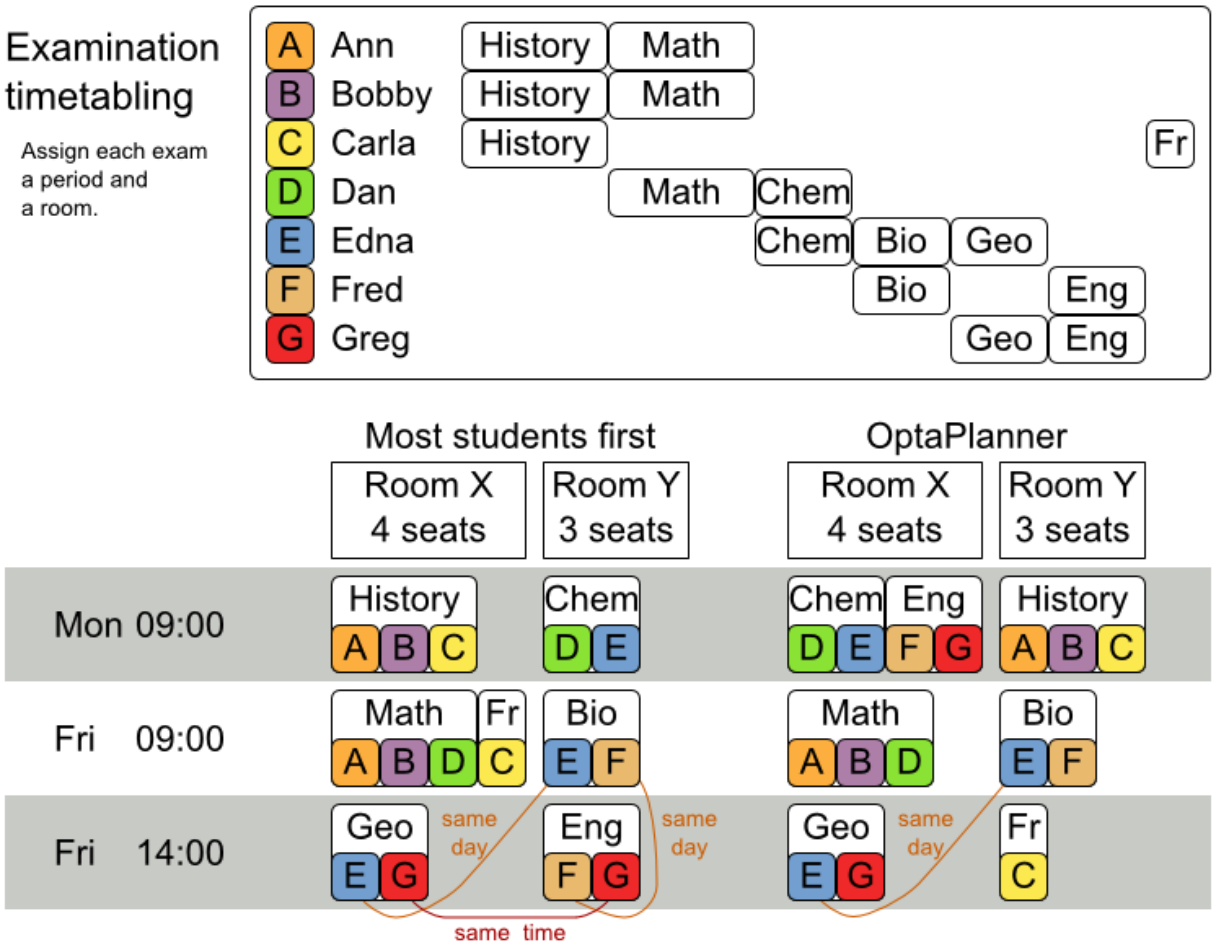
> *24tasks-8employees has 24 tasks, 6 skills, 8 employees, 4 task types and 4 customers with a search space of 10^30.*
> *50tasks-5employees has 50 tasks, 5 skills, 5 employees, 10 task types and 10 customers with a search space of 10^69.*
> *100tasks-5employees has 100 tasks, 5 skills, 5 employees, 20 task types and 15 customers with a search space of 10^164.*
> *500tasks-20employees has 500 tasks, 6 skills, 20 employees, 100 task types and 60 customers with a search space of 10^1168.*

*图 **3.10.** 域模型*



***3.16.** 考试时间表（**ITC 2007** 年跟踪 **1 -** 考试）*

将各考试安排为时段和上门。多个考试可以在同一期间内共享相同的空间。



硬约束：

- *考试冲突：必须在同一时间段内共享学员的两个考试。*
- *房间容量：房座容量必须随时可用。*
- *期间持续时间：其所有考试的持续时间必须有效。*
- *相关的硬限制（为每个数据集指定）：*
  - ***coindence**：两个指定的考试必须使用相同的期限（但可能还会使用其它课程）。*
  - *排除：两个指定的考试不得使用相同的期限。*

- 后：在另一个指定的考试期后，必须在一段时间内进行指定的考试。

- 相关硬约束（为每个数据集指定）：
  - 排行：一个指定的考试不必与任何其他考试共享其空间。

软限制（其各自具有重要优势）：

- 同一人不能连续有两个考试。
- 同一人不能同时拥有两个考试。
- 周期分布：共享学员的两个考试应该是间隔很多句点。
- 混合持续时间：共享房间应该没有不同持续时间的两种考试。
- 前端加载：在计划前面应预先调度大型考试。
- 周期 **penalty**（每个数据集指定）：有些句点（使用时指定句点）。
- **room penalty**（每个数据集指定）：有些房间在使用时有感激。

它使用大量实时测试数据集。

这个问题由 国际时间选项卡 2007 年跟踪 1 定义。**Geoffrey De Smet** 在竞争中，使用非常早的 **OptaPlanner** 版本。然后，从那时起已进行了很多改进。

问题大小

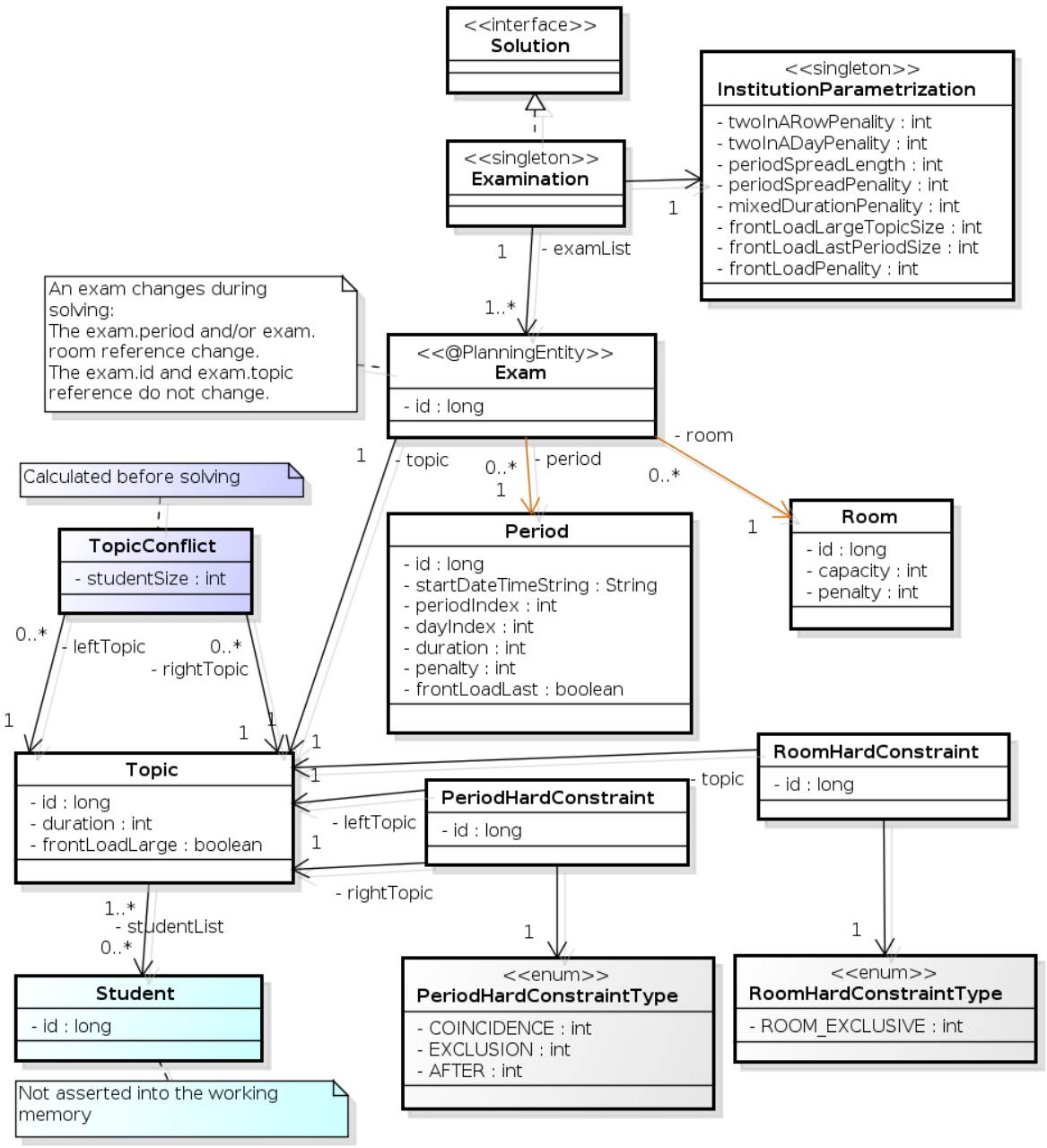
-

> *exam_comp_set1 has 7883 students, 607 exams, 54 periods, 7 rooms, 12 period constraints and 0 room constraints with a search space of 10^1564.*
> *exam_comp_set2 has 12484 students, 870 exams, 40 periods, 49 rooms, 12 period constraints and 2 room constraints with a search space of 10^2864.*
> *exam_comp_set3 has 16365 students, 934 exams, 36 periods, 48 rooms, 168 period constraints and 15 room constraints with a search space of 10^3023.*
> *exam_comp_set4 has 4421 students, 273 exams, 21 periods, 1 rooms, 40 period constraints and 0 room constraints with a search space of 10^360.*
> *exam_comp_set5 has 8719 students, 1018 exams, 42 periods, 3 rooms, 27 period constraints and 0 room constraints with a search space of 10^2138.*
> *exam_comp_set6 has 7909 students, 242 exams, 16 periods, 8 rooms, 22 period constraints and 0 room constraints with a search space of 10^509.*
> *exam_comp_set7 has 13795 students, 1096 exams, 80 periods, 15 rooms, 28 period constraints and 0 room constraints with a search space of 10^3374.*
> *exam_comp_set8 has 7718 students, 598 exams, 80 periods, 8 rooms, 20 period constraints and 1 room constraints with a search space of 10^1678.*

3.16.1. 用于考试时间设置的域模型

下图显示了考试的主要课程：

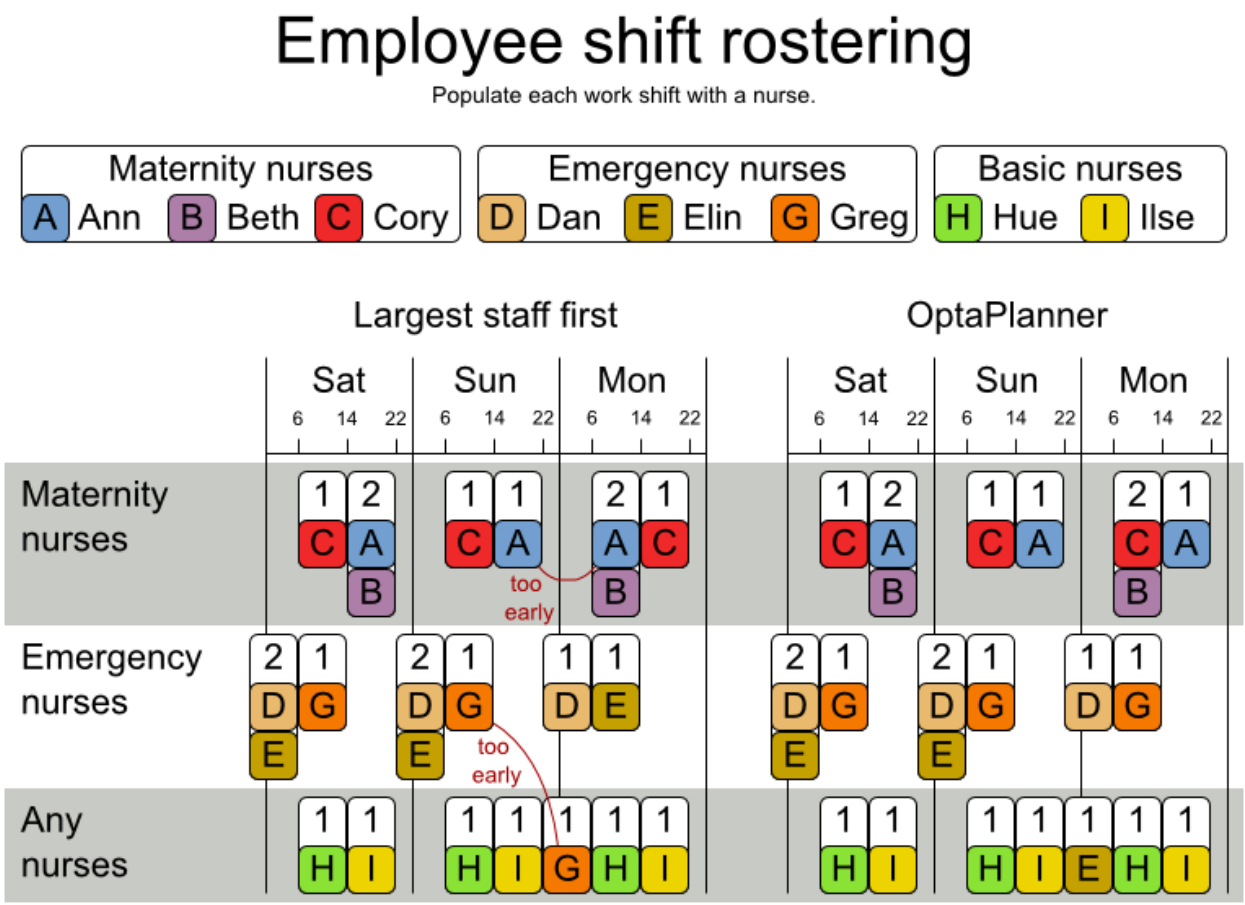
*图 **3.11.** 检查域类图*



*请注意，我们已将考试概念分成 考试 课程和主题 课程。在解决过程中，考试 实例会改变（这是计划实体类），当它们的期间或房间属性改变时。主题、**Period** 和 **Room** 实例在解决过程中不会发生改变（它们有问题的事实，就像其他一些课程一样）。*

3.17. NURSE ROSTER(2010INRC 2010)

对于每个转变，请安排相关变化。



硬约束：

- *无未分配的转变（内置）：所有变化都需要分配给员工。*
- *发生冲突：员工每天只能进行一次转换。*

软限制：

- *合同义务。业务经常违反这些情况，因此他们决定将这些内容定义为软限制，而不是硬约束。*
  - *最小和最大分配：每个员工需要工作超过 **x** 个变化，超过 **y** 个变化（取决于其合约）。*
  - *最少和连续工作天数：每个员工需要在一行中的 **x** 到 **y** 天（取决于其合约）之间工*

作。

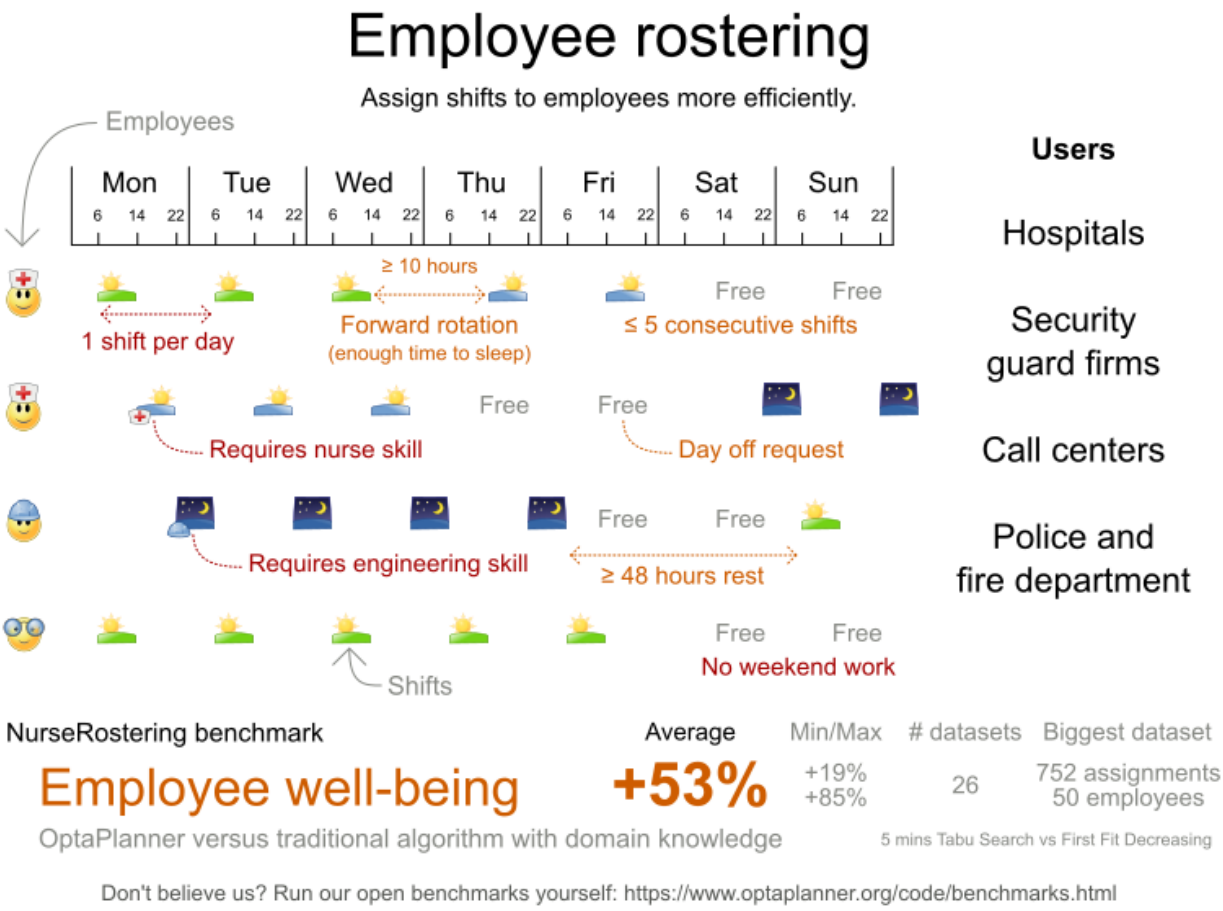
- 最少和最多的空闲天 ：每个员工都需要在一行中的 **x** 到 **y** 天之间释放（取决于其合约）。
- 最小和最大连续工作每周 ：每个员工在一行中的 **x** 和 **y** 周期限之间工作（取决于其合约）。
- 完整的周末 ：每个员工都需要在一个周末或根本上每天工作。
- 周末时相同的转换类型 ：对于同一员工同一周末的每周发生变化都必须是相同的转变类型。
- 不需要的模式 ：一行中不需要的转变类型组合，例如，较早的转变，后再进行较晚的转变。

- 雇员：
  - 申请 当日：员工希望处理某个特定天。
  - 天下请求 ：员工不希望在特定一天上工作。
  - 申请 转变：员工希望分配到特定的转变。
  - 转移请求 ：员工不想分配到特定的转变。

- 备选技能 ：分配给技能的员工应掌握这种转变所需的各项技能。

这个问题由 **2010 年国际 Nurse Rosterionion 定义**。

*图 **3.12.** 价值主张*



问题大小

*有三种 **dataset** 类型：*

- ***Sprint**：必须以秒为单位解决。*
- ***Medium:** 必须以分钟为单位解决。*
- ***long**：必须在几小时内解决。*

> *toy1 has 1 skills, 3 shiftTypes, 2 patterns, 1 contracts, 6 employees, 7 shiftDates, 35 shiftAssignments and 0 requests with a search space of 10^27.*
> *toy2 has 1 skills, 3 shiftTypes, 3 patterns, 2 contracts, 20 employees, 28 shiftDates, 180 shiftAssignments and 140 requests with a search space of 10^234.*
>
> *sprint01 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.*
> *sprint02 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.*

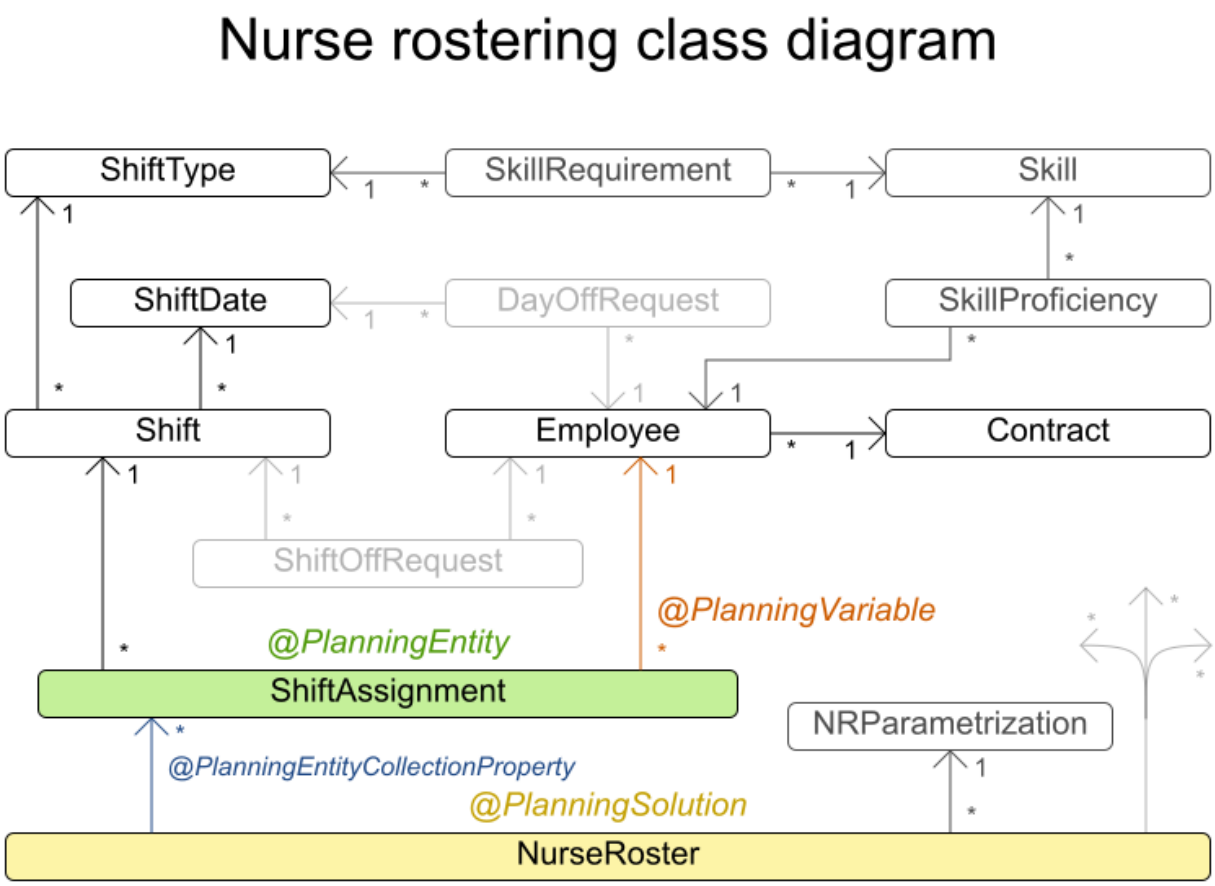
sprint03 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint04 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint05 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint06 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint07 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint08 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint09 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint10 has 1 skills, 4 shiftTypes, 3 patterns, 4 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_hint01 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_hint02 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_hint03 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_late01 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_late02 has 1 skills, 3 shiftTypes, 4 patterns, 3 contracts, 10 employees, 28 shiftDates, 144 shiftAssignments and 139 requests with a search space of 10^144.
sprint_late03 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 160 shiftAssignments and 150 requests with a search space of 10^160.
sprint_late04 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 160 shiftAssignments and 150 requests with a search space of 10^160.
sprint_late05 has 1 skills, 4 shiftTypes, 8 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_late06 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_late07 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.
sprint_late08 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 0 requests with a search space of 10^152.
sprint_late09 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 0 requests with a search space of 10^152.
sprint_late10 has 1 skills, 4 shiftTypes, 0 patterns, 3 contracts, 10 employees, 28 shiftDates, 152 shiftAssignments and 150 requests with a search space of 10^152.

medium01 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of 10^906.
medium02 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of 10^906.
medium03 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of 10^906.
medium04 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of 10^906.
medium05 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 31 employees, 28 shiftDates, 608 shiftAssignments and 403 requests with a search space of 10^906.
medium_hint01 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of 10^632.
medium_hint02 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 428

shiftAssignments and 390 requests with a search space of 10^632.
medium_hint03 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of 10^632.
medium_late01 has 1 skills, 4 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 424 shiftAssignments and 390 requests with a search space of 10^626.
medium_late02 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of 10^632.
medium_late03 has 1 skills, 4 shiftTypes, 0 patterns, 4 contracts, 30 employees, 28 shiftDates, 428 shiftAssignments and 390 requests with a search space of 10^632.
medium_late04 has 1 skills, 4 shiftTypes, 7 patterns, 3 contracts, 30 employees, 28 shiftDates, 416 shiftAssignments and 390 requests with a search space of 10^614.
medium_late05 has 2 skills, 5 shiftTypes, 7 patterns, 4 contracts, 30 employees, 28 shiftDates, 452 shiftAssignments and 390 requests with a search space of 10^667.

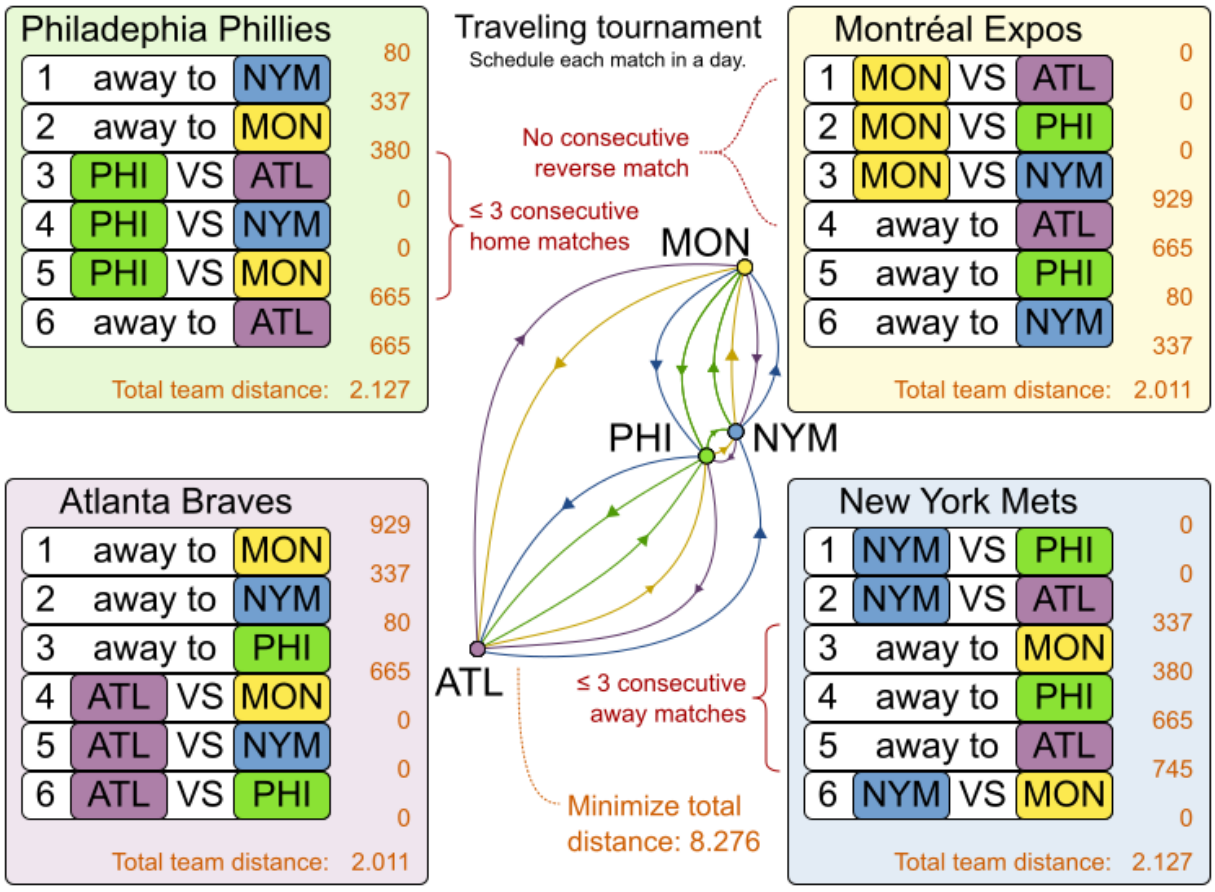
long01 has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of 10^1250.
long02 has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of 10^1250.
long03 has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of 10^1250.
long04 has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of 10^1250.
long05 has 2 skills, 5 shiftTypes, 3 patterns, 3 contracts, 49 employees, 28 shiftDates, 740 shiftAssignments and 735 requests with a search space of 10^1250.
long_hint01 has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and 0 requests with a search space of 10^1257.
long_hint02 has 2 skills, 5 shiftTypes, 7 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and 0 requests with a search space of 10^1257.
long_hint03 has 2 skills, 5 shiftTypes, 7 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and 0 requests with a search space of 10^1257.
long_late01 has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and 0 requests with a search space of 10^1277.
long_late02 has 2 skills, 5 shiftTypes, 9 patterns, 4 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and 0 requests with a search space of 10^1277.
long_late03 has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and 0 requests with a search space of 10^1277.
long_late04 has 2 skills, 5 shiftTypes, 9 patterns, 4 contracts, 50 employees, 28 shiftDates, 752 shiftAssignments and 0 requests with a search space of 10^1277.
long_late05 has 2 skills, 5 shiftTypes, 9 patterns, 3 contracts, 50 employees, 28 shiftDates, 740 shiftAssignments and 0 requests with a search space of 10^1257.

*图 **3.13.** 域模型*



3.18. TRAVELING TOURNAMENT 问题(TTP)

*计划在 **n** 个团队数之间的匹配。*



硬约束：

- *每个团队会针对其他每个团队进行两次操作：一次，一次。*
- *每个团队在每个时间上都有完全匹配。*
- *团队不能连续三个以上的主页或连续三部的匹配。*
- *无重复者：与团队相比，不连续匹配两个。*

软限制：

- *最大程度降低所有团队的距离。*

*在 **Michael Trick** 网站（其中包含世界记录） 上也定义了问题。*

问题大小

```
1-nl04    has  6 days,  4 teams and   12 matches with a search space of    10^5.
1-nl06    has 10 days,  6 teams and   30 matches with a search space of   10^19.
1-nl08    has 14 days,  8 teams and   56 matches with a search space of   10^43.
1-nl10    has 18 days, 10 teams and   90 matches with a search space of   10^79.
1-nl12    has 22 days, 12 teams and  132 matches with a search space of  10^126.
1-nl14    has 26 days, 14 teams and  182 matches with a search space of  10^186.
1-nl16    has 30 days, 16 teams and  240 matches with a search space of  10^259.
2-bra24   has 46 days, 24 teams and  552 matches with a search space of  10^692.
3-nfl16   has 30 days, 16 teams and  240 matches with a search space of  10^259.
3-nfl18   has 34 days, 18 teams and  306 matches with a search space of  10^346.
3-nfl20   has 38 days, 20 teams and  380 matches with a search space of  10^447.
3-nfl22   has 42 days, 22 teams and  462 matches with a search space of  10^562.
3-nfl24   has 46 days, 24 teams and  552 matches with a search space of  10^692.
3-nfl26   has 50 days, 26 teams and  650 matches with a search space of  10^838.
3-nfl28   has 54 days, 28 teams and  756 matches with a search space of  10^999.
3-nfl30   has 58 days, 30 teams and  870 matches with a search space of 10^1175.
3-nfl32   has 62 days, 32 teams and  992 matches with a search space of 10^1367.
4-super04 has  6 days,  4 teams and   12 matches with a search space of    10^5.
4-super06 has 10 days,  6 teams and   30 matches with a search space of   10^19.
4-super08 has 14 days,  8 teams and   56 matches with a search space of   10^43.
4-super10 has 18 days, 10 teams and   90 matches with a search space of   10^79.
4-super12 has 22 days, 12 teams and  132 matches with a search space of  10^126.
4-super14 has 26 days, 14 teams and  182 matches with a search space of  10^186.
5-galaxy04 has  6 days,  4 teams and   12 matches with a search space of    10^5.
5-galaxy06 has 10 days,  6 teams and   30 matches with a search space of   10^19.
5-galaxy08 has 14 days,  8 teams and   56 matches with a search space of   10^43.
5-galaxy10 has 18 days, 10 teams and   90 matches with a search space of   10^79.
5-galaxy12 has 22 days, 12 teams and  132 matches with a search space of  10^126.
5-galaxy14 has 26 days, 14 teams and  182 matches with a search space of  10^186.
5-galaxy16 has 30 days, 16 teams and  240 matches with a search space of  10^259.
5-galaxy18 has 34 days, 18 teams and  306 matches with a search space of  10^346.
5-galaxy20 has 38 days, 20 teams and  380 matches with a search space of  10^447.
5-galaxy22 has 42 days, 22 teams and  462 matches with a search space of  10^562.
5-galaxy24 has 46 days, 24 teams and  552 matches with a search space of  10^692.
5-galaxy26 has 50 days, 26 teams and  650 matches with a search space of  10^838.
5-galaxy28 has 54 days, 28 teams and  756 matches with a search space of  10^999.
5-galaxy30 has 58 days, 30 teams and  870 matches with a search space of 10^1175.
5-galaxy32 has 62 days, 32 teams and  992 matches with a search space of 10^1367.
5-galaxy34 has 66 days, 34 teams and 1122 matches with a search space of 10^1576.
5-galaxy36 has 70 days, 36 teams and 1260 matches with a search space of 10^1801.
5-galaxy38 has 74 days, 38 teams and 1406 matches with a search space of 10^2042.
5-galaxy40 has 78 days, 40 teams and 1560 matches with a search space of 10^2301.
```

3.19. 更低的时间调度

以时间和方式调度所有任务，以最大程度降低电源成本。电源价格因时间而异。这是 作业权利调度的一种形式。

硬约束：

- 开始时间限制：每个任务必须在最早的开始和最新开始限制之间启动。
- 最大容量：不能超过每台机器的每个资源的最大容量。
- 启动和关闭：在分配任务的期间，每台机器都必须处于激活状态。在任务之间，可以闲置它以避免启动和关闭成本。

Medium 约束：

- 电源成本：降低整个计划的总功耗。
  - 机器电源成本：每个活跃或闲置机器都会消耗电源成本，这导致了电源成本（取决于该期间的电源价格）。
  - 任务电源成本：每个任务都消耗了电源成本，这导致了电源成本（取决于一段时间内的电源价格）。
  - 机器启动和关闭成本：计算机启动或关闭的所有时间都会产生额外的成本。

软限制（在原始问题定义中添加）：

- 早开始：请参阅立即启动任务，而不是稍后启动。

这个问题由 **ICON** 质询 定义。

问题大小

sample01 has 3 resources, 2 machines, 288 periods and 25 tasks with a search space of 10^53.
sample02 has 3 resources, 2 machines, 288 periods and 50 tasks with a search space of 10^114.
sample03 has 3 resources, 2 machines, 288 periods and 100 tasks with a search space of 10^226.
sample04 has 3 resources, 5 machines, 288 periods and 100 tasks with a search space of 10^266.
sample05 has 3 resources, 2 machines, 288 periods and 250 tasks with a search space of 10^584.
sample06 has 3 resources, 5 machines, 288 periods and 250 tasks with a search space of 10^673.
sample07 has 3 resources, 2 machines, 288 periods and 1000 tasks with a search space of 10^2388.
sample08 has 3 resources, 5 machines, 288 periods and 1000 tasks with a search space of 10^2748.
sample09 has 4 resources, 20 machines, 288 periods and 2000 tasks with a search space of 10^6668.
instance00 has 1 resources, 10 machines, 288 periods and 200 tasks with a search space of 10^595.
instance01 has 1 resources, 10 machines, 288 periods and 200 tasks with a search space of 10^599.
instance02 has 1 resources, 10 machines, 288 periods and 200 tasks with a search space of 10^599.
instance03 has 1 resources, 10 machines, 288 periods and 200 tasks with a search space of 10^591.
instance04 has 1 resources, 10 machines, 288 periods and 200 tasks with a search space of 10^590.
instance05 has 2 resources, 25 machines, 288 periods and 200 tasks with a search space of 10^667.
instance06 has 2 resources, 25 machines, 288 periods and 200 tasks with a search space of 10^660.
instance07 has 2 resources, 25 machines, 288 periods and 200 tasks with a search space of 10^662.
instance08 has 2 resources, 25 machines, 288 periods and 200 tasks with a search space of 10^651.
instance09 has 2 resources, 25 machines, 288 periods and 200 tasks with a search space of 10^659.
instance10 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1657.
instance11 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1644.
instance12 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1637.
instance13 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1659.
instance14 has 2 resources, 20 machines, 288 periods and 500 tasks with a search space of 10^1643.
instance15 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10^1782.
instance16 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of

10^1778.
instance17 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10^1764.
instance18 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10^1769.
instance19 has 3 resources, 40 machines, 288 periods and 500 tasks with a search space of 10^1778.
instance20 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10^3689.
instance21 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10^3678.
instance22 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10^3706.
instance23 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10^3676.
instance24 has 3 resources, 50 machines, 288 periods and 1000 tasks with a search space of 10^3681.
instance25 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10^3774.
instance26 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10^3737.
instance27 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10^3744.
instance28 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10^3731.
instance29 has 3 resources, 60 machines, 288 periods and 1000 tasks with a search space of 10^3746.
instance30 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10^7718.
instance31 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10^7740.
instance32 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10^7686.
instance33 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10^7672.
instance34 has 4 resources, 70 machines, 288 periods and 2000 tasks with a search space of 10^7695.
instance35 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10^7807.
instance36 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10^7814.
instance37 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10^7764.
instance38 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10^7736.
instance39 has 4 resources, 80 machines, 288 periods and 2000 tasks with a search space of 10^7783.
instance40 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of 10^15976.
instance41 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of 10^15935.
instance42 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of 10^15887.
instance43 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of 10^15896.
instance44 has 4 resources, 90 machines, 288 periods and 4000 tasks with a search space of

> *10^15885.*
> *instance45 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of 10^20173.*
> *instance46 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of 10^20132.*
> *instance47 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of 10^20126.*
> *instance48 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of 10^20110.*
> *instance49 has 4 resources, 100 machines, 288 periods and 5000 tasks with a search space of 10^20078.*

3.20. 投资资产类分配（PORTFOLIO 优化）

决定在每个资产类投资的相对数量。

硬约束：

- *风险最大：标准开发总量不得高于标准开发最大值。*
  - *标准开发总体计算通过应用 Markowitz 组合 Theory 将资产类关联起来考虑起来。*
- *地区上限：每个地区具有最大数量。*
- *最大扇区数：每个扇区具有最大数量。*

软限制：

- *最大化预期的回报。*

问题大小

> *de_smet_1 has 1 regions, 3 sectors and 11 asset classes with a search space of 10^4.*
> *irrinki_1 has 2 regions, 3 sectors and 6 asset classes with a search space of 10^3.*

较大的数据集尚未创建或测试，但不应构成问题。好的数据来源是 这一资产关联网站。

3.21. 会议调度

*将每个会议分配给一个小时和房间。**Timeslots** 可能会重叠。在 ***.xlsx** 文件中读取和写入，该文件可使用 **libreoffice** 或 **Excel** 编辑。*

硬约束：

- ***talk type of timeslot**：讨论的类型必须与 **timeslot** 的对话类型匹配。*
- *房间不可用：讨论的房间必须在对话期间可用。*
- *间冲突：两个对话在重叠时不能使用相同的空间。*
- *发言人不可用时间：所有对话的发言人必须在对话期间可用。*
- *发言人冲突：两个对话在重叠时无法共享发言人。*
- *通用用途和房间标签：*
  - *发言人所需时间标签：如果发言人具有所需的 **timelot** 标签，则所有其或她的讨论都必须使用该标签分配给一个小时。*
  - *发言人禁止标记：如果发言人具有禁止的时间板板，那么他或她的所有话都不能被分配给具有该标签的时期。*

- ***talk required timelot tag**：如果一个 **talk** 具有所需的 **timelot** 标签，则必须将其分配给带有该标签的 **timeslot**。*
- *禁止时间标签：如果对话具有禁止的计时标签，则无法使用该标签将其分配给时间。*
- ***speaker required room tag**：如果发言人具有必需的 **room** 标签，则必须将其所有或她的讨论分配到具有该标签的房间。*
- *演讲者禁止房间标签：如果发言人有禁止的房间标签，那么他或她的所有客户都不能分配给具有该标签的房间。*
- ***talk required room tag**： 如果 **talk** 具有必需的 **room** 标签，则必须将其分配给具有该标签的房间。*
- *谈话空间标签：如果对话有一个禁止的房间标签，则无法将其分配给具有该标签的房间。*

- *互斥标签：共享这样的标签不能调度在重叠的时间里。*
- *对话：所有预备讨论之后必须调度 **talk talk**。*

软限制：

- ***me track 冲突**：减少在重叠时间内共享主题标签的讨论数量。*
- *扇区冲突：减少在重叠期内共享相同的扇区标签的讨论数量。*
- *内容受众级别流违反：对于每内容标签，在高级讨论前，安排简介。*
- *受众级多样性：对于每个时间，请尽量减少与不同受众级别的讨论数量。*

- *语言多样性：对于每一时间来说，最大程度地提高与不同语言的沟通数量。*

- *通用用途和房间标签：*

  - *发言人首选的时间标签：如果发言人具有首选 **timelot** 标签，则本标签中的所有讨论都应分配给一个小时。*

  - *发言人不需要的时间**lot** 标签： 如果发言人具有不所需时间标签，那么他或她的任何客户都不能被分配给具有该标签的时间段。*

  - ***talk preferred timeslot tag**：如果一个 **talk** 具有首选 **timeslot** 标签，则应使用该标签将其分配给一个 **timeslot**。*

  - ***talk undesired timeslot tag:** 如果 **talk** 带有不良的 **timelot** 标签，则不应将其分配给具有该标签的运行时间。*

  - *演讲者首选的房间标签：如果发言人拥有首选房间标签，则本人或她的所有讨论都应分配给具有该标签的房间标签。*

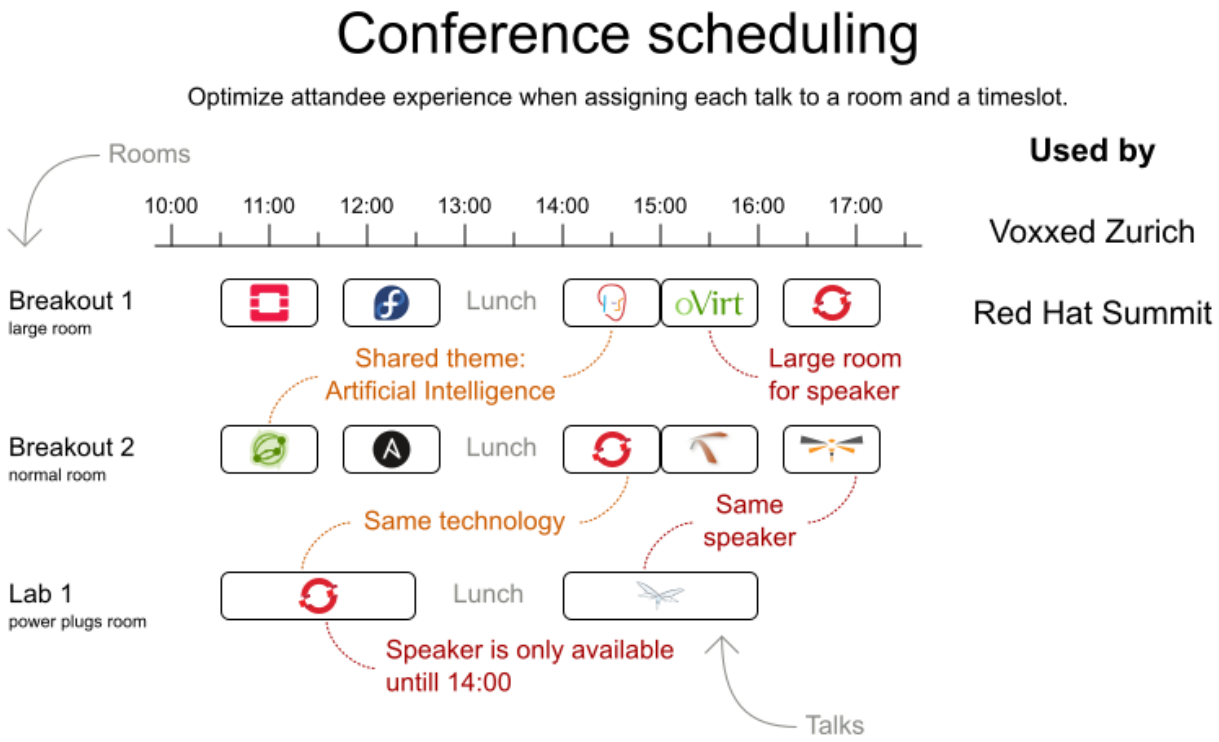
  - ***speaker undesired room** 标签： 如果发言人具有不需要的房间标签，则不应将任何其讨论分配给具有该标签的房间。*

  - ***talk preferred room tag**：如果 **talk** 具有首选空间标签，则应将其分配给具有该标签的房间。*

  - ***talk undesired room tag:** 如果 **talk anirdesired room** 标签，则不应将其分配给具有该标签的房间。*

- *第二天讨论：所有共享主题标签或内容标签都应以最少的天数调度（在同一天内）。*

*图 **3.14.** 价值主张*



问题大小

> *18talks-6timeslots-5rooms has 18 talks, 6 timeslots and 5 rooms with a search space of 10^26.*
> *36talks-12timeslots-5rooms has 36 talks, 12 timeslots and 5 rooms with a search space of 10^64.*
> *72talks-12timeslots-10rooms has 72 talks, 12 timeslots and 10 rooms with a search space of 10^149.*
> *108talks-18timeslots-10rooms has 108 talks, 18 timeslots and 10 rooms with a search space of 10^243.*
> *216talks-18timeslots-20rooms has 216 talks, 18 timeslots and 20 rooms with a search space of 10^552.*

3.22. STTOUR

从展到演示到展示的车车，但计划仅显示可用日。

硬约束：

- *安排每个必需显示。*
- *计划尽可能多的显示。*

***Medium** 约束：*

- *最大化收入机会。*
- *最小化驱动时间。*
- *快于稍后访问。*

软限制：

- *避免长时间推动时间。*

问题大小

> *47shows has 47 shows with a search space of 10^59.*

3.23. FLIGHT CREW 调度

为试点和航班出员指派航班。

硬约束：

- 所需技能：每个班级分配都具有必要的技能。例如，**flight AB0001** 需要 **2** 个试点和 **3** 班级参加。

- 动态冲突：每个员工只能同时参加一个动态

- 在两个航班之间传输：在两个机之间，员工必须能够从 **arrival airport** 转让到 **departure airport**。例如，**An** 到达 **Brussels at 10:00**，在位于 **15:00** 的阿姆斯特区。

- 员工不可用：该员工必须在航班的某一天可用。例如，**An** 是 **1-Feb** 上的 **PTO**。

软限制：

- 首次分配来自家

- 最后分配到达家

- 每个员工的负载均衡持续时间

问题大小

> 175flights-7days-Europe has 2 skills, 50 airports, 150 employees, 175 flights and 875 flight assignments with a search space of 10^1904.
> 700flights-28days-Europe has 2 skills, 50 airports, 150 employees, 700 flights and 3500 flight assignments with a search space of 10^7616.
> 875flights-7days-Europe has 2 skills, 50 airports, 750 employees, 875 flights and 4375 flight assignments with a search space of 10^12578.
> 175flights-7days-US has 2 skills, 48 airports, 150 employees, 175 flights and 875 flight assignments with a search space of 10^1904.

第 4 章 下载红帽构建的 OPTAPLANNER 示例

您可以下载 **Red Hat build of OptaPlanner** 示例，作为红帽客户门户网站中提供的 **{PRODUCTPAM}** 附加组件软件包的一部分。

流程

1. 进入红帽客户门户网站中的 **Software Downloads** 页面（需要登录），然后从下拉列表中选择产品和版本：
   - 产品： 流程自动化管理器
   - 版本： **7.13.4**
2. 下载 **Red Hat Process Automation Manager 7.13 Add Ons**。
3. 提取 **rhpam-7.13.4-add-ons.zip** 文件。提取 的附加组件文件夹包含 **rhpam- 7.13.4-planner-engine.zip** 文件。
4. 提取 **rhpam-7.13.4-planner-engine.zip** 文件。

结果

提取的 **rhpam-7.13.4-planner-engine** 目录包含以下子目录下的示例源代码：

- 示例/源/**src/main/java/org/optaplanner/examples**
- 示例/源/**src/main/resources/org/optaplanner/examples**

4.1. 运行 OPTAPLANNER 示例

红帽构建的 **OptaPlanner** 包含多个示例，演示各种计划用例。下载并使用示例来探索不同类型的规划解决方案。

先决条件

- *您已下载并提取示例，如 第 4 章 下载红帽构建的 OptaPlanner 示例 所述。*

流程

1. *要运行示例，请在 **rhpam-7.13.4-planner-engine/examples** 目录中输入以下命令之一：*

   Linux** 或 **Mac:

   ```
   $ ./runExamples.sh
   ```

   ***Windows**：*

   ```
   $ runExamples.bat
   ```

   ***OptaPlanner Examples** 窗口将打开。*

2. *选择一个示例来运行该示例。*

注意

***Red Hat build of OptaPlanner** 没有 **GUI** 依赖项。它还可在服务器或移动 **JVM** 上运行，就像桌面一样。*

4.2. 在 IDE 中运行 OPTAPLANNER 示例构建（INTELLIJ、ECLIPSE 或 NETBEANS）

*如果您使用集成开发环境(IDE)，如 **IntelliJ**、**Eclipse** 或 **Netbeans**，您可以在开发环境中运行下载的 **OptaPlanner** 示例。*

先决条件

- *您已下载并提取了 **OptaPlanner** 示例，如 第 4 章 下载红帽构建的 OptaPlanner 示例 所述。*

流程

1. 打开 **OptaPlanner** 示例作为新项目：

   a. 对于 **IntelliJ** 或 **Netbeans**，开放 示例/源/**pom.xml** 作为新项目。**Maven** 集成指南，指导您完成剩余的安装。跳过这个过程中的其余步骤。

   b. 对于 **Eclipse**，为 **/examples/binaries** 目录打开一个新项目，它位于 **rhpam-7.13.4-planner-engine** 目录下。

2. 将所有位于 二进制文件 目录中的所有 **JAR** 文件添加到 **classpath** 中，但 **examples/binaries/optaplanner-examples-7.67.0.Final-redhat-00024.jar** 文件除外。

3. 添加 **Java** 源目录 **src/main/java** 和 **Java** 资源目录 **src/main/resources**，它位于 **rhpam-7.13.4-planner-engine/examples/sources/** 目录下。

4. 创建运行配置：

   - **main class: org.optaplanner.examples.app.OptaPlannerExamplesApp**
   - VM 参数（可选）：**-Xmx512M -server -Dorg.optaplanner.examples.dataDir=examples/sources/data**
   - 工作目录：**example/sources**

5. 运行 **run** 配置。

第 5 章 BUSINESS CENTRAL 中的 OPTAPLANNER 入门：员工名单示例

您可以在 **Business Central** 中构建和部署员工 模板示例项目。该项目演示了如何创建解决传统规划问题所需的每个业务中心资产，并使用红帽构建的 **OptaPlanner** 来查找可能的解决方案。

您可以在 **Business Central** 中部署预配置的 员工 **-rostering** 项目。或者，您可以使用 **Business Central** 自行创建项目。

注意

Business Central 中的 **employees -rostering** 示例项目不包括数据集。您必须使用 **REST API** 调用以 **XML** 格式提供数据集。

5.1. 在 BUSINESS CENTRAL 中部署 EMPLOYEES ROSTERING 示例项目

Business Central 包括很多可用于熟悉产品和其功能的示例项目。员工的 **rostering** 示例项目经过设计和创建，以展示红帽构建 **OptaPlanner** 的变速用例。使用以下步骤部署和运行 **Business Central** 中的员工降级示例。

先决条件

- **Red Hat Process Automation Manager** 下载并安装。有关安装选项，请参阅 *规划 Red Hat Process Automation Manager 安装*。
- 您已启动了 **Red Hat Process Automation Manager**，如安装文档，且您使用具有 **admin** 权限的用户身份登录 **Business Central**。

流程

1. 在 **Business Central** 中，点击 **Menu** → **Design** → **Projects**。
2. 在预配置的 **MySpace** 空间中，点 **Try Samples**。
3. 从示例项目列表中选择 **employees -rostering**，然后单击右上角的 **Ok** 来导入项目。

4. 资产列表变得复杂后，请单击 **Build & Deploy** 来部署员工的 **rostering** 示例。

本文档的其余部分介绍了各个项目资产及其配置。

5.2. 重新排序员工降级示例项目

员工的 **rostering** 示例项目是 **Business Central** 中提供的一个预配置的项目。您可以在 第 **5.1** 节 “在 **Business Central** 中部署 **employees rostering** 示例项目” 中了解如何部署此项目。

您可以创建员工的名册示例"全新"。您可以使用本例中的工作流在 **Business Central** 中创建您自己的类似项目。

5.2.1. 设置员工的 rostering 项目

要在 **Business Central** 中开始开发解决方法，您必须设置该项目。

先决条件

- **Red Hat Process Automation Manager** 下载并安装。
- 您已部署了 **Business Central**，并使用具有管理员角色的用户登录。

流程

1. 在 **Business Central** 中点 **Menu** → **Design** → **Projects** → **Add Project** 创建一个新项目。
2. 在 **Add Project** 窗口中，填写以下字段：
   - **Name : staff-rostering**
   - **Description**（可选）：使用 **OptaPlanner** 的 **Employee rostering** 问题优化。可根据自身技能分配员工进行转变。

可选：点 **Configure Advanced Options** 填充 组 **ID**、**Artifact ID** 和版本信息。

- 组 **ID** ：员工
- 工件 **ID** ：员工
- 版本 **1**：**1NAPSHOT**

3. 单击 **Add**，将项目添加到 **Business Central** 项目存储库。

5.2.2. 问题事实和规划实体

员工时间表问题中的每个域类都归类为以下一种：

- 不相关的类：未由任何分数限制使用。从规划角度来说，这个数据已过时。
- 问题事实 类：在分数约束下使用，但在规划期间不会改变（只要问题仍保持相同），例如 **Shift** 和 **Employee**。问题事实类的所有属性都是问题属性。
- 计划实体类：在规划 过程中对分数约束和更改使用，例如 **ShiftAssignment**。规划期间更改的属性是 规划变量。其他属性是问题属性。

询问以下问题：

- 计划过程中发生了哪些类变化？
- 哪个类具有我想要 更改 的变量？

该类是规划实体。

计划实体类需要使用 **@PlanningEntity** 注释注释，或使用红帽在域设计器中的

OptaPlanner dock 进行定义。

每个规划实体类都有一个或多个 规划变量，并且还必须有一个或多个定义属性。

大多数用例只有一个规划实体类，每个计划实体类只有一个规划变量。

5.2.3. 为员工降级项目创建数据模型

使用这个部分，创建在 **Business Central** 中运行员工指定示例项目所需的数据对象。

先决条件

- 您已完成 第 **5.2.1** 节 “设置员工的 **rostering** 项目” 中描述的项目设置。

流程

1. 使用新项目时，点击项目视角中的 **Data Object**，或者点击 **Add Asset** → **Data Object** 以创建新的数据对象。
2. 将第一个数据对象 时间 命名为，然后选择 **employees rostering.employeerostering** 作为软件包。

   点 确定。
3. 在 **Data Objects** 视角中，点 **+add** 字段将字段 添加到 **Timeslot data** 对象。
4. 在 **id** 字段中，键入 **endTime**。
5. 单击 **Type** 旁边的下拉菜单，然后选择 **LocalDateTime**。
6. 点击 **Create** 并继续 添加另一个字段。

7. *添加另一个字段，其中包含* ***id startTime*** *和* ***Type LocalDateTime****。*

8. *点* ***Create****。*

9. *单击右上角的* ***Save****，以保存* ***Timeslot*** *数据对象。*

10. *单击右上角的* ***x*** *以关闭* ***Data Objects*** *透视图，再返回到* ***Assets*** *菜单。*

11. *使用前面的步骤创建以下数据对象及其属性：*

表 ***5.1.*** *技巧*

id	类型
name	**字符串**

表 ***5.2.*** *员工*

id	类型
name	**字符串**
技能	**employeerostering.employeerostering.Skill[List]**

表 ***5.3.*** *改变*

id	类型
requiredSkill	**employeerostering.employeerostering.Skill**
timeslot	**employeerostering.employeerostering.Timeslot**

表 ***5.4. DayOffRequest***

id	类型
date	**LocalDate**

id	类型
员工	**employeerostering.employeerostering.Employee**

*表 **5.5. ShiftAssignment***

id	类型
员工	**employeerostering.employeerostering.Employee**
改变	**employeerostering.employeerostering.Shift**

有关创建数据对象的更多示例，请参阅开始使用决策服务。

*5.2.3.1. 创建员工的 **roster** 计划实体*

*为了解决员工问答规划问题，您必须创建一个计划实体和解决方法。计划实体在域设计人员中使用 **OptaPlanner dock** 中可用的属性定义。*

*使用以下步骤将 **ShiftAssignment** 数据对象定义为员工名册的规划实体。*

先决条件

- *您已通过完成 **第 5.2.3 节 “为员工降级项目创建数据模型”** 中的步骤创建运行员工代理示例所需的相关数据对象和规划实体。*

流程

1. *在项目 **Assets** 菜单中，打开 **ShiftAssignment data** 对象。*
2. *在 **Data Objects** 视角中，点右侧的*

   *来打开 **OptaPlanner dock**。*

3. 选择 计划实体。

4. 从 **ShiftAssignment data** 对象下的字段列表中选择 **employees**。

5. 在 **OptaPlanner dock** 中，选择 规划变量。

   在 **Value Range Id** 输入字段中，键入 **employeesRange**。这会在计划实体中添加 **@ValueRangeProvider** 注释，您可以通过单击设计者中的 **Source** 选项卡来查看。

   planning 变量的值范围通过 **@ValueRangeProvider** 注释定义。**@ValueRangeProvider** 注释始终具有属性 **id**，它被 **@PlanningVariable** 属性值**RangeProviderRefs** 引用。

6. 关闭 **dock** 并单击 **Save** 以保存数据对象。

5.2.3.2. 创建员工的 roster 计划解决方案

员工的 **roster** 问题依赖于定义的规划解决方案。计划解决方案由红帽构建 **OptaPlanner dock** 中的属性在域设计器中定义。

先决条件

- 您已通过完成 第 **5.2.3** 节 “为员工降级项目创建数据模型” 和 第 **5.2.3.1** 节 “创建员工的 **roster** 计划实体” 中的步骤创建运行员工指定示例所需的相关数据对象和规划实体。

流程

1. 使用标识符 **EmployeeRoster** 创建一个新的数据对象。

2. 创建以下字段：

表 5.6. EmployeeRoster

id	类型
dayOffRequestList	**employeerostering.employeerostering.DayOffRequest[List]**

id	类型
shiftAssignmentList	**employeerostering.employeerostering.ShiftAssignment[List]**
shiftList	**employeerostering.employeerostering.Shift[List]**
skillList	**employeerostering.employeerostering.Skill[List]**
timeslotList	**employeerostering.employeerostering.Timeslot[List]**

3. *在 **Data Objects** 视角中，点右侧的*

*来打开 **OptaPlanner dock**。*

4. *选择 规划解决方案.*

5. *将默认的 **hard soft** 分数 保留为 **Solution Score Type**。这会在 **EmployeeRoster** 数据对象中自动生成 分数，该分数作为类型。*

6. *使用以下属性添加新字段：*

id	类型
employeeList	**employeerostering.employeerostering.Employee[List]**

7. *选择 **employees List** 字段后，打开 **OptaPlanner dock**，再选择 **Planning Value Range Provider** 框。*

   *在 **id** 字段中，键入 **staff Range**。关闭 **dock**。*

8. *点击右上角的 **Save** 保存资产。*

5.2.4. 员工的降级限制

员工名单是一个规划问题。所有规划问题都包括必须满足约束才能找到最佳解决方案。

Business Central 中的员工指定示例项目包括以下硬和软限制：

硬约束

- 员工仅针对每天进行一次转变。
- 所有需要特定员工技能的改变都会为拥有该特定技能的员工分配。

软限制

- 所有员工都被分配了一项转变。
- 如果雇员请求一天，其变化会被重新分配给其他员工。

硬和软限制在 Business Central 中通过自由-form DRL 设计器定义，也可以使用指导规则。

5.2.4.1. DRL（Drools 规则语言）规则

DRL(Drools Rule)规则是您在 .drl 文本文件中直接定义的业务规则。这些 DRL 文件是 Business Central 中所有其他规则资产渲染的源。您可以在 Business Central 界面中创建和管理 DRL 文件，或使用 Red Hat CodeReady Studio 或其他集成开发环境(IDE)在外部创建它们。DRL 文件可以包含一个或多个规则，它们至少定义规则条件（在时）和操作（然后再）。Business Central 中的 DRL 设计器为 Java、DRL 和 XML 提供语法高亮显示。

DRL 文件由以下组件组成：

DRL 文件中的组件

> package

```
import

function  // Optional

query  // Optional

declare   // Optional

global   // Optional

rule "rule name"
    // Attributes
    when
        // Conditions
    then
        // Actions
end

rule "rule2 name"

...
```

以下示例 **DRL** 规则决定了 **loan** 应用程序决策服务中的年龄限制：

loan Application age 限制的规则示例

```
rule "Underage"
  salience 15
  agenda-group "applicationGroup"
  when
    $application : LoanApplication()
    Applicant( age < 21 )
  then
    $application.setApproved( false );
    $application.setExplanation( "Underage" );
end
```

DRL 文件包含单个或多个规则、查询和函数，并可以定义由规则和查询分配和使用的属性等资源声明。**DRL** 软件包必须在 **DRL** 文件的顶部列出，规则通常最后列出。所有其他 **DRL** 组件可遵循任何顺序。

每个规则必须在规则软件包中具有唯一的名称。如果您在软件包中的任何 **DRL** 文件中使用相同的规则名称多次，则规则无法编译。始终使用双引号括起规则名称（规则**"rule name"**），以防止可能出现的编译错误，特别是在规则名称中使用空格。

与 **DRL** 规则相关的所有数据对象都必须位于与 **Business Central** 中的 **DRL** 文件相同的项目软件包中。默认导入同一软件包中的资产。其他软件包中的现有资产可以通过 **DRL** 规则导入。

5.2.4.2. 使用 DRL designer 定义员工指定限制

您可以使用 **Business Central** 中的自由**-form DRL** 设计器为员工名册创建约束定义。

使用此流程创建一个 硬约束，使其没有员工分配在之前的 **10** 小时后开始的转换。

流程

1. 在 **Business Central** 中，前往 **Menu** → **Design** → **Projects** 并点项目名称。
2. 点 **Add Asset** → **DRL** 文件。
3. 在 **DRL** 文件名 字段中，键入 **complexScoreRules**。
4. 选择 **employees rostering.employeerostering** 软件包。
5. 点 **+Ok** 创建 **DRL** 文件。
6. 在 **DRL** 设计器的 **Model** 选项卡中，将 **Employee10HourShiftSpace** 规则定义为 **DRL** 文件：

```
package employeerostering.employeerostering;

rule "Employee10HourShiftSpace"
    when
        $shiftAssignment : ShiftAssignment( $employee : employee != null, $shiftEndDateTime :
shift.timeslot.endTime)
        ShiftAssignment( this != $shiftAssignment, $employee == employee, $shiftEndDateTime
<= shift.timeslot.endTime,
```

```
            $shiftEndDateTime.until(shift.timeslot.startTime,
java.time.temporal.ChronoUnit.HOURS) <10)
    then
        scoreHolder.addHardConstraintMatch(kcontext, -1);
end
```

7. 点 **Save** 保存 **DRL** 文件。

有关创建 **DRL** 文件的更多信息，请参阅使用 **DRL** 规则设计决策服务。

5.2.5. 使用指导规则为员工名单创建规则

您可以使用 **Business Central** 中的指导规则设计人员创建为员工名单定义硬和软约束的规则。

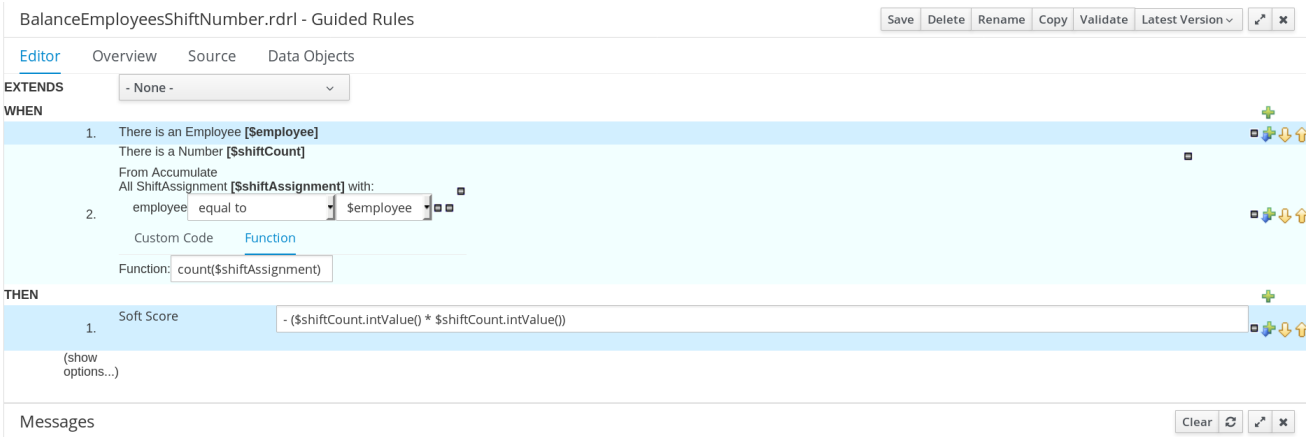
5.2.5.1. 指导规则

指导规则是您通过规则创建在业务中心基于 **UI** 的指导规则设计者中创建的业务规则。指导规则设计器根据所定义规则的数据对象，提供可接受输入的字段和选项。您定义的指南规则与所有其他规则资产一样编译为 **Drools Rule Language(DRL)**规则。

与指导规则相关的所有数据对象都必须位于与指导规则相同的项目软件包中。默认导入同一软件包中的资产。创建必要的数据对象和指导规则后，您可以使用指南规则设计器的 **Data Objects** 选项卡来验证所有所需数据对象是否已列出或导入其他现有数据对象，方法是添加新项目。

5.2.5.2. 创建用于平衡员工切换数量的指导规则

BalanceEmployeesShiftNumber 指导规则创建一个软约束，确保以尽可能均匀地平衡的方式为员工分配切换。它通过创建一个分数影响，在变送变小时增加。分数公式由规则实施，使 **Solver** 以更均衡的方式分发转换。



流程

1. *在 **Business Central** 中，前往 **Menu** → **Design** → **Projects** 并点项目名称。*

2. *点 **Add Asset** → **Guided Rule**。*

3. *输入 **BalanceEmployeesShiftNumber** 作为 引导规则 名称，再选择 **employees rostering.employeerostering** 软件包。*

4. *点 **Ok** 创建规则资产。*

5. *点 **WHEN** 字段中的*

*来添加 **WHEN** 条件。*

6. *在规则窗口的 **Add a** 条件 中选择 **Employee**。单击 **+Ok**。*

7. *单击 **Employee** 条件来修改限制，并添加变量名称 **$employee**。*

8. *添加来自 加速 的 **WHEN** 条件。*

   a. *在 **From Accumulate** 条件 上方点击 添加模式 并选择 **Number** 作为事实类型，从下拉列表中选择 **Number**。*

   b. *将变量名称 **$shiftCount** 添加到 **Number** 条件中。*

   c. *在 **From Accumulate** 条件下，单击添加模式，然后从下拉列表中选择 **ShiftAssignment fact** 类型。*

   d. *将变量名称 **$shiftAssignment** 添加到 **ShiftAssignment fact** 类型。*

   e. *再次单击 **ShiftAssignment** 条件，然后从 **Add a restrictions on a field** 下拉列表中*

选择 **employees**。

f. 从 **employees** 约束旁边的下拉列表中选择 等于。

g. 点击下拉按钮旁的  图标添加变量，然后点击 **Field value** 窗口中的 **Bound** 变量。

h. 从下拉列表中选择 **$employee**。

i. 在 **Function** 框中，键入 **count($shiftAssignment)**。

9. 点 **wordpress N** 字段中的  来添加 **wordpressN** 条件。

10. 在 **Add a new action** 窗口中选择 **Modify Soft Score**。单击 **+Ok**。

a. 在方框中输入以下表达式：
-（$shiftCount.intValue（）*$shiftCount.intValue（））

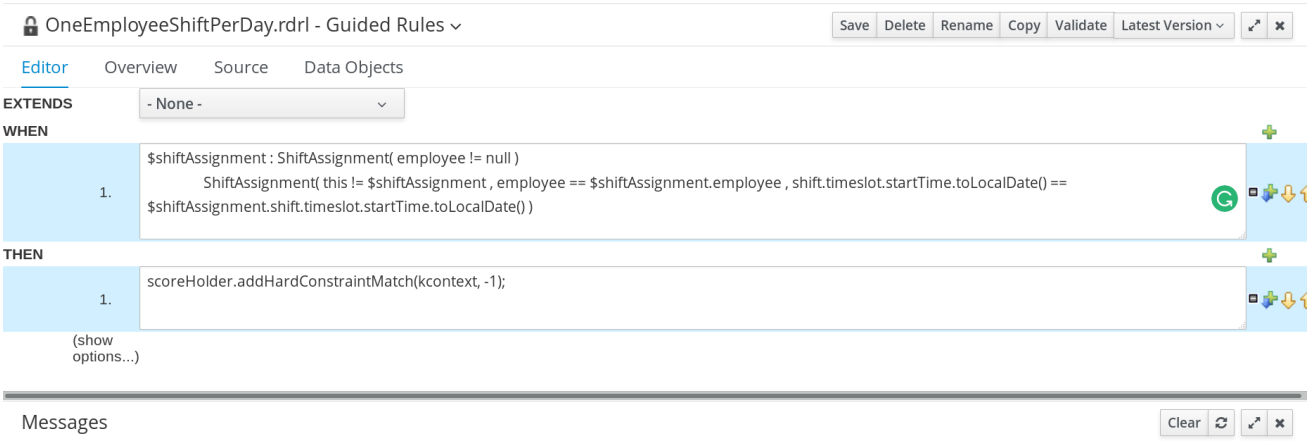
11. 单击右上角的 **Validate** 来检查所有规则条件是否有效。如果规则验证失败，解决错误消息中描述的任何问题，查看规则中的所有组件，然后重试验证规则直到规则通过为止。

12. 单击 **Save** 以保存该规则。

有关创建指南规则的更多信息，请参阅使用指导规则 设计决策服务。

5.2.5.3. 为每天不一换个变化创建指导规则

OneEmployeeShiftPerDay 指导规则创建了一个硬约束，员工不会超过一天的转变。在员工指定示例中，这个约束是使用指导的规则设计人员创建的。



流程

1. *在 **Business Central** 中，前往 **Menu** → **Design** → **Projects** 并点项目名称。*

2. *点 **Add Asset** → **Guided Rule**。*

3. *输入 **OneEmployeeShiftPerDay** 作为 引导规则 名称，再选择 **employees rostering.employeerostering** 软件包。*

4. *点 **Ok** 创建规则资产。*

5. *点 **WHEN** 字段中的*

*来添加 **WHEN** 条件。*

6. *从 **Add a condition to rule** 窗口中选择 **Free form DRL**。*

7. *在自由表单 **DRL** 框中，输入以下条件：*

```
$shiftAssignment : ShiftAssignment( employee != null )
  ShiftAssignment( this != $shiftAssignment , employee == $shiftAssignment.employee
, shift.timeslot.startTime.toLocalDate() ==
$shiftAssignment.shift.timeslot.startTime.toLocalDate() )
```

此条件指出，不能分配给已经在同一天进行另一个切换分配的员工。

8. 点 **wordpress N** 字段中的

来添加 **wordpressN** 条件。

9. 从 **Add a new action** 窗口中选择 **Add free form DRL**。

10. 在自由表单 **DRL** 框中，输入以下条件：

```
scoreHolder.addHardConstraintMatch(kcontext, -1);
```

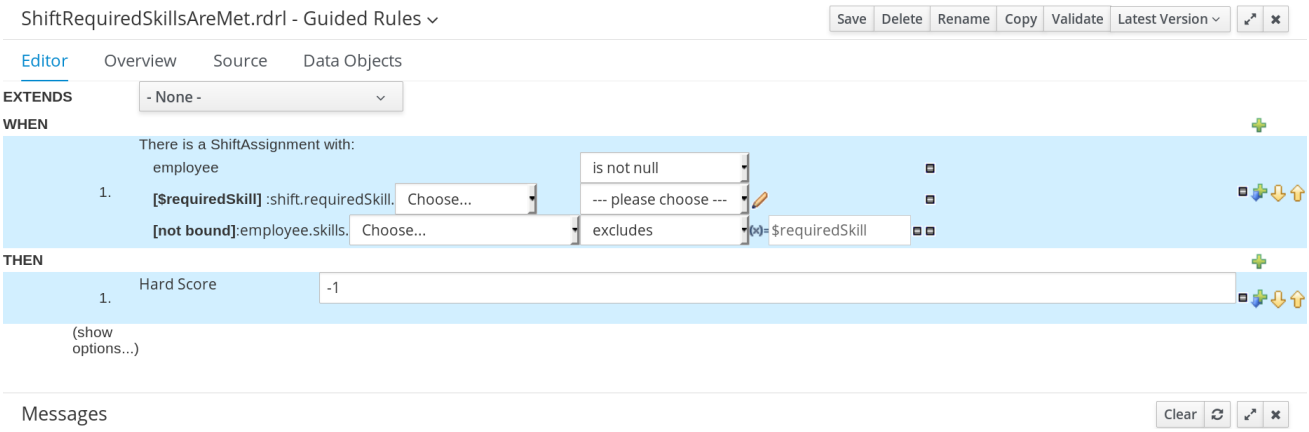
11. 单击右上角的 **Validate** 来检查所有规则条件是否有效。如果规则验证失败，解决错误消息中描述的任何问题，查看规则中的所有组件，然后重试验证规则直到规则通过为止。

12. 单击 **Save** 以保存该规则。

有关创建指南规则的更多信息，请参阅使用指导规则 设计决策服务。

5.2.5.4. 创建一条指导规则以满足不断变化的要求技能

ShiftReqiredSkillsAreMet guided 规则创建了一个硬约束，以确保所有转换都被分配了一组正确技能的员工。在员工指定示例中，这个约束是使用指导的规则设计人员创建的。



流程

1. 在 **Business Central** 中，前往 **Menu** → **Design** → **Projects** 并点项目名称。

2. 点 **Add Asset → Guided Rule**。

3. 输入 **ShiftReqiredSkillsAreMet** 作为 引导规则 名称并选择 **employees rostering.employeerostering** 软件包。

4. 点 **Ok** 创建规则资产。

5. 点 **WHEN** 字段中的

来添加 **WHEN** 条件。

6. 选择 **Add a condition to rule** 窗口中的 **ShiftAssignment**。单击 **+Ok**。

7. 单击 **ShiftAssignment** 条件，然后从 **Add a limits on a field** 下拉列表中选择 **employees**。

8. 在设计人员中，单击 员工 旁边的下拉列表，然后选择 不是 **null**。

9. 单击 **ShiftAssignment** 条件，然后单击 **Expression** 编辑器。

   a. 在设计人员中，单击 **[not bound]** 以打开 **Expression** 编辑器，并将表达式绑定到变量 **$requiredSkill**。点 **Set**。

   b. 在设计人员（ **/requiredSkill** 旁边），从第一个下拉列表中选择 转换，然后从下一下拉列表中选择 **requiredSkill**。

10. 单击 **ShiftAssignment** 条件，然后单击 **Expression** 编辑器。

    a. 在设计人员（ **[not bound]** 旁边），从第一个下拉列表中选择 员工，然后从下一下拉列表中选择 技能。

'

b. 将下一个下拉列表保留为 *选择*。

c. 在下一个下拉菜单中，请更改 *以排除* 。

d. 点击 *排除* 的

图标，在 **Field value** 窗口中点击 **New formula** 按钮。

e. 在公式框中输入 **$requiredSkill**。

11. 点 **wordpress N** 字段中的

来添加 **wordpressN** 条件。

12. 在 **Add a new action** 窗口中选择 **Modify Hard Score**。单击 **+Ok**。

13. 在分数操作框中输入 **-1**。

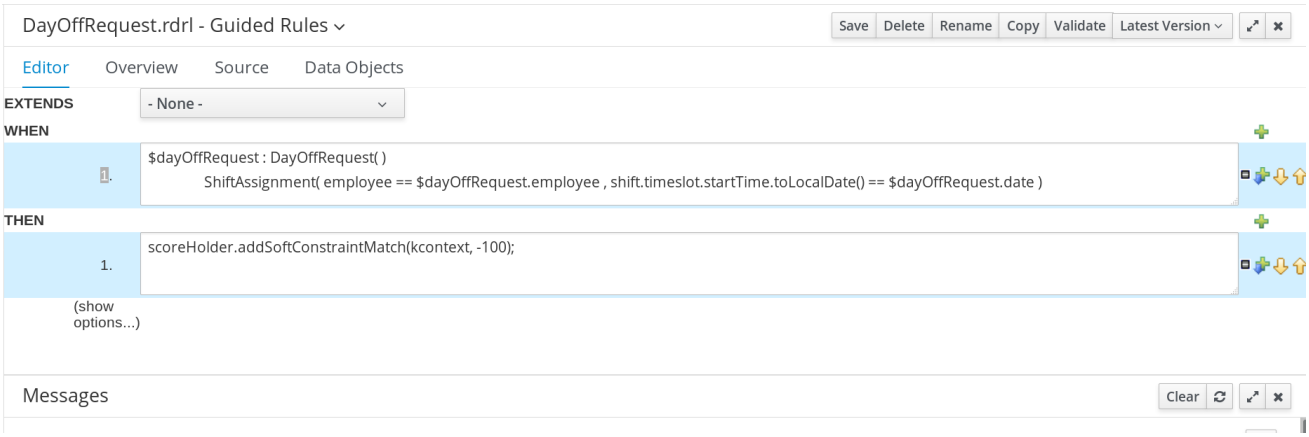
14. 单击右上角的 **Validate** 来检查所有规则条件是否有效。如果规则验证失败，解决错误消息中描述的任何问题，查看规则中的所有组件，然后重试验证规则直到规则通过为止。

15. 单击 **Save** 以保存该规则。

有关创建指南规则的更多信息，请参阅使用指导规则 设计决策服务。

5.2.5.5. 创建用于管理开箱即用请求的指南规则

DayOffRequest guided 规则会创建一个软约束。如果最初被分配了该转换的人，则这个限制可以重新分配给另一位员工。在员工指定示例中，这个约束是使用指导的规则设计人员创建的。



流程

1. *在 **Business Central** 中，前往 **Menu** → **Design** → **Projects** 并点项目名称。*

2. *点 **Add Asset** → **Guided Rule**。*

3. *输入 **DayOffRequest** 作为 引导规则 名称，再选择 **employees rostering.employeerostering** 软件包。*

4. *点 **Ok** 创建规则资产。*

5. *点 **WHEN** 字段中的*

*来添加 **WHEN** 条件。*

6. *从 **Add a condition to rule** 窗口中选择 **Free form DRL**。*

7. *在自由表单 **DRL** 框中，输入以下条件：*

```
$dayOffRequest : DayOffRequest( )
  ShiftAssignment( employee == $dayOffRequest.employee ,
shift.timeslot.startTime.toLocalDate() == $dayOffRequest.date )
```

如果转移被分配给一个进行当天发出的某个员工，则该条件将指出，该条件可以取消对当天的转换进行取消签名。

8.

点 **wordpress N** 字段中的

来添加 **wordpressN** 条件。

9. 从 **Add a new action** 窗口中选择 **Add free form DRL**。

10. 在自由表单 **DRL** 框中，输入以下条件：

```
scoreHolder.addSoftConstraintMatch(kcontext, -100);
```

11. 单击右上角的 **Validate** 来检查所有规则条件是否有效。如果规则验证失败，解决错误消息中描述的任何问题，查看规则中的所有组件，然后重试验证规则直到规则通过为止。

12. 单击 **Save** 以保存该规则。

有关创建指南规则的更多信息，请参阅使用指导规则 设计决策服务。

5.2.6. 为员工降级创建解决器配置

您可以在 **Business Central** 中创建并编辑 **Solver** 配置。**Solver** 配置设计器会创建一个解决器配置，可在项目部署后运行。

先决条件

- **Red Hat Process Automation Manager** 下载并安装。
- 您已为员工名录示例创建并配置了所有相关资产。

流程

1. 在 **Business Central** 中，点击 **Menu** → **Projects**，然后点击您的项目打开它。
2. 在 **Assets** 视角中，点 **Add Asset** → **Solver** 配置

3. 在 **Create new Solver 配置** 窗口中，为您的 **Solver** 键入名称 **EmployeeRosteringSolverConfig**，然后点击 **Ok**。

   这会打开 **Solv er** 配置 设计器。

4. 在 **Score Director Factory** 配置部分中，定义包含评分规则定义的 **KIE** 基础。员工的 **rostering** 示例项目使用 **defaultKieBase**。

   a. 选择在 **KIE** 基本中定义的一个 **KIE** 会话。员工的 **rostering** 示例项目使用 **defaultKieSession**。

5. 单击右上角的 **Validate**，以检查 **Score Director Factory** 配置是否正确。如果验证失败，解决错误消息中描述的任何问题，然后重试进行验证，直到配置通过。

6. 点 **Save** 保存 **Solver** 配置。

5.2.7. 为员工降级项目配置 *Solver* 终止

您可以将 **Solver** 配置为在指定时间后终止。默认情况下，计划引擎会提供无限的时间段来解决问题实例。

员工的 **rostering** 示例项目设置为运行 **30** 秒。

先决条件

- 您已为员工名册项目创建了所有相关资产，并在 **Business Central** 中创建了 **EmployeeRosteringSolverConfig solver** 配置，如 *第 5.2.6 节 “为员工降级创建解决器配置”* 所述。

流程

1. 从 **Assets** 视角打开 **EmployeeRosteringSolverConfig**。这将打开 **Solv er** 配置 设计器。

2. 在 **Termination** 部分，点 **Add** 在所选逻辑组中创建新终止元素。

3. 从下拉列表中选择所花费的 终止类型。这会在终止配置中添加为输入字段。

4. 使用时间元素旁边的箭头来调整 **30** 秒的时间。

5. 单击右上角的 **Validate**，以检查 **Score Director Factory** 配置是否正确。如果验证失败，解决错误消息中描述的任何问题，然后重试进行验证，直到配置通过。

6. 点 **Save** 保存 **Solver** 配置。

5.3. 使用 REST API 访问解决问题

在部署或重新创建示例地址后，您可以使用 **REST API** 访问它。

您必须使用 **REST API** 注册解决器实例。然后，您可以提供数据集并检索优化的解决方案。

先决条件

- 根据本文档中前面的章节设置和部署员工的 **rostering** 项目。您可以部署示例项目，如 *第 5.1 节 “在 Business Central 中部署 employees rostering 示例项目”* 所述，或者重新创建项目，如 *第 5.2 节 “重新排序员工降级示例项目”* 所述。

5.3.1. 使用 REST API 注册 Solver

您必须使用 **REST API** 注册 **solver** 实例，然后才能使用 **solver**。

每个临时解决方案都能够一次优化一个规划问题。

流程

1. 使用以下标头创建 **HTTP** 请求：

```
authorization: admin:admin
X-KIE-ContentType: xstream
content-type: application/xml
```

2. 使用以下请求注册 **Solver**：

PUT

http://localhost:8080/kie-server/services/rest/server/containers/employeerostering_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver

请求正文

```
<solver-instance>
  <solver-config-file>employeerostering/employeerostering/EmployeeRosteringSolverConfig.solver.xml</solver-config-file>
</solver-instance>
```

5.3.2. 使用 REST API 调用 Solver

在注册了解决器实例后，您可以使用 **REST API** 将数据设置为解决方法，并检索优化的解决方案。

流程

1. 使用以下标头创建 **HTTP** 请求：

```
authorization: admin:admin
X-KIE-ContentType: xstream
content-type: application/xml
```

2. 使用数据收集向 **Solver** 提交请求，如下例所示：

POST

http://localhost:8080/kie-server/services/rest/server/containers/employeerostering_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver/state/solving

请求正文

```
<employeerostering.employeerostering.EmployeeRoster>
  <employeeList>
    <employeerostering.employeerostering.Employee>
      <name>John</name>
      <skills>
        <employeerostering.employeerostering.Skill>
          <name>reading</name>
```

```
      </employeerostering.employeerostering.Skill>
    </skills>
  </employeerostering.employeerostering.Employee>
  <employeerostering.employeerostering.Employee>
    <name>Mary</name>
    <skills>
      <employeerostering.employeerostering.Skill>
        <name>writing</name>
      </employeerostering.employeerostering.Skill>
    </skills>
  </employeerostering.employeerostering.Employee>
  <employeerostering.employeerostering.Employee>
    <name>Petr</name>
    <skills>
      <employeerostering.employeerostering.Skill>
        <name>speaking</name>
      </employeerostering.employeerostering.Skill>
    </skills>
  </employeerostering.employeerostering.Employee>
 </employeeList>
 <shiftList>
  <employeerostering.employeerostering.Shift>
    <timeslot>
      <startTime>2017-01-01T00:00:00</startTime>
      <endTime>2017-01-01T01:00:00</endTime>
    </timeslot>
    <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee/skills/emplo
yeerostering.employeerostering.Skill"/>
  </employeerostering.employeerostering.Shift>
  <employeerostering.employeerostering.Shift>
    <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
    <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emp
loyeerostering.employeerostering.Skill"/>
  </employeerostering.employeerostering.Shift>
  <employeerostering.employeerostering.Shift>
    <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
    <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emp
loyeerostering.employeerostering.Skill"/>
  </employeerostering.employeerostering.Shift>
 </shiftList>
 <skillList>
  <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee/skills/employe
erostering.employeerostering.Skill"/>
  <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emplo
yeerostering.employeerostering.Skill"/>
  <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emplo
yeerostering.employeerostering.Skill"/>
 </skillList>
 <timeslotList>
  <employeerostering.employeerostering.Timeslot
```

```
reference="../../shiftList/employeerostering.employeerostering.Shift/timeslot"/>
  </timeslotList>
  <dayOffRequestList/>
  <shiftAssignmentList>
    <employeerostering.employeerostering.ShiftAssignment>
      <shift reference="../../../shiftList/employeerostering.employeerostering.Shift"/>
    </employeerostering.employeerostering.ShiftAssignment>
    <employeerostering.employeerostering.ShiftAssignment>
      <shift reference="../../../shiftList/employeerostering.employeerostering.Shift[3]"/>
    </employeerostering.employeerostering.ShiftAssignment>
    <employeerostering.employeerostering.ShiftAssignment>
      <shift reference="../../../shiftList/employeerostering.employeerostering.Shift[2]"/>
    </employeerostering.employeerostering.ShiftAssignment>
  </shiftAssignmentList>
</employeerostering.employeerostering.EmployeeRoster>
```

3. *请求最佳解决方案以寻求规划问题：*

GET

http://localhost:8080/kie-server/services/rest/server/containers/employeerostering_1.0.0-SNAPSHOT/solvers/EmployeeRosteringSolver/bestsolution

响应示例

```
<solver-instance>
  <container-id>employee-rostering</container-id>
  <solver-id>solver1</solver-id>
  <solver-config-file>employeerostering/employeerostering/EmployeeRosteringSolverConfig.solver.xml</solver-config-file>
  <status>NOT_SOLVING</status>
  <score scoreClass="org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore">0hard/0soft</score>
  <best-solution class="employeerostering.employeerostering.EmployeeRoster">
    <employeeList>
      <employeerostering.employeerostering.Employee>
        <name>John</name>
        <skills>
          <employeerostering.employeerostering.Skill>
            <name>reading</name>
          </employeerostering.employeerostering.Skill>
        </skills>
      </employeerostering.employeerostering.Employee>
      <employeerostering.employeerostering.Employee>
        <name>Mary</name>
        <skills>
```

```
        <employeerostering.employeerostering.Skill>
          <name>writing</name>
        </employeerostering.employeerostering.Skill>
      </skills>
    </employeerostering.employeerostering.Employee>
    <employeerostering.employeerostering.Employee>
      <name>Petr</name>
      <skills>
        <employeerostering.employeerostering.Skill>
          <name>speaking</name>
        </employeerostering.employeerostering.Skill>
      </skills>
    </employeerostering.employeerostering.Employee>
  </employeeList>
  <shiftList>
    <employeerostering.employeerostering.Shift>
      <timeslot>
        <startTime>2017-01-01T00:00:00</startTime>
        <endTime>2017-01-01T01:00:00</endTime>
      </timeslot>
      <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee/skills/emplo
yeerostering.employeerostering.Skill"/>
    </employeerostering.employeerostering.Shift>
    <employeerostering.employeerostering.Shift>
      <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
      <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emp
loyeerostering.employeerostering.Skill"/>
    </employeerostering.employeerostering.Shift>
    <employeerostering.employeerostering.Shift>
      <timeslot reference="../../employeerostering.employeerostering.Shift/timeslot"/>
      <requiredSkill
reference="../../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emp
loyeerostering.employeerostering.Skill"/>
    </employeerostering.employeerostering.Shift>
  </shiftList>
  <skillList>
    <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee/skills/employe
erostering.employeerostering.Skill"/>
    <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[3]/skills/emplo
yeerostering.employeerostering.Skill"/>
    <employeerostering.employeerostering.Skill
reference="../../employeeList/employeerostering.employeerostering.Employee[2]/skills/emplo
yeerostering.employeerostering.Skill"/>
  </skillList>
  <timeslotList>
    <employeerostering.employeerostering.Timeslot
reference="../../shiftList/employeerostering.employeerostering.Shift/timeslot"/>
  </timeslotList>
  <dayOffRequestList/>
  <shiftAssignmentList/>
```

```
    <score>0hard/0soft</score>
  </best-solution>
</solver-instance>
```

第 6 章 OPTAPLANNER 和 QUARKUS 入门

您可以使用 https://code.quarkus.redhat.com 网站生成红帽构建的 OptaPlanner Quarkus Maven 项目，并自动添加并配置要在应用程序中使用的扩展。然后您可以下载 Quarkus Maven 存储库，或使用您项目中的在线 Maven 存储库。

6.1. APACHE MAVEN 和 RED HAT BUILD OF QUARKUS

Apache Maven 是 Java 应用程序开发中使用的分布式构建自动化工具，用于创建、管理和构建软件项目。Maven 使用名为 Project Object Model(POM)文件的标准配置文件来定义项目并管理构建流程。POM 文件描述使用 XML 文件生成的项目打包和输出的模块和组件依赖关系、构建顺序和目标。这可确保以正确、一致的方式构建项目。

Maven 存储库

Maven 存储库存储 Java 库、插件和其他构建构件。默认公共存储库是 Maven 2 中央存储库，但存储库可以是专有仓库和内部的，以在开发团队之间共享常见工件。还可从第三方提供存储库。

您可以将在线 Maven 存储库与 Quarkus 项目一起使用，也可以下载红帽 Quarkus Maven 存储库构建。

Maven 插件

Maven 插件是定义实现一个或多个目标的 POM 文件的组成部分。Quarkus 应用程序使用以下 Maven 插件：

- Quarkus Maven 插件(quarkus-maven-plugin): Enables Maven 创建 Quarkus 项目，支持生成 uber-JAR 文件，并提供开发模式。
- Maven Surefire 插件(maven-surefire-plugin): 在构建生命周期的测试阶段用来对应用程序执行单元测试。插件会生成包含测试报告的文本和 XML 文件。

6.1.1. 为在线存储库配置 Maven settings.xml 文件

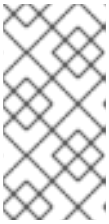
您可以通过配置用户 settings.xml 文件，将在线 Maven 存储库用于 Maven 项目。这是推荐的方法。与共享服务器上的存储库管理器或存储库一起使用的 Maven 设置可以提供更好的控制和管理项目。

注意

当您通过修改 **Maven settings.xml** 文件来配置存储库时，更改会应用到所有 **Maven** 项目。

流程

1. 在文本编辑器中打开 **Maven ~/.m2/settings.xml** 文件或集成开发环境**(IDE)**。



注意

如果 **~/.m2/** 目录中没有 **settings.xml** 文件，请将 **$MAVEN_HOME/.m2/conf/** 目录中的 **settings.xml** 文件复制到 **~/.m2/** 目录中。

2. 在 **settings.xml** 文件的 **<profiles >** 元素中添加以下行：

```
<!-- Configure the Maven repository -->
<profile>
 <id>red-hat-enterprise-maven-repository</id>
 <repositories>
  <repository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>https://maven.repository.redhat.com/ga/</url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>
    <enabled>false</enabled>
   </snapshots>
  </repository>
 </repositories>
 <pluginRepositories>
  <pluginRepository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>https://maven.repository.redhat.com/ga/</url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>
    <enabled>false</enabled>
   </snapshots>
  </pluginRepository>
 </pluginRepositories>
</profile>
```

3. 将以下行添加到 **settings.xml** 文件的 **< activeProfiles >** 元素，并保存文件。

```
<activeProfile>red-hat-enterprise-maven-repository</activeProfile>
```

6.1.2. 下载并配置 Quarkus Maven 存储库

如果您不想使用在线 **Maven** 存储库，您可以下载并配置 **Quarkus Maven** 存储库，以使用 **Maven** 创建 **Quarkus** 应用程序。**Quarkus Maven** 存储库包含许多 **Java** 开发人员通常用来构建应用程序的要求。这个步骤描述了如何编辑 **settings.xml** 文件来配置 **Quarkus Maven** 存储库。

注意

当您通过修改 **Maven settings.xml** 文件来配置存储库时，更改会应用到所有 **Maven** 项目。

流程

1. 从红帽客户门户网站的 **Software Downloads** 页面（需要登录）下载 **Quarkus Maven repository ZIP** 文件。

2. 展开下载的存档。

3. 将目录改为 **~/.m2/** 目录，并在文本编辑器或集成开发环境(IDE)中打开 **Maven settings.xml** 文件。

4. 将以下行添加到 **settings.xml** 文件的 **<profiles >** 元素，其中 **QUARKUS_MAVEN_REPOSITORY** 是您下载的 **Quarkus Maven** 存储库的路径。**QUARKUS_MAVEN_REPOSITORY** 的格式必须是 **file://$PATH**，如 **file:///home/userX/rh-quarkus-2.13.GA-maven-repository/maven-repository**。

```
<!-- Configure the Quarkus Maven repository -->
<profile>
  <id>red-hat-enterprise-maven-repository</id>
  <repositories>
    <repository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>QUARKUS_MAVEN_REPOSITORY</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
```

```
      <enabled>false</enabled>
    </snapshots>
  </repository>
 </repositories>
 <pluginRepositories>
  <pluginRepository>
   <id>red-hat-enterprise-maven-repository</id>
   <url>QUARKUS_MAVEN_REPOSITORY</url>
   <releases>
    <enabled>true</enabled>
   </releases>
   <snapshots>
    <enabled>false</enabled>
   </snapshots>
  </pluginRepository>
 </pluginRepositories>
</profile>
```

5. 将以下行添加到 **settings.xml** 文件的 **< activeProfiles >** 元素，并保存文件。

```
<activeProfile>red-hat-enterprise-maven-repository</activeProfile>
```

重要

如果您的 **Maven** 存储库包含过时的工件，您可能会在构建或部署项目时遇到以下 **Maven** 错误消息之一，其中 **ARTIFACT_NAME** 是缺少的工件和 **PROJECT_NAME** 的名称，即您要构建的项目的名称：

- 缺少工件 **PROJECT_NAME**
- **[ERROR] Failed on project ARTIFACT_NAME; Could not resolve dependencies for PROJECT_NAME**

要解决这个问题，删除位于 **~/.m2/repository** 目录中的本地存储库的缓存版本，以强制下载最新的 **Maven** 工件。

6.2. 使用 MAVEN 插件创建 OPTAPLANNER RED HAT BUILD OF QUARKUS MAVEN 项目

您可以使用 **Apache Maven** 和 **Quarkus Maven** 插件，使用红帽构建的 **OptaPlanner** 和 **Quarkus** 应用程序启动并运行。

先决条件

- 安装了 **OpenJDK 11** 或更高版本。红帽构建的 **Open JDK** 可从红帽客户门户网站的 **Software Downloads** 页面获取（需要登录）。

- 已安装 **Apache Maven 3.6** 或更高版本。**Maven** 可以从 **Apache Maven Project** 网站获得。

流程

1. 在命令终端中，输入以下命令验证 **Maven** 是否使用 **JDK 11**，并且 **Maven** 版本为 **3.6** 或更高版本：

```
mvn --version
```

2. 如果前面的命令没有返回 **JDK 11**，请将到 **JDK 11** 的路径添加到 **PATH** 环境变量，然后再次输入前面的命令。

3. 要生成 **Quarkus OptaPlanner quickstart** 项目，请输入以下命令：

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:2.13.Final-redhat-00006:create \
    -DprojectGroupId=com.example \
    -DprojectArtifactId=optaplanner-quickstart  \
    -Dextensions="resteasy,resteasy-jackson,optaplanner-quarkus,optaplanner-quarkus-jackson" \
    -DplatformGroupId=com.redhat.quarkus.platform
    -DplatformVersion=2.13.Final-redhat-00006 \
    -DnoExamples
```

此命令在 **./optaplanner-quickstart** 目录中创建以下元素：

- **Maven** 结构

- **src/main/docker**中的 **Dockerfile** 文件示例

- 应用程序配置文件

表 **6.1. mvn io.quarkus:quarkus-maven-plugin:2.13.Final-redhat-00006:create** 命令中使用的属性

属性	描述
projectGroupId	项目的组 ID。
projectArtifactId	项目的工件 ID。
extensions	用于此项目的以逗号分隔的 Quarkus 扩展列表。如需 Quarkus 扩展的完整列表，请在命令行中输入 **mvn quarkus:list-extensions**。
noExamples	使用项目结构创建项目，但不包括测试或类。

***projectGroupID** 和 **projectArtifactID** 属性的值用于生成项目版本。默认项目版本为 **1 InventoryServiceSNAPSHOT**。*

4. *要查看您的 **OptaPlanner** 项目，请将目录改为 **OptaPlanner Quickstarts** 目录：*

```
cd optaplanner-quickstart
```

5. *检查 **pom.xml** 文件。内容应类似以下示例：*

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.quarkus.platform</groupId>
      <artifactId>quarkus-bom</artifactId>
      <version>2.13.Final-redhat-00006</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>io.quarkus.platform</groupId>
      <artifactId>quarkus-optaplanner-bom</artifactId>
      <version>2.13.Final-redhat-00006</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy-jackson</artifactId>
```

```
    </dependency>
    <dependency>
      <groupId>org.optaplanner</groupId>
      <artifactId>optaplanner-quarkus</artifactId>
    </dependency>
    <dependency>
      <groupId>org.optaplanner</groupId>
      <artifactId>optaplanner-quarkus-jackson</artifactId>
    </dependency>
    <dependency>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-junit5</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

6.3. 使用 CODE.QUARKUS.REDHAT.COM 创建 QUARKUS MAVEN 项目

您可以使用 **code.quarkus.redhat.com** 网站生成红帽构建的 **OptaPlanner Quarkus Maven** 项目，并自动添加并配置要在应用程序中使用的扩展。另外，**code.quarkus.redhat.com** 会自动管理将项目编译到原生可执行文件所需的配置参数。

本节介绍了如何生成 **OptaPlanner Maven** 项目并包含以下主题：

- 指定应用程序的基本详情。
- 选择您要包含在项目中的扩展。
- 使用您的项目文件生成可下载归档。
- 使用自定义命令编译和启动应用程序。

先决条件

- 您有一个 **Web** 浏览器。

流程

1. 在网页浏览器中 打开 **https://code.quarkus.redhat.com**：

2. 指定项目详情：

3. 输入项目的组名称。名称的格式遵循 **Java** 软件包命名约定，如 **com.example**。

4. 输入您要用于从项目生成的 **Maven** 工件的名称，如 **code-with-quarkus**。

5. 选择 **Build Tool > Maven** 指定要创建 **Maven** 项目。您选择的构建工具决定了项目：

   - 生成的项目的目录结构
   - 生成的项目中使用的配置文件格式
   - 用于编译和启动应用程序的自定义构建脚本和命令，会在您生成项目后为您显示 **code.quarkus.redhat.com**

注意

红帽只支持使用 **code.quarkus.redhat.com** 创建 **OptaPlanner Maven** 项目。红帽不支持生成 **Gradle** 项目。

6. 输入要在项目生成的工件中使用的版本。此字段的默认值为 **15000 SNAPSHOT**。建议使用语义版本，但若愿意，您可以使用不同类型的版本。

7. 输入构建工具在打包项目时生成的工件名称。

根据 **Java** 软件包命名惯例，软件包名称应与软件包名称匹配您用于项目的组名称，但您可以指定不同的名称。

注意

code.quarkus.redhat.com 网站自动使用 **OptaPlanner** 的最新版本。您可以在生成项目后手动更改 **pom.xml** 文件中的 **BOM** 版本。

8. 选择以下扩展以作为依赖项包含：

- **RESTEasy JAX-RS (quarkus-resteasy)**
- **resteasy Jackson(quarkus-resteasy-jackson)**
- **OptaPlanner AI constraint solver(optaplanner-quarkus)**
- **OptaPlanner Jackson(optaplanner-quarkus-jackson)**

红帽为列表上的单个扩展提供不同级别的支持，这些扩展由每个扩展名称旁的标签来表示：

- 红帽完全支持 **SUPPORTED** 扩展用于生产环境中的企业应用程序。
- **TECH-PREVIEW** 扩展受红帽在生产环境中的支持，在 技术预览功能支持范围 下受到红帽的支持。
- 红帽在生产环境中不支持 **DEV-SUPPORT** 扩展，但红帽提供的核心功能则由红帽开发人员用于开发新的应用程序。
- 计划使用具有相同功能的较新的技术或实现替换 **DEPRECATED** 扩展。

红帽不支持在生产环境中使用的未标记扩展。

9. 选择 **Generate your application** 确认您的选择并显示覆盖屏幕，其中包含包含您生成的项目的存档的下载链接。**overlay** 屏幕还显示可用于编译和启动应用程序的自定义命令。

10. 选择 **Download the ZIP**，以使用生成的项目文件将存档保存到您的系统中。

11. 提取存档的内容。

12. 进入包含您提取的项目文件的目录：

```
cd <directory_name>
```

13. 以开发模式编译并启动应用程序：

```
./mvnw compile quarkus:dev
```

6.4. 使用 QUARKUS CLI 创建红帽 QUARKUS MAVEN 项目构建

您可以使用 **Quarkus** 命令行界面(CLI)创建 **Quarkus OptaPlanner** 项目。

先决条件

- 已安装 **Quarkus CLI**。如需更多信息，请参阅使用 **Quarkus** 命令行界面 构建 **Quarkus** 应用程序。

流程

1. 创建 **Quarkus** 应用程序：

```
quarkus create app -P io.quarkus:quarkus-bom:2.13.Final-redhat-00006
```

2. 要查看可用的扩展，请输入以下命令：

```
quarkus ext -i
```

这个命令返回以下扩展：

```
optaplanner-quarkus
optaplanner-quarkus-benchmark
optaplanner-quarkus-jackson
optaplanner-quarkus-jsonb
```

3. 输入以下命令在项目的 **pom.xml** 文件中添加扩展：

```
quarkus ext add resteasy-jackson
quarkus ext add optaplanner-quarkus
quarkus ext add optaplanner-quarkus-jackson
```

4. 在文本编辑器中打开 **pom.xml** 文件。该文件应该类似以下示例：

```
<?xml version="1.0"?>
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd" xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.acme</groupId>
  <artifactId>code-with-quarkus-optaplanner</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <properties>
 <compiler-plugin.version>3.8.1</compiler-plugin.version>
 <maven.compiler.parameters>true</maven.compiler.parameters>
 <maven.compiler.source>11</maven.compiler.source>
 <maven.compiler.target>11</maven.compiler.target>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
 <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
 <quarkus.platform.group-id>io.quarkus</quarkus.platform.group-id>
 <quarkus.platform.version>2.13.Final-redhat-00006</quarkus.platform.version>
 <surefire-plugin.version>3.0.0-M5</surefire-plugin.version>
  </properties>
  <dependencyManagement>
 <dependencies>
   <dependency>
     <groupId>${quarkus.platform.group-id}</groupId>
     <artifactId>${quarkus.platform.artifact-id}</artifactId>
     <version>${quarkus.platform.version}</version>
     <type>pom</type>
     <scope>import</scope>
   </dependency>
   <dependency>
     <groupId>io.quarkus.platform</groupId>
     <artifactId>optaplanner-quarkus</artifactId>
     <version>2.2.2.Final</version>
     <type>pom</type>
     <scope>import</scope>
   </dependency>
 </dependencies>
  </dependencyManagement>
  <dependencies>
 <dependency>
   <groupId>io.quarkus</groupId>
   <artifactId>quarkus-arc</artifactId>
 </dependency>
 <dependency>
   <groupId>io.quarkus</groupId>
   <artifactId>quarkus-resteasy</artifactId>
 </dependency>
```

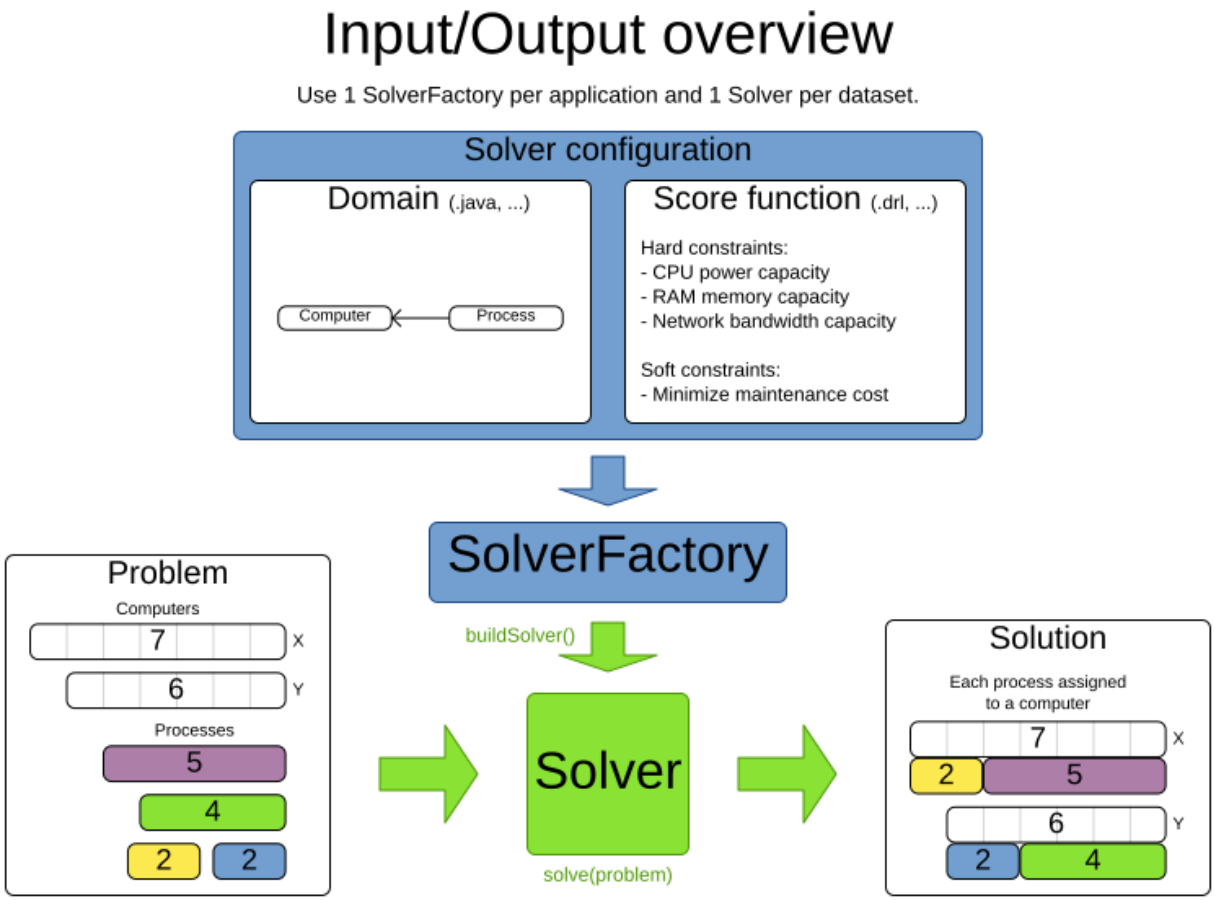
```
<dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-quarkus</artifactId>
</dependency>
<dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-quarkus-jackson</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-resteasy-jackson</artifactId>
</dependency>
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-junit5</artifactId>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>io.rest-assured</groupId>
  <artifactId>rest-assured</artifactId>
  <scope>test</scope>
</dependency>
 </dependencies>
 <build>
<plugins>
  <plugin>
    <groupId>${quarkus.platform.group-id}</groupId>
    <artifactId>quarkus-maven-plugin</artifactId>
    <version>${quarkus.platform.version}</version>
    <extensions>true</extensions>
    <executions>
      <execution>
        <goals>
          <goal>build</goal>
          <goal>generate-code</goal>
          <goal>generate-code-tests</goal>
        </goals>
      </execution>
    </executions>
  </plugin>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>${compiler-plugin.version}</version>
    <configuration>
      <parameters>${maven.compiler.parameters}</parameters>
    </configuration>
  </plugin>
  <plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>${surefire-plugin.version}</version>
    <configuration>
      <systemPropertyVariables>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
        <maven.home>${maven.home}</maven.home>
      </systemPropertyVariables>
```

```
    </configuration>
   </plugin>
  </plugins>
  </build>
  <profiles>
 <profile>
   <id>native</id>
   <activation>
    <property>
     <name>native</name>
    </property>
   </activation>
   <build>
    <plugins>
     <plugin>
      <artifactId>maven-failsafe-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <executions>
       <execution>
        <goals>
         <goal>integration-test</goal>
         <goal>verify</goal>
        </goals>
        <configuration>
         <systemPropertyVariables>
          <native.image.path>${project.build.directory}/${project.build.finalName}-
runner</native.image.path>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
           <maven.home>${maven.home}</maven.home>
         </systemPropertyVariables>
        </configuration>
       </execution>
      </executions>
     </plugin>
    </plugins>
   </build>
   <properties>
    <quarkus.package.type>native</quarkus.package.type>
   </properties>
 </profile>
  </profiles>
</project>
```

部分 III. 红帽构建的 OPTAPLANNER SOLVER

OptaPlanner 解决一个规划问题包括以下步骤：

1. 将计划问题作为标上 **@PlanningSolution** 注释的类建模 （例如， **NQueens** 类）。
2. 配置 **Solver** （例如， 第一个 **Fit** 和 **Tabu Search solver** 用于任何 **NQueens** 实例）。
3. 从您的 数据层加载 一个问题数据（例如 **Four Queens** 实例）。这是规划问题。
4. 使用 **Solv er.solve(problem)**解决问题， 这将返回找到最佳解决方案。



第 7 章 配置红帽构建的 OPTAPLANNER SOLVER

您可以使用以下方法配置 **OptaPlanner solver**：

- 使用 **XML** 文件。
- 使用 **SolverConfig API**。
- 在域模型上添加类注释和 **JavaBean** 属性注解。
- 控制 **OptaPlanner** 用于访问您的域的方法。
- 定义自定义属性。

7.1. 使用 XML 文件配置 OPTAPLANNER SOLVER

您可以使用 **XML** 文件配置解决器。在遵循 **Maven** 目录结构的典型项目中，在构建带有 **SolverFactory** 的 **Solver Factory** 后，**address rConfig XML** 文件位于 **$PROJECT_DIR/src/main/resources/org/optaplanner/examples/<PROJECT>/solver** 目录，其中 **<PROJECT >** 是 **OptaPlanner** 项目的名称。或者，也可以通过 **SolverFactory .createFromXmlFile**（）从文件创建一个 **SolververFactory**。但是，出于便携性的原因，建议使用 **classpath** 资源。

Solver 和 **SolverFactory** 都有一个名为 **Solution_** 的通用类型，它是代表规划问题和解决方案的类。

OptaPlanner 通过更改配置相对容易地切换优化算法。

流程

1. 使用 **Solver Factory** 构建 **Solverver** 实例。
2. 配置解析器配置 **XML** 文件：

a. 定义模型。

b. 定义分数函数。

c. 可选：配置优化算法。

以下示例是 **NQueens** 问题的解析器 **XML** 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <!-- Define the model -->

<solutionClass>org.optaplanner.examples.nqueens.domain.NQueens</solutionClass
>
  <entityClass>org.optaplanner.examples.nqueens.domain.Queen</entityClass>

  <!-- Define the score function -->
  <scoreDirectorFactory>

<scoreDrl>org/optaplanner/examples/nqueens/solver/nQueensConstraints.drl</scor
eDrl>
  </scoreDirectorFactory>

  <!-- Configure the optimization algorithms (optional) -->
  <termination>
    ...
  </termination>
  <constructionHeuristic>
    ...
  </constructionHeuristic>
  <localSearch>
    ...
  </localSearch>
</solver>
```

注意

*在某些环境中，如 **OSGi** 和 **JBoss** 模块，您的 **JAR** 文件中的解决方案、分数 **DRL** 和域类等类路径资源可能不适用于 **optaplanner-core JAR** 文件的默认 **ClassLoader**。在这些情况下，以参数的形式为您的类提供类加载器：*

```
SolverFactory<NQueens> solverFactory =
SolverFactory.createFromXmlResource(
        ".../nqueensSolverConfig.xml",
getClass().getClassLoader());
```

3. *使用解决器配置 **XML** 文件配置 **SolverFactory**，以类路径资源形式提供，如 **ClassLoader.getResource**（） 所定义：*

```
SolverFasctory<NQueens> solverFactory =
SolverFactory.createFromXmlResource(
        "org/optaplanner/examples/nqueens/solver/nqueensSolverConfig.xml");
Solver<NQueens> solver = solverFactory.buildSolver();
```

7.2. 使用 JAVA API 配置 OPTAPLANNER SOLVER

*您可以使用 **SolverConfig API** 配置一个临时解决方案。这在运行时动态更改值特别有用。以下示例在 **NQueens** 项目中构建 **Solver** 之前，会根据系统属性更改正在运行的时间：*

```
SolverConfig solverConfig = SolverConfig.createFromXmlResource(
        "org/optaplanner/examples/nqueens/solver/nqueensSolverConfig.xml");
solverConfig.withTerminationConfig(new TerminationConfig()
        .withMinutesSpentLimit(userInput));

SolverFactory<NQueens> solverFactory = SolverFactory.create(solverConfig);
Solver<NQueens> solver = solverFactory.buildSolver();
```

*解决器配置 **XML** 文件中的每个元素都作为软件包命名空间 **org.optaplanner.core.config** 中的 **Config** 类或一个属性提供。这些配置 类是 **XML** 格式的 **Java** 表示。它们构建软件包命名空间的运行时组件 **org.optaplanner.core.impl.**，并将它们汇编为有效的 **Solver**。*

注意

要为每个用户请求动态配置 **SolverFactory**，在初始化过程中构建模板 **SolverConfig**，并将其复制为每个用户请求的复制构造器。以下示例演示了如何使用 **NQueens** 问题进行此操作：

```
private SolverConfig template;

public void init() {
    template = SolverConfig.createFromXmlResource(
"org/optaplanner/examples/nqueens/solver/nqueensSolverConfig.xml");
    template.setTerminationConfig(new TerminationConfig());
}

// Called concurrently from different threads
public void userRequest(..., long userInput) {
    SolverConfig solverConfig = new SolverConfig(template); // Copy it
    solverConfig.getTerminationConfig().setMinutesSpentLimit(userInput);
    SolverFactory<NQueens> solverFactory =
SolverFactory.create(solverConfig);
    Solver<NQueens> solver = solverFactory.buildSolver();
    ...
}
```

7.3. OPTAPLANNER 注解

您必须指定域模型中的哪些类是规划实体、哪些属性是计划变量等。使用以下方法之一在 **OptaPlanner** 项目中添加注解：

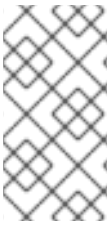
- 在域模型上添加类注释和 **JavaBean** 属性注解。属性注释必须位于 **getter** 方法上，而不是在 **setter** 方法上。注解的 **getter** 方法不需要公开。这是推荐的方法。
- 在域模型上添加类注释和字段注解。注解的字段不需要公共字段。

7.4. 指定 OPTAPLANNER 域访问

默认情况下，**OptaPlanner** 使用反射访问您的域。与直接访问相比，反射比较可靠，但速度慢。另外，您还可以使用 **Gizmo** 将 **OptaPlanner** 配置为访问您的域，这将生成字节码来直接访问您的域的字段和方法，而无需反映。但是，此方法有以下限制：

- 计划注解只能基于公共字段和公共 **getter**。

- ***io.quarkus.gizmo:gizmo*** *必须在* ***classpath*** *上。*



注意

当使用 ***OptaPlanner with Quarkus*** *时，不能应用这些限制，因为* ***Gizmo*** *是默认的域访问类型。*

流程

要在 ***Quarkus*** *之外使用* ***Gizmo***，*在* ***workr*** *配置中设置* ***domainAccessType*** *：*

```
<solver>
  <domainAccessType>GIZMO</domainAccessType>
</solver>
```

7.5. 配置自定义属性

在 ***OptaPlanner*** *项目中，您可以添加自定义属性来解决配置元素，这些元素实例化类，并明确提及自定义属性的文档。*

先决条件

- *您有一个解决方法。*

流程

1. *添加自定义属性。*

   例如，如果您的 ***EasyScoreCalculator*** *具有大量缓存的计算结果，并且您希望在一个基准中增加缓存大小，请添加* ***myCacheSize*** *属性：*

```
<scoreDirectorFactory>
  <easyScoreCalculatorClass>...MyEasyScoreCalculator</easyScoreCalculatorClass>
  <easyScoreCalculatorCustomProperties>
    <property name="myCacheSize" value="1000"/><!-- Override value -->
  </easyScoreCalculatorCustomProperties>
</scoreDirectorFactory>
```

2. 为每个自定义属性添加公共集器，在构建 **Solver** 时调用该属性。

```
public class MyEasyScoreCalculator extends EasyScoreCalculator<MySolution,
SimpleScore> {

    private int myCacheSize = 500; // Default value

    @SuppressWarnings("unused")
    public void setMyCacheSize(int myCacheSize) {
        this.myCacheSize = myCacheSize;
    }

  ...
}
```

大多数值类型都被支持，包括 布尔值、**int**、**double**、**BigDecimal**、**String** 和 **enums**。

第 8 章 OPTAPLANNER SOLVER

解决者找到规划问题的最佳的最佳解决方案。解决者一次只能解决一个规划问题实例。解决者通过 **SolverFactory** 方法构建：

```
public interface Solver<Solution_> {

    Solution_ solve(Solution_ problem);

    ...
}
```

只能从单一线程访问地址，除了 **javadoc** 中被记录为线程安全的方法除外。**solve**（） 方法处理当前的线程。**Hogging** 线程可能会导致 **REST** 服务的 **HTTP** 超时，它需要额外的代码来并行解决多个数据集。要避免这个问题，请使用 **SolverManager**。

8.1. 解决问题

使用解决方法解决规划问题。

先决条件

- 从解决器配置构建的 **Solver**
- 代表规划问题实例的 **@PlanningSolution** 注解

流程

提供规划问题作为 **solve**（） 方法的参数。解决者将返回找到的最佳解决方案。

以下示例解决了 **NQueens** 问题：

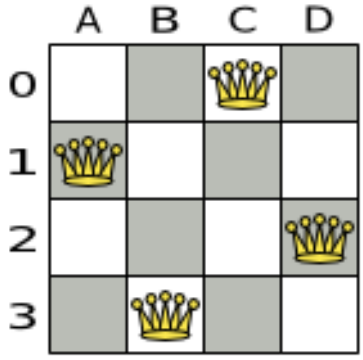
```
NQueens problem = ...;
NQueens bestSolution = solver.solve(problem);
```

在本示例中，**solution** （） 方法会返回一个 **NQueens** 实例，并且每个 **Queen** 分配给 行。

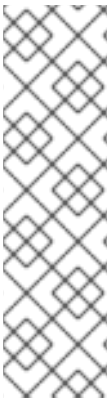
注意

提供给 解决方法(Solution) 方法的解决方案实例可以是部分或完全初始化，这通常是重复规划的情况。

图 8.1. 8ms 中的 Four Queens Puzzle best Solution(Also a Optimal Solution)



解决(Solution) 方法可能需要很长时间，具体要看问题的大小和解决方法配置。Solver 智能地处理可能解决方案的搜索空间，并记住在解决过程中遇到的最佳解决方案。根据很多因素，包括问题的大小、Solver 的时间、解决器配置等等，最佳解决方案 可能是或可能不是最佳解决方案。



注意

提供给方法 解决(Solution) 的解决方案实例已被 Solver 更改，但不要为最好的解决方案出现错误。

方法返回的解决方案实例 (Solution) 或 getBestSolution（） 很可能是向方法 解决 (Solution) 方法的规划克隆，这意味着它是不同的实例。

8.2. SOLVER 环境模式

解决方案环境模式允许您检测实施中的常见错误。它不会影响日志级别。

寻址程序只有一个随机实例。有些解决方法配置使用随机实例，其随机实例数量不止于其他实例。例如，Simulated Anneing 算法依赖于随机数字，而 Tabu Search 仅依赖于它解析分数绑定。环境模式会影响该随机实例的影响。

您可以在解决器配置 XML 文件中设置环境模式。以下示例设置了 FAST_ASSERT 模式：

```
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <environmentMode>FAST_ASSERT</environmentMode>
  ...
</solver>
```

以下列表描述了您可以在解析器配置文件中使用的环境模式：

- **FULL_ASSERT** 模式会开启所有断言，例如，在每次移动中递增分数计算未损坏，导致移动实施、约束、引擎本身等问题的快速失败。这个模式是可重复生成的。它还容易受到入侵，因为它调用了方法 **computeScore**（） 比非**assert** 模式更频繁。**FULL_ASSERT** 模式非常慢，因为它不依赖于增量分数计算。

- **NON_INTRUSIVE_FULL_ASSERT** 模式开启一些断言，在移动实施、约束、引擎本身等方面快速失败。这个模式是可重复生成的。它是非侵入性，因为它不调用 计算**Score**（） 比一个非断模式更频繁。**NON_INTRUSIVE_FULL_ASSERT** 模式非常慢，因为它不依赖于增量分数计算。

- **FAST_ASSERT** 模式开启大多数断言，例如，撤消**Move** 的分数与移动前的分数相同，以在 **Move** 实施、移动实施、约束、引擎本身等时快速出现失败。这个模式是可重复生成的。它还容易受到入侵，因为它调用了方法 **computeScore**（） 比非**assert** 模式更频繁。**FAST_ASSERT** 模式很慢。编写与 **FAST_ASSERT** 模式的简短运行情况的测试案例。

- **REPRODUCIBLE** 模式是默认模式，因为建议在开发过程中。在这个模式中，两个以相同的 **OptaPlanner** 版本运行，会以相同的顺序执行相同的代码。这两个运行在每个步骤都具有相同的结果，但适用下列注释时除外。这可让您持续复制错误。它还可让您对某些重构进行基准测试，如分数约束性能优化，并在运行时非常公平。

注意

尽管使用 **REPRODCIBLE** 模式，但出于以下原因，您的应用程序可能仍不能完全重复：

- 使用 **HashSet** 或另一个 集合，**JVM** 运行之间有不一致的顺序，用于计划实体或规划值的集合，但并非正常问题事实，特别是在解决方案实现中。使用 **LinkedHashSet** 替换它。
- 将有时间的依赖算法合并起来，特别是模拟算法，以及所花费的时间和终止时间。分配的 **CPU** 时间有足够大的差异会影响时间值。将 **Simulated Anneing** 算法替换为 **Late Acceptance** 算法，或者用步骤数终止替换终止所花费的时间。

- **REPRODUCIBLE** 模式可能比 **NON_REPRODUCIBLE** 模式稍慢。如果您的生产环境可从可重复生成的可复制性中受益，则在生产环境中使用此模式。在实践中，如果未指定任何 **seed**，则 **REPRODUCIBLE** 模式使用默认固定随机 **seed**，同时还会禁用某些并发优化，比如加速工作。

- **NON_REPRODUCIBLE** 模式可能比 **REPRODUCIBLE** 模式更快。避免在开发过程中使用它，因为它进行了调试和修复困难。如果在生产环境中复制性不重要，请在生产中使用 **NON_REPRODUCIBLE** 模式。在实践中，如果没有指定任何 **seed**，这个模式不使用固定的随机 **seed**。

8.3. 更改 OPTAPLANNER SOLVER 日志记录级别

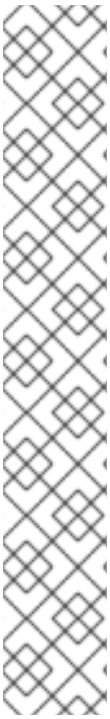
您可以在 **OptaPlanner solver** 中更改日志记录级别，以查看解决器活动。以下列表描述了不同的日志记录级别：

- 错误 ：日志错误，除了以 **RuntimeException** 格式发送到调用代码的除外。

  如果发生错误，**OptaPlanner** 通常会快速失败。它将子类 **RuntimeException** 引发到调用代码的详细消息。为了避免重复的日志消息，它不会以错误的形式将其记录。除非调用代码明确捕获并消除 **RuntimeException**，否则 **Thread's default 'ExceptionHandler** 会将它作为错误记录。同时，代码会破坏进一步危害或模糊处理错误。

- **Warn** ：日志可疑情况

- ***info*** *：每个阶段日志和解决方法*
- *调试 ：记录每个阶段的每个步骤*
- ***Trace*** *：记录每个阶段每个步骤的每个步骤*



注意

指定 ***trace*** *日志记录会显著降低性能。但是，在开发过程中，追踪 日志不足以发现瓶颈。*

对于快速步骤算法（如 ***Late Acceptance*** *和* ***Simulated Anneing****，但不适用于慢速步骤算法，如* ***Tabu Search****），甚至可能会降低性能的速度。*

trace' 和 debug *日志记录都会导致使用大多数附加器进行多线程解决。*

在 ***Eclipse*** *中，对控制台的调试日志往往会导致拥塞，分数计算速度超过* ***10000*** *秒。这个问题都不是* ***IntelliJ*** *或* ***Maven*** *命令行。*

流程

将日志记录级别设置为 ***debug*** *日志记录来查看阶段结束的时间以及执行速度。*

以下示例显示了 ***debug*** *日志记录的输出：*

```
INFO  Solving started: time spent (3), best score (-4init/0), random (JDK with seed 0).
DEBUG     CH step (0), time spent (5), score (-3init/0), selected move count (1), picked move
(Queen-2 {null -> Row-0}).
DEBUG     CH step (1), time spent (7), score (-2init/0), selected move count (3), picked move
(Queen-1 {null -> Row-2}).
DEBUG     CH step (2), time spent (10), score (-1init/0), selected move count (4), picked move
(Queen-3 {null -> Row-3}).
DEBUG     CH step (3), time spent (12), score (-1), selected move count (4), picked move (Queen-0
{null -> Row-1}).
INFO  Construction Heuristic phase (0) ended: time spent (12), best score (-1), score calculation
speed (9000/sec), step total (4).
DEBUG     LS step (0), time spent (19), score (-1),     best score (-1), accepted/selected move count
(12/12), picked move (Queen-1 {Row-2 -> Row-3}).
```

```
DEBUG    LS step (1), time spent (24), score (0), new best score (0), accepted/selected move count (9/12), picked move (Queen-3 {Row-3 -> Row-2}).
INFO  Local Search phase (1) ended: time spent (24), best score (0), score calculation speed (4000/sec), step total (2).
INFO  Solving ended: time spent (24), best score (0), score calculation speed (7000/sec), phase total (2), environment mode (REPRODUCIBLE).
```

所有消耗值的时间都为毫秒。

所有内容都记录到 SLF4J，它是一个简单的日志 facade，它将每个日志消息委派给 Logback、Apache Commons Logging、Log4j 或 java.util.logging。为您选择的日志记录框架添加依赖项到日志适应器。

8.4. 使用 LOGBACK 记录 OPTAPLANNER SOLVER 活动

Logback 是推荐的日志框架，用于 OptaPlanner。使用 Logback 记录 OptaPlanner solver 活动。

先决条件

- *您有一个 OptaPlanner 项目。*

流程

1. *将以下 Maven 依赖项添加到 OptaPlanner 项目的 pom.xml 文件中：*

注意

您不需要添加额外的网桥依赖项。

```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.x</version>
</dependency>
```

2. *在 logback.xml 文件中的 org.optaplanner 软件包配置日志记录级别，如下例所示，其中 <LEVEL> 是一个在 第 8.4 节 “使用 Logback 记录 OptaPlanner solver 活动” 中列出的日志记录级别。*

```
<configuration>

 <logger name="org.optaplanner" level="<LEVEL>"/>

 ...

</configuration>
```

3. 可选：如果您有一个多租户应用程序，其中可能会有多个 **Solver** 实例同时运行，请将每个实例的日志记录分开到单独的文件中：

   a. 使用映射 诊断上下文 **(MDC)**周围 **address** （） 调用：

```
MDC.put("tenant.name",tenantName);
MySolution bestSolution = solver.solve(problem);
MDC.remove("tenant.name");
```

   b. 将您的日志记录器配置为为每个 **${tenant.name}** 使用不同的文件。例如，在 **logback.xml** 文件中使用 **SiftingAppender:**

```
<appender name="fileAppender" class="ch.qos.logback.classic.sift.SiftingAppender">
  <discriminator>
    <key>tenant.name</key>
    <defaultValue>unknown</defaultValue>
  </discriminator>
  <sift>
    <appender name="fileAppender.${tenant.name}" class="...FileAppender">
      <file>local/log/optaplanner-${tenant.name}.log</file>
      ...
    </appender>
  </sift>
</appender>
```

注意

当运行多个 **solvers** 或一个多线程解决时，大多数附加程序（包括控制台）时，会导致使用 **debug** 和 **trace** 日志记录进行拥塞。切换到 **async** 附加程序以避免出现这个问题或关闭 调试 日志记录。

4. 如果 **OptaPlanner** 不识别新级别，请临时添加系统属性 **-Dlogback.LEVEL=true** 以进行故障排除。

8.5. 使用 LOG4J 记录 OPTAPLANNER SOLVER 活动

如果您已经使用 **Log4J**，且您不想切换到更快速的后续后继而使用 **Log4J**，您可以为 **Log4J** 配置 **OptaPlanner** 项目。

先决条件

- 您有一个 **OptaPlanner** 项目
- 您使用 **Log4J** 日志框架

流程

1. 将网桥依赖项添加到项目 **pom.xml** 文件中：

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.x</version>
</dependency>
```

2. 在 **log4j.xml** 文件中在软件包 **org.optaplanner** 上配置日志级别，如下例所示，其中 **< LEVEL& gt;** 是 *第 8.4 节 “使用 Logback 记录 OptaPlanner solver 活动”* 中列出的日志记录级别。

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

  <category name="org.optaplanner">
    <priority value="<LEVEL>" />
  </category>

  ...

</log4j:configuration>
```

3. 可选：如果您有一个多租户应用程序，其中可能会有多个 **Solver** 实例同时运行，请将每个实例的日志记录分开到单独的文件中：

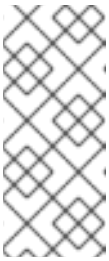
   a. 使用映射 *诊断上下文* **(MDC)**周围 **address** （） 调用：

```
MDC.put("tenant.name",tenantName);
MySolution bestSolution = solver.solve(problem);
MDC.remove("tenant.name");
```

b.

将您的日志记录器配置为为每个 **${tenant.name}** 使用不同的文件。例如，在 **logback.xml** 文件中使用 **SiftingAppender:**

```
<appender name="fileAppender" class="ch.qos.logback.classic.sift.SiftingAppender">
  <discriminator>
    <key>tenant.name</key>
    <defaultValue>unknown</defaultValue>
  </discriminator>
  <sift>
    <appender name="fileAppender.${tenant.name}" class="...FileAppender">
      <file>local/log/optaplanner-${tenant.name}.log</file>
      ...
    </appender>
  </sift>
</appender>
```



注意

当运行多个 **solvers** 或一个多线程解决时，大多数附加程序（包括控制台）时，会导致使用 **debug** 和 **trace** 日志记录进行拥塞。切换到 **async** 附加程序以避免出现这个问题或关闭 调试 日志记录。

8.6. 监控解决器

OptaPlanner 通过 **Micrometer** 公开指标数据，这是 **Java** 应用程序的指标检测库。您可以将 **Micrometer** 与流行监控系统一起使用来监控 **OptaPlanner solver**。

8.6.1. 为 Micrometer 配置 Quarkus OptaPlanner 应用程序

要将 **OptaPlanner Quarkus** 应用程序配置为使用 **Micrometer** 和指定的监控系统，请将 **Micrometer** 依赖项添加到 **pom.xml** 文件中。

先决条件

- 您有一个 **Quarkus OptaPlanner** 应用程序。

流程

1. 将以下依赖项添加到应用程序的 **pom.xml** 文件中，其中 **< MONITORING_SYSTEM& gt;** 是一个由 **Micrometer** 和 **Quarkus** 支持的监控系统：

注意

Prometheus 目前是唯一由 **Quarkus** 支持的监控系统。

```
<dependency>
 <groupId>io.quarkus</groupId>
 <artifactId>quarkus-micrometer-registry-<MONITORING_SYSTEM></artifactId>
</dependency>
```

2. 要在 **development** 模式下运行应用程序，请输入以下命令：

```
mvn compile quarkus:dev
```

3. 要查看应用程序的指标，请在浏览器中输入以下 **URL**：

```
http://localhost:8080/q/metrics
```

8.6.2. 为 Micrometer 配置 Spring Boot OptaPlanner 应用程序

要将 **Spring Boot OptaPlanner** 应用程序配置为使用 **Micrometer** 和指定的监控系统，请将 **Micrometer** 依赖项添加到 **pom.xml** 文件中。

先决条件

- 您有一个 **Spring Boot OptaPlanner** 应用程序。

流程

1. 将以下依赖项添加到应用程序的 **pom.xml** 文件中，其中 **< MONITORING_SYSTEM& gt;** 是一个由 **Micrometer** 和 **Spring Boot** 支持的监控系统：

```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
 <groupId>io.micrometer</groupId>
 <artifactId>micrometer-registry-<MONITORING_SYSTEM></artifactId>
</dependency>
```

2.

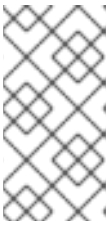
在应用程序的 **application.properties** 文件中添加配置信息。如需更多信息，请参阅 **Micrometer** 网站。

3. 运行以下命令，请运行以下命令：

```
mvn spring-boot:run
```

4. 要查看应用程序的指标，请在浏览器中输入以下 **URL**：

http://localhost:8080/actuator/metrics



注意

使用以下 **URL** 作为 **Prometheus scraper** 路径：**http://localhost:8080/actuator/prometheus**

8.6.3. 为 Micrometer 配置一个普通 Java OptaPlanner 应用程序

要将普通 **Java OptaPlanner** 应用程序配置为使用 **Micrometer**，您必须将您选择的监控系统的 **Micrometer** 依赖项和配置信息添加到项目的 **POM.XML** 文件中。

先决条件

- 您有一个普通 **Java OptaPlanner** 应用程序。

流程

1. 将以下依赖项添加到应用程序的 **pom.xml** 文件中，其中 **< MONITORING_SYSTEM& gt;** 是一个使用 **Micrometer** 配置的监控系统，**& lt;VERSION >** 是您使用的 **Micrometer** 的版本：

```
<dependency>
 <groupId>io.micrometer</groupId>
 <artifactId>micrometer-registry-<MONITORING_SYSTEM></artifactId>
 <version><VERSION></version>
</dependency>
<dependency>
 <groupId>io.micrometer</groupId>
 <artifactId>micrometer-core</artifactId>
 <version>`<VERSION>`</version>
</dependency>
```

2. 将监控系统的 **Micrometer** 配置信息添加到项目 **pom.xml** 文件的开头。如需更多信息，请参阅 **Micrometer** 网站。

3. 在配置信息下方添加以下行，其中 **<MONITORING_SYSTEM>** 是您添加的监控系统：

```
Metrics.addRegistry(<MONITORING_SYSTEM>);
```

以下示例演示了如何添加 **Prometheus** 监控系统：

```
PrometheusMeterRegistry prometheusRegistry = new
PrometheusMeterRegistry(PrometheusConfig.DEFAULT);
try {
    HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
    server.createContext("/prometheus", httpExchange -> {
        String response = prometheusRegistry.scrape();
        httpExchange.sendResponseHeaders(200, response.getBytes().length);
        try (OutputStream os = httpExchange.getResponseBody()) {
            os.write(response.getBytes());
        }
    });
    new Thread(server::start).start();
} catch (IOException e) {
    throw new RuntimeException(e);
}
Metrics.addRegistry(prometheusRegistry);
```

4. 打开监控系统，查看您的 **OptaPlanner** 项目的指标。公开以下指标：

注意

指标的名称和格式因 **registry** 而异。

- **OptaPlanner.solver.errors** ：从测量开始解决过程中发生的错误总数。
- **OptaPlanner.solver.solve-length.active-count:**当前解决的解决方法数量。
- **OptaPlanner.solver.solve-length.seconds-max: run time of longest-running currently active solver.**

- ***OptaPlanner.solver.solve-length.seconds-sum:**每个活跃解决问题持续时间的总和。例如，如果存在两个活跃的 **solvers**，一个只运行三分钟，另一个为一分钟，总计解决时间为 **4** 分钟。*

8.6.4. 其他指标

*如需了解更多详细的监控，您可以在解决器配置中配置 **OptaPlanner** 以监控性能成本的额外指标。以下示例使用 **BEST_SCORE** 和 **SCORE_CALCULATION_COUNT** 指标：*

```
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
 <monitoring>
  <metric>BEST_SCORE</metric>
  <metric>SCORE_CALCULATION_COUNT</metric>
  ...
 </monitoring>
 ...
</solver>
```

您可以在这个配置中启用以下指标：

- ***SOLVE_DURATION**（默认是 **Micrometer** 计量 **ID**：**optaplanner.solver.solve.duration**）：衡量解决最长活动解决时间、活动解决者数量以及所有活跃解决者的累计持续时间。*

- ***ERROR_COUNT**（默认为启用 **Micrometer** 量表 **ID**： **optaplanner.solver.errors**）：衡量在解决问题时发生的错误数量。*

- ***SCORE_CALCULATION_COUNT**（默认为 **Micrometer** 量表 **ID**：**optaplanner.solver.score.calculation.count**）：测量执行 **OptaPlanner** 的分数计算次数。*

- ***BEST_SCORE**（**Micrometer** 计量 **ID: optaplanner.solver.best.score.***）：衡量 **OptaPlanner** 最佳解决方案的性能。分数的每个级别都有单独的计量。例如，对于 **HardSoftScore**，有 **optaplanner.solver.best.score .score** 和 **optaplanner.best.score** 量表。*

- ***STEP_SCORE**（**Micrometer** 计量 **ID**： **optaplanner.solver.step.score.***）：衡量 **OptaPlanner** 采用每个步骤的分数。分数的每个级别都有单独的计量。例如，对于*

HardSoftScore，有 optaplanner.solver.step.score 和 optaplanner.solver.step.score.soft.score 计量。

- *BEST_SOLUTION_MUTATION (Micrometer meter ID: optaplanner.solver.butation)：测量连续最佳解决方案之间更改的规划变量数量。*

- *MOVE_COUNT_PER_STEP （Micrometer 计量 ID：optaplanner.solver.step.move.count）：测量步骤中评估的移动数量。*

- *MEMORY_USE （Micrometer 计量 ID： jvm.memory.used）：测量 JVM 中使用的内存量。此指标不衡量被解析器使用的内存量；同一 JVM 上的两个解决者将报告此指标的相同值。*

- *CONSTRAINT_MATCH_TOTAL_BEST_SCORE （Micrometer 量表 ID: optaplanner.solver.constraint.match.best.score.*）：衡量 OptaPlanner 已发现的最佳解决方案分数影响。分数的每个级别都有单独的计量，每个约束都有标签。例如，对于在软件包 "com.example" 中的约束"Minimize Cost"的 HardSoftScore，有 optaplanner.solver.constraint.match.best.score.hard.score 和 optaplanner.solver.constraint.match。 best.score.soft.score 量表，带有标签 "constraint.package=com.example" 和 "constraint.name=Minimize Cost"。*

- *CONSTRAINT_MATCH_TOTAL_STEP_SCORE （Micrometer 计量 ID: optaplanner.solver.constraint.match.step.score.*）：测量当前步骤中每个约束的影响。分数的每个级别都有单独的计量，每个约束都有标签。例如，对于在软件包 "com.example" 中的约束 "Minimize Cost" 约束一个 HardSoftScore，有 optaplanner.solver.constraint.match.step.score.hard.score 和 optaplanner.solver.constraint.match.step 带有标签 "constraint.com.example" 和 "constraint.name=Minimize Cost" 的 .score.soft.score 量表。*

- *PICKED_MOVE_TYPE_BEST_SCORE_DIFF (Micrometer meter ID: optaplanner.solver.move.type.best.score.diff.*):衡量特定移动类型如何改进最佳解决方案。每个分数级别都有单独的计量，即移动类型的标签。例如，对于进程的计算机的 HardSoftScore 和 ChangeMove，有 optaplanner.solver.move.type.best.score.diff.hard.score 和 optaplanner.solver.move.type.best.score.diff.soft.score meters with tag move.type=ChangeMveoProcess.compute.*

- *PICKED_MOVE_TYPE_STEP_SCORE_DIFF （Micrometer 计量 ID：optaplanner.solver.move.type.step.score.diff.*）：衡量特定移动类型可以改进最佳解决方案。每个分数级别都有单独的计量，即移动类型的标签。例如，对于进程的计算机的 HardSoftScore 和 ChangeMove，有 optaplanner.solver.move.type.step.score.diff.hard.score 和 optaplanner.solver.move.type.step.score.diff.soft.score meters with tag move.type=Move(compute.compute.Process).*

8.7. 配置随机数生成器

许多 **heuristics** 和 **metaheuristics** 依赖于伪随机数字生成器来移动选择，以解决分数绑定、根据移动接受度等问题。在解决过程中，使用相同的随机实例可以重复使用，以提高随机值的可重复性、性能和统一分布。

随机 **seed** 是一个数字，用于初始化伪随机数字生成器。

流程

1. 可选：要更改随机实例的随机看到，请指定 一个随机的 ：

```
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <randomSeed>0</randomSeed>
  ...
</solver>
```

2. 可选：要更改伪随机数字生成器实现，请为下面的解析器配置文件中列出的 **randomType** 属性指定一个值，其中 **< RANDOM_NUMBERATOR>** 是一个伪随机数字生成器：

```
<solver xmlns="https://www.optaplanner.org/xsd/solver"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.optaplanner.org/xsd/solver
https://www.optaplanner.org/xsd/solver/solver.xsd">
  <randomType><RANDOM_NUMBER_GENERATOR></randomType>
  ...
</solver>
```

支持以下 **pseudorandom** 编号生成器：

- **JDK** （默认）：标准随机数字生成器实施**(java.util.Random)**
- **MERSENNE_TWISTER: Commons Math**的实施随机数
- **WELL512A,WELL1024A,WELL19937A,WELL19937C,WELL44497A** 和 **WELL44497B: Random number generator by Commons Math**

*在大多数情况下，**randomType** 属性的值对多个数据集最佳解决方案的平均质量没有重大影响。*

第 9 章 OPTAPLANNER SOLVERMANAGER

Solver Manager 是一个或多个 Solver 实例的常见问题，旨在简化 REST 和其他企业服务中的规划问题。

与 Solver.solve(...) 方法不同，SolverManager 具有以下特征：

- SolverManager.solve(...) 立即返回：它会调度一个问题，以便在不阻止调用线程的情况下进行异步解决。这可避免 HTTP 和其他技术的超时问题。
- SolverManager.solve(...) 可并行解决同一域的多个规划问题。

在内部，SolverManager 管理了一个临时解决方案程序线程的线程池，它调用 Solver.solve(...)，以及处理最佳解决方案更改事件的线程池。

在 Quarkus 和 Spring Boot 中，SolverManager 实例会在您的代码中自动注入。如果您使用 Quarkus 或 Spring Boot 以外的平台，请使用 create(...) 方法构建 SolverManager 实例：

```
SolverConfig solverConfig =
SolverConfig.createFromXmlResource(".../cloudBalancingSolverConfig.xml");
SolverManager<CloudBalance, UUID> solverManager = SolverManager.create(solverConfig,
new SolverManagerConfig());
```

提交至 Solv erManager.solve(...)方法的 每个问题都必须有唯一的问题 ID。之后调用 getSolverStatus(problemId) 或终止Early(problemId) 使用问题 ID 来区分规划问题。问题 ID 必须是一个不可变的类，如 Long、String 或 java.util.UUID。

SolverManagerConfig 类有一个 parallelSolverCount 属性，用于控制并行运行了多少 solvers。例如，如果 parallelSolverCount 属性被设置为 4，您提交了五个问题，则四个问题开始立即解决，在第一个问题之一结束时启动第五个问题。如果这些问题每五分钟解决，则第五个问题需要 10 分钟完成。默认情况下，parallelSolverCount 设置为 AUTO，它解析为 CPU 内核的一半，无论解决者的 moveThreadCount 是什么。

要获得最佳解决方案，在解决终止后通常会使用 SolverJob.getFinalBestSolution（）：

```
CloudBalance problem1 = ...;
UUID problemId = UUID.randomUUID();
```

```
// Returns immediately
SolverJob<CloudBalance, UUID> solverJob = solverManager.solve(problemId, problem1);
...
CloudBalance solution1;
try {
    // Returns only after solving terminates
    solution1 = solverJob.getFinalBestSolution();
} catch (InterruptedException | ExecutionException e) {
    throw ...;
}
```

然而，在用户需要解决方案前解决批处理问题以及用户正在主动等待解决方案时进行实时解决问题。

当前的 **SolverManager** 实施在一个计算机节点上运行，但未来的工作旨在跨云分发解决方法。

9.1. 批量解决问题

批量解决问题正在并行解决多个数据集。批量解决问题尤其有用：

- 夜之间通常没有问题变化，或者没有问题变化。例如，某些组织强制执行截止时间，在 午夜前提交所有关闭请求。
- 解决者会运行时间更长的时间，因为 **nobody** 正在等待结果，而 **CPU** 资源通常更便宜。
- 当员工到达下一工作日时，可以使用解决方案。

流程

为批量解决并行解决的问题，通过 **parallelSolverCount** 限制，每个数据集的 **address (…)** 被调用：

```
public class TimeTableService {

    private SolverManager<TimeTable, Long> solverManager;

    // Returns immediately, call it for every data set
    public void solveBatch(Long timeTableId) {
        solverManager.solve(timeTableId,
                // Called once, when solving starts
                this::findById,
                // Called once, when solving ends
                this::save);
    }
```

```
    public TimeTable findById(Long timeTableId) {...}

    public void save(TimeTable timeTable) {...}

}
```

9.2. 解决和倾听显示进度

当用户等待某个解决方案时，当解决问题时，可能需要等待几分钟或几小时后才会收到结果。为确保一切顺利的用户，通过显示最佳解决方案以及目前获得的最佳分数来显示进度。

流程

1. 要处理中间最佳解决方案，请使用 **solveAndListen(...)**：

```
public class TimeTableService {

    private SolverManager<TimeTable, Long> solverManager;

    // Returns immediately
    public void solveLive(Long timeTableId) {
        solverManager.solveAndListen(timeTableId,
                // Called once, when solving starts
                this::findById,
                // Called multiple times, for every best solution change
                this::save);
    }

    public TimeTable findById(Long timeTableId) {...}

    public void save(TimeTable timeTable) {...}

    public void stopSolving(Long timeTableId) {
        solverManager.terminateEarly(timeTableId);
    }

}
```

此实施使用数据库与 **UI** 通信，后者轮询数据库。更高级的实施将最佳的解决方案直接推送到 **UI** 或消息传递队列。

2. 用户对中间最佳解决方案满意且不想等待更长的时间，请致电 **SolverManager.terminateEarly(problemId)**。

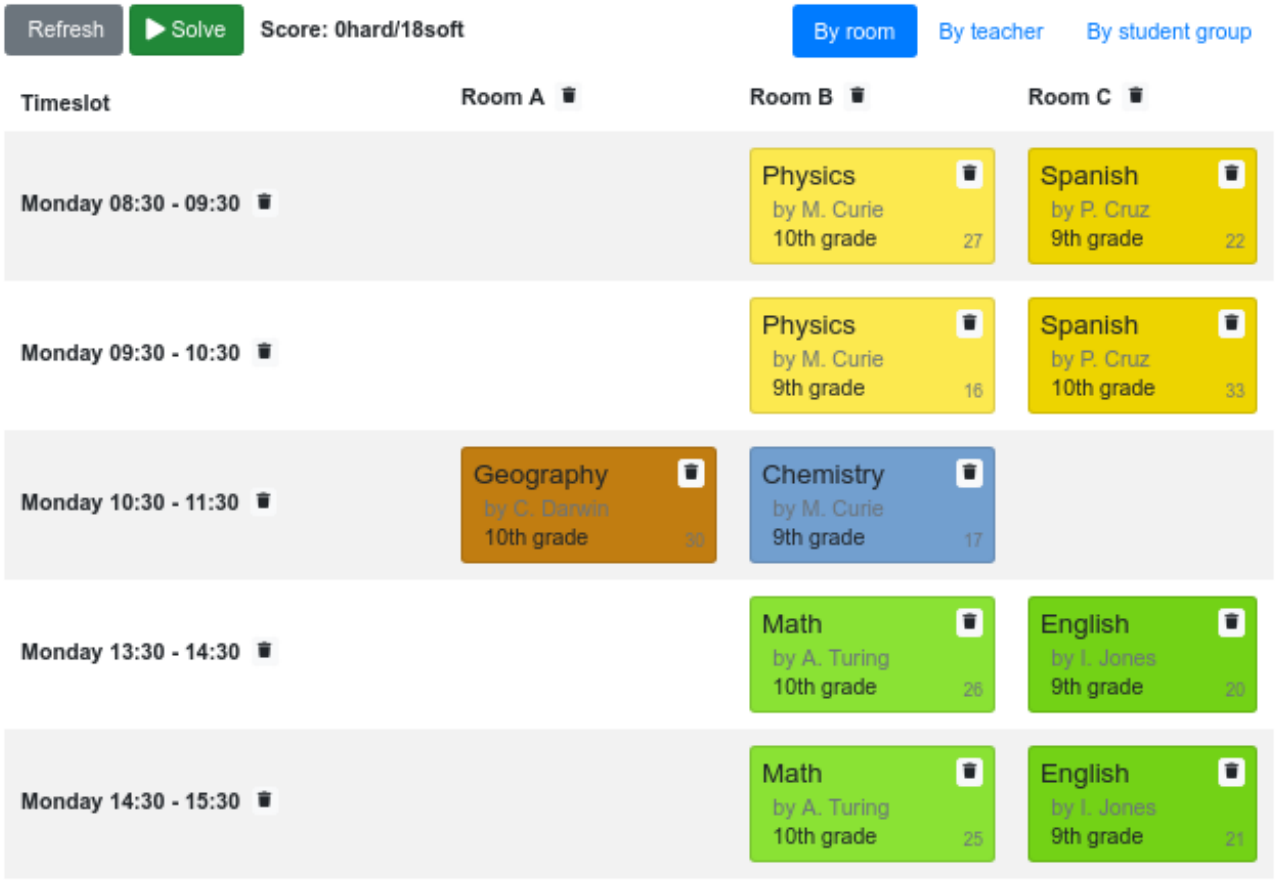
部分 IV. RED HAT BUILD OF OPTAPLANNER QUICK START GUIDE

Red Hat build of OptaPlanner 提供以下快速启动指南，以演示 **OptaPlanner** 如何与不同的 **techologies** 集成：

- **Red Hat build of OptaPlanner on Red Hat build of Quarkus: a school timetable quick start Guide**
- **Red Hat build of OptaPlanner on Red Hat build of Quarkus: a vaccination appointment** 调度程序快速启动指南
- **Red Hat build of OptaPlanner on Spring Boot: a school timetable quick start Guide**
- **Red Hat build of OptaPlanner with Java solvers**：云平衡快速启动指南

第 10 章 RED HAT BUILD OF OPTAPLANNER ON RED HAT BUILD OF QUARKUS: A SCHOOL TIMETABLE QUICK START GUIDE

本指南指导您使用红帽构建的 **OptaPlanner** 约束完成了创建红帽 **Quarkus** 应用程序的构建过程。您将构建一个针对学生和生生而优化院长的 **REST** 应用程序



您的服务会自动将 **Lesson** 实例分配给 **Timeslot** 和 **Room** 实例，方法是使用 **AI** 遵循以下硬和软 调度限制 ：

- 个空间最多可以同时有一处。
- 老师可以同时在上课时进行教学。
- 学员最多可以同时参加。
- 教授者更倾向于在单一房间学习。

- 指导者更倾向于在课间学习顺序课时和不类差距。

数学会上讲，该学校的时间范围是一个 **NP-hard** 问题。这意味着很难扩展。简单地说，通过带有 **brute** 强制的可能组合将花费数百万年时间，而在超级计算机中，甚至在超级计算机中也要花费数百万年时间。幸运的是，**AI** 约束寻址者（如红帽构建的 **OptaPlanner**）具有高级算法，可在合理的时间内提供最接近的解决方案。被视为有合理的时间，它取决于问题的目标。

先决条件

- 安装了 **OpenJDK 11** 或更高版本。红帽构建的 **Open JDK** 可从红帽客户门户网站的 **Software Downloads** 页面获取（需要登录）。
- 已安装 **Apache Maven 3.6** 或更高版本。**Maven** 可以从 **Apache Maven Project** 网站获得。
- 使用 **IntelliJ IDEA**、**VS Code**、**Eclipse** 或 **NetBeans** 等 **IDE**。

10.1. 使用 MAVEN 插件创建 OPTAPLANNER RED HAT BUILD OF QUARKUS MAVEN 项目

您可以使用 **Apache Maven** 和 **Quarkus Maven** 插件，使用红帽构建的 **OptaPlanner** 和 **Quarkus** 应用程序启动并运行。

先决条件

- 安装了 **OpenJDK 11** 或更高版本。红帽构建的 **Open JDK** 可从红帽客户门户网站的 **Software Downloads** 页面获取（需要登录）。
- 已安装 **Apache Maven 3.6** 或更高版本。**Maven** 可以从 **Apache Maven Project** 网站获得。

流程

1. 在命令终端中，输入以下命令验证 **Maven** 是否使用 **JDK 11**，并且 **Maven** 版本为 **3.6** 或更高版本：

```
mvn --version
```

2. 如果前面的命令没有返回 **JDK 11**，请将到 **JDK 11** 的路径添加到 **PATH** 环境变量，然后再

次输入前面的命令。

3. *要生成 **Quarkus OptaPlanner quickstart** 项目，请输入以下命令：*

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:2.13.Final-redhat-00006:create \
    -DprojectGroupId=com.example \
    -DprojectArtifactId=optaplanner-quickstart  \
    -Dextensions="resteasy,resteasy-jackson,optaplanner-quarkus,optaplanner-quarkus-jackson" \
    -DplatformGroupId=com.redhat.quarkus.platform
    -DplatformVersion=2.13.Final-redhat-00006 \
    -DnoExamples
```

*此命令在 **./optaplanner-quickstart** 目录中创建以下元素：*

- ***Maven** 结构*
- ***src/main/docker**中的 **Dockerfile** 文件示例*
- *应用程序配置文件*

***表 10.1. mvn io.quarkus:quarkus-maven-plugin:2.13.Final-redhat-00006:create** 命令中使用的属性*

属性	描述
projectGroupId	项目的组 ID。
projectArtifactId	项目的工件 ID。
extensions	用于此项目的以逗号分隔的 Quarkus 扩展列表。如需 Quarkus 扩展的完整列表，请在命令行中输入 **mvn quarkus:list-extensions**。
noExamples	使用项目结构创建项目，但不包括测试或类。

***projectGroupID** 和 **projectArtifactID** 属性的值用于生成项目版本。默认项目版本为 **1 InventoryServiceSNAPSHOT**。*

4. *要查看您的 **OptaPlanner** 项目，请将目录改为 **OptaPlanner Quickstarts** 目录：*

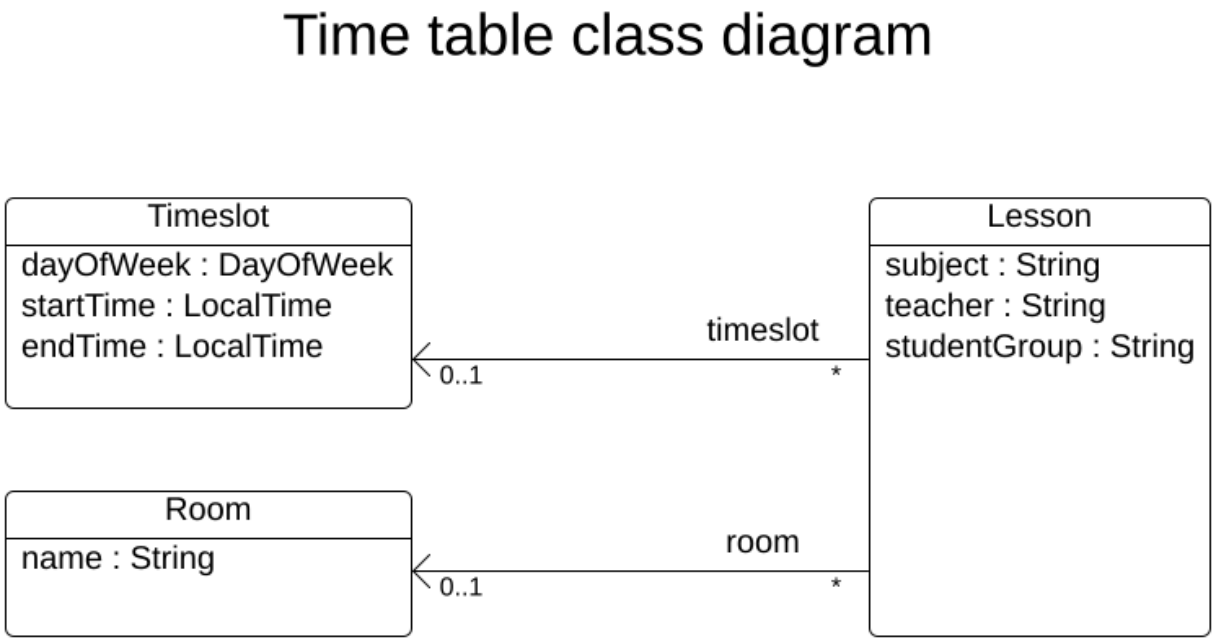
```
cd optaplanner-quickstart
```

5. *检查 **pom.xml** 文件。内容应类似以下示例：*

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.quarkus.platform</groupId>
      <artifactId>quarkus-bom</artifactId>
      <version>2.13.Final-redhat-00006</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
    <dependency>
      <groupId>io.quarkus.platform</groupId>
      <artifactId>quarkus-optaplanner-bom</artifactId>
      <version>2.13.Final-redhat-00006</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy-jackson</artifactId>
  </dependency>
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-quarkus</artifactId>
  </dependency>
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-quarkus-jackson</artifactId>
  </dependency>
  <dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-junit5</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

10.2. 对域对象建模

*红帽构建的 **OptaPlanner** 时间表项目的目标是将每节分配给一个时间段和一个房间。为此，请添加三个类： **Timeslot**、**Lesson** 和 **Room**，如下图所示：*



Timeslot

***Timeslot** 类表示在未教授课程的时间间隔，例如： **Monday 10:30 - 11:30** 或 **Tuesday 13:30 - 14:30 - 14:30**。在这个示例中，所有时间插槽都有相同的持续时间，且在 **lunch** 或其他中断期间没有时间插槽。*

*时间段没有日期，因为高校的计划只每周重复。无法进行持续 规划。次性被称为 问题事实，因为在解决过程中不会有定时实例变化。此类类不需要任何特定于 **OptaPlanner** 的注解。*

room

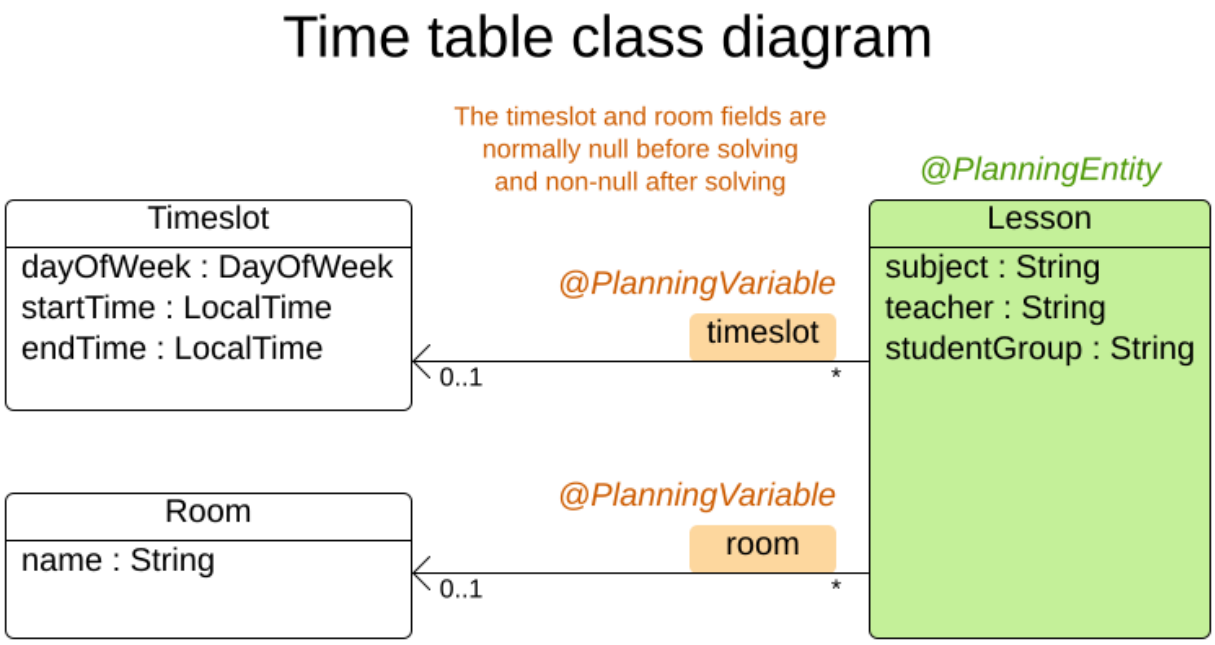
***Room** 类代表了课程讲授的方式，例如 **Room A** 或 **Room B**。在本例中，所有空间都没有容量限制，它们可容纳所有课时。*

*房间 实例在解决过程中不会改变，因此 **Room** 也是 问题。*

lesson

*在课时，由课课课表示，向班级教授一个主题，例如： **A.Turing for a 9th grade** 或 **Chemistry by M.Curie** 为 **10th grade**。如果一个主题每周都会为同一学员组讲授多次，则有多个小的、只能由 **id** 区分的实例。例如，第 **9** 分之时，每周有 **6** 个数学。*

*在寻求过程中，**OptaPlanner** 改变了 **lesson** 类的 **timelot** 和 **room** 字段，从而将每个较小的时间分配给一个时间段和一个房间。 因为 **OptaPlanner** 更改了这些字段，所以 **lesson** 是一个 规划实体 ：*



*上图中的大多数字段包含输入数据，但 **orange** 字段除外。在输入数据中 未分配的 时间和房间 字段 **(null)**，并在输出数据中分配（而不是 **null**）。**OptaPlanner** 在解决过程中更改这些字段。这些字段称为规划变量。为了 **OptaPlanner** 可以识别它们，**timelot** 和 **room** 字段都需要 **@PlanningVariable** 注解。它们包含类**(Lesson)**需要 **@PlanningEntity** 注释。*

流程

1. *创建 **src/main/java/com/example/domain/Timeslot.java** 类：*

```
package com.example.domain;

import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;

    private Timeslot() {
    }

    public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
        this.dayOfWeek = dayOfWeek;
```

```
        this.startTime = startTime;
        this.endTime = endTime;
    }

    @Override
    public String toString() {
        return dayOfWeek + " " + startTime.toString();
    }

    // ********************************
    // Getters and setters
    // ********************************

    public DayOfWeek getDayOfWeek() {
        return dayOfWeek;
    }

    public LocalTime getStartTime() {
        return startTime;
    }

    public LocalTime getEndTime() {
        return endTime;
    }

}
```

请注意，**toString**（） 方法保留了输出短片，以便可以更轻松地读取 **OptaPlanner** 的 **DEBUG** 或 **abrt** 日志，如稍后所示。

2. 创建 **src/main/java/com/example/domain/Room.java** 类：

```
package com.example.domain;

public class Room {

    private String name;

    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    // ********************************
    // Getters and setters
```

```
    // ********************************

    public String getName() {
        return name;
    }

}
```

3. 创建 **src/main/java/com/example/domain/Lesson.java** 类：

```
package com.example.domain;

import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
    }

    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // ********************************
    // Getters and setters
    // ********************************

    public Long getId() {
        return id;
    }

    public String getSubject() {
```

```
        return subject;
    }

    public String getTeacher() {
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {
        return timeslot;
    }

    public void setTimeslot(Timeslot timeslot) {
        this.timeslot = timeslot;
    }

    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
        this.room = room;
    }

}
```

Lesson 类具有 **@PlanningEntity** 注释，因此 **OptaPlanner** 知道这个类在解决过程中发生了变化，因为它包含一个或多个计划变量。

time lot 字段具有 **@PlanningVariable** 注释，因此 **OptaPlanner** 知道它可以更改其值。为了查找要分配给此字段的潜在的 **Timeslot** 实例，**OptaPlanner** 使用 **valueRangeProviderRefs** 属性连接到一个提供 **List<Timeslot>** 的值供应商。有关值范围供应商的信息，请参阅 第 **10.4** 节 “在规划解决方案中收集域对象”。

出于同样原因，**room** 字段还具有 **@PlanningVariable** 注释。

10.3. 定义限制并计算分数

当解决问题时，分数 代表特定解决方案的质量。得分越高。红帽构建的 **OptaPlanner** 寻找最佳解决方案，这是可用时间获得最高分数的解决方案。这可能是 最佳解决方案。

因为可时间的示例用例具有硬和软约束，因此请使用 **HardSoftScore** 类来代表分数：

- 硬限制不得被破坏。例如： 一个房间可以有以上时间。

- 软限制不应该被破坏。例如： 更喜欢在单个房里的教学方式。

硬约束的加法是相对于其他硬约束的权重。软限制会高于其他软限制。硬限制始终大于软限制，无论其对应的权重如何。

要计算分数，您可以实施 **EasyScoreCalculator** 类：

```
public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

   @Override
   public HardSoftScore calculateScore(TimeTable timeTable) {
      List<Lesson> lessonList = timeTable.getLessonList();
      int hardScore = 0;
      for (Lesson a : lessonList) {
         for (Lesson b : lessonList) {
            if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
                  && a.getId() < b.getId()) {
               // A room can accommodate at most one lesson at the same time.
               if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                  hardScore--;
               }
               // A teacher can teach at most one lesson at the same time.
               if (a.getTeacher().equals(b.getTeacher())) {
                  hardScore--;
               }
               // A student can attend at most one lesson at the same time.
               if (a.getStudentGroup().equals(b.getStudentGroup())) {
                  hardScore--;
               }
            }
         }
      }
      int softScore = 0;
      // Soft constraints are only implemented in the "complete" implementation
      return HardSoftScore.of(hardScore, softScore);
   }

}
```

不幸的是，这个解决方案无法进行很好的扩展，因为它是非增量：每次分配给不同的时间段或房间，都会重新评估所有课程来计算新分数。

更好的解决方法是创建一个 **src/main/java/com/example/solver/TimeTableConstraintProvider.java**

类来执行增量分数计算。此类使用 OptaPlanner 的 ConstraintStream API，它由 Java 8 Streams 和 SQL 产生。ConstraintProvider 缩放比 EasyScoreCalculator 更好的度量：O(n)而不是 O(n)。

流程

创建以下 src/main/java/com/example/solver/TimeTableConstraintProvider.java 类：

```
package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
                // Hard constraints
                roomConflict(constraintFactory),
                teacherConflict(constraintFactory),
                studentGroupConflict(constraintFactory),
                // Soft constraints are only implemented in the "complete" implementation
        };
    }

    private Constraint roomConflict(ConstraintFactory constraintFactory) {
        // A room can accommodate at most one lesson at the same time.

        // Select a lesson ...
        return constraintFactory.from(Lesson.class)
                // ... and pair it with another lesson ...
                .join(Lesson.class,
                        // ... in the same timeslot ...
                        Joiners.equal(Lesson::getTimeslot),
                        // ... in the same room ...
                        Joiners.equal(Lesson::getRoom),
                        // ... and the pair is unique (different id, no reverse pairs)
                        Joiners.lessThan(Lesson::getId))
                // then penalize each pair with a hard weight.
                .penalize("Room conflict", HardSoftScore.ONE_HARD);
    }

    private Constraint teacherConflict(ConstraintFactory constraintFactory) {
        // A teacher can teach at most one lesson at the same time.
        return constraintFactory.from(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getTeacher),
                        Joiners.lessThan(Lesson::getId))
```

```
                .penalize("Teacher conflict", HardSoftScore.ONE_HARD);
    }

    private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
        // A student can attend at most one lesson at the same time.
        return constraintFactory.from(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getStudentGroup),
                        Joiners.lessThan(Lesson::getId))
                .penalize("Student group conflict", HardSoftScore.ONE_HARD);
    }

}
```

10.4. 在规划解决方案中收集域对象

TimeTable 实例嵌套了单个 **dataset** 的所有 **Timeslot**、**Room** 和 **Lesson** 实例。另外，由于它包含了 **allon**，每个都具有特定规划变量状态，所以它是一个 规划解决方案，并有分数：

- 如果课程仍然没有被签名，那么它是一个 未初始化 的解决方案，例如分数为 **-4init/0hard/0soft** 的解决方案。
- 如果损坏硬限制，那么它是一种 不可预见的 解决方案，例如分数为 **-2hard/-3soft** 的解决方案。
- 如果遵循所有硬约束，那么它是一个可行的解决方案，例如，分数为 **0hard/-7soft** 的解决方案。

TimeTable 类有一个 **@PlanningSolution** 注释，因此红帽构建的 **OptaPlanner** 知道该类包含所有输入和输出数据。

具体来说，这个类是问题的输入：

- 带有所有时间窗的 **timeslotList** 字段
  - 这是问题事实列表，因为它们在解决过程中不会改变。

- 包含所有房间有一个 **roomList** 字段
  - 这是问题事实列表，因为它们在解决过程中不会改变。
- 带有所有节节的 **lessonList** 字段
  - 这是计划实体列表，因为他们在解决过程中发生了变化。
  - 在每个课 中 ：
    - **timeslot** 和 **room** 字段的值通常为 **null**，因此取消分配。它们是规划变量。
    - 其他字段（如 主题、**S quill r** 和 **studentGroup** ）已被填写。这些字段是问题属性。

但是，这个类也是解决方案的输出：

- 每个 **lessonList** 字段在解决后都有非 **null time lot** 和 **room** 字段
- 代表输出解决方案质量 的分数 字段，如 **0hard/-5soft**

流程

创建 **src/main/java/com/example/domain/TimeTable.java** 类：

```
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
```

```
@PlanningSolution
public class TimeTable {

    @ValueRangeProvider(id = "timeslotRange")
    @ProblemFactCollectionProperty
    private List<Timeslot> timeslotList;

    @ValueRangeProvider(id = "roomRange")
    @ProblemFactCollectionProperty
    private List<Room> roomList;

    @PlanningEntityCollectionProperty
    private List<Lesson> lessonList;

    @PlanningScore
    private HardSoftScore score;

    private TimeTable() {
    }

    public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
            List<Lesson> lessonList) {
        this.timeslotList = timeslotList;
        this.roomList = roomList;
        this.lessonList = lessonList;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public List<Timeslot> getTimeslotList() {
        return timeslotList;
    }

    public List<Room> getRoomList() {
        return roomList;
    }

    public List<Lesson> getLessonList() {
        return lessonList;
    }

    public HardSoftScore getScore() {
        return score;
    }

}
```

值范围供应商

***timeslotList** 字段是一个值范围 **provider**。它拥有 **OptaPlanner** 可以从中选择的 **Timeslot** 实例，分配给 **lesson** 实例的 **timeslot** 字段。**time lotList** 字段具有 **@ValueRangeProvider** 注释，可通过在*

Lesson 中将 id 与 @PlanningVariable 的 valueRangeProviderRefs 匹配。

遵循相同的逻辑，roomList 字段也有 @ValueRangeProvider 注释。

问题事实和规划实体属性

另外，OptaPlanner 需要知道它可能会更改哪些较小的实例，以及如何检索由 TimeTableConstraintProvider 计算的 Timeslot 和 Room 实例。

time lotList 和 roomList 字段具有 @ProblemFactCollectionProperty 注释，因此您的 TimeTableConstraintProvider 可以从这些实例中进行选择。

lessonList 有一个 @PlanningEntityCollectionProperty 注解，因此 OptaPlanner 可在解决过程中更改它们，并且您的 TimeTableConstraintProvider 也可从那些中选择。

10.5. 创建 SOLVER 服务

解决 REST 线程中的规划问题会导致 HTTP 超时问题。因此，Quarkus 扩展注入 SolverManager，它会在单独的线程池中运行 solvers，并可并行处理多个数据集。

流程

创建 src/main/java/org/acme/optaplanner/rest/TimeTableResource.java 类：

```
package org.acme.optaplanner.rest;

import java.util.UUID;
import java.util.concurrent.ExecutionException;
import javax.inject.Inject;
import javax.ws.rs.POST;
import javax.ws.rs.Path;

import org.acme.optaplanner.domain.TimeTable;
import org.optaplanner.core.api.solver.SolverJob;
import org.optaplanner.core.api.solver.SolverManager;

@Path("/timeTable")
public class TimeTableResource {

    @Inject
    SolverManager<TimeTable, UUID> solverManager;

    @POST
```

```
    @Path("/solve")
    public TimeTable solve(TimeTable problem) {
        UUID problemId = UUID.randomUUID();
        // Submit the problem to start solving
        SolverJob<TimeTable, UUID> solverJob = solverManager.solve(problemId, problem);
        TimeTable solution;
        try {
            // Wait until the solving ends
            solution = solverJob.getFinalBestSolution();
        } catch (InterruptedException | ExecutionException e) {
            throw new IllegalStateException("Solving failed.", e);
        }
        return solution;
    }

}
```

此初始实施会等待解决方法完成，这仍然可能导致 **HTTP** 超时。完整的实现可避免 **HTTP** 超时更明确。

10.6. 设置解决者终止时间

如果您的计划应用程序没有终止设置或终止事件，它理论上会永久运行，并最终会导致 **HTTP** 超时错误。要防止这种情况发生，请使用 **optaplanner.solver.termination.spent-limit** 参数指定应用程序终止的时间长度。在大多数应用程序中，将时间至少设置为五分钟**(5m)**。但是，在 **Timetable** 示例中，将解决时间限制为 **5** 秒，这不足以避免 **HTTP** 超时。

流程

使用以下内容创建 **src/main/resources/application.properties** 文件：

```
quarkus.optaplanner.solver.termination.spent-limit=5s
```

10.7. 运行 SCHOOL TIMETABLE 应用程序

创建了 **school timetable** 项目后，以开发模式运行它。在开发模式中，您可以在应用程序运行时更新应用程序源和配置。您的更改将显示在正在运行的应用程序中。

先决条件

- 您已创建了 **school timetable** 项目。

流程

1. 要在开发模式中编译应用程序，请从项目目录中输入以下命令：

```
./mvnw compile quarkus:dev
```

2. 测试 **REST** 服务。您可以使用任何 **REST** 客户端。以下示例使用 **Linux** 命令 **curl** 发送 **POST** 请求：

```
$ curl -i -X POST http://localhost:8080/timeTable/solve -H "Content-Type:application/json" -d
'{"timeslotList":[{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"}],"roomList":
[{"name":"Room A"},{"name":"Room B"}],"lessonList":[{"id":1,"subject":"Math","teacher":"A.
Turing","studentGroup":"9th grade"},{"id":2,"subject":"Chemistry","teacher":"M.
Curie","studentGroup":"9th grade"},{"id":3,"subject":"French","teacher":"M.
Curie","studentGroup":"10th grade"},{"id":4,"subject":"History","teacher":"I.
Jones","studentGroup":"10th grade"}]}'
```

在在 **application.properties** 文件中指定 的时间 被终止后，服务会返回类似以下示例的输出：

```
HTTP/1.1 200
Content-Type: application/json
...

{"timeslotList":...,"roomList":...,"lessonList":[{"id":1,"subject":"Math","teacher":"A.
Turing","studentGroup":"9th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":
{"name":"Room A"}},{"id":2,"subject":"Chemistry","teacher":"M. Curie","studentGroup":"9th
grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":
{"name":"Room A"}},{"id":3,"subject":"French","teacher":"M. Curie","studentGroup":"10th
grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":
{"name":"Room B"}},{"id":4,"subject":"History","teacher":"I. Jones","studentGroup":"10th
grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":
{"name":"Room B"}}],"score":"0hard/0soft"}
```

请注意，您的应用程序将全部四节课时分配给两个时间段之一，另一个是两个房间。另请注意，它符合所有硬约束。例如，**M. Curie** 的两节课时间不同。

3. 要在解决时间内查看 **OptaPlanner** 的作用，请查看服务器端的信息日志。以下是 **info** 日志输出示例：

```
... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
```

```
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score calculation speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score calculation speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed (28524/sec), phase total (2), environment mode (REPRODUCIBLE).
```

10.8. 测试应用程序

良好的应用程序包括测试覆盖。测试您时间项目中的限制和解决方法。

10.8.1. 测试 school timetable 约束

要测试隔离中可时间项目的每个约束，请在单元测试中使用 **ConstraintVerifier**。这会测试每个约束的基点，与其他测试隔离开来，这在添加新限制时降低了维护。

此测试会验证约束 **TimeTableConstraintProvider::roomConflict** 在同一房间，且两节课中的课程具有相同的 **timeslot**，具有匹配权重 **1**。因此，如果约束 **weight** 为 **10** 个位，它将分数减少 **-10hard**。

流程

创建 **src/test/java/org/acme/optaplanner/solver/TimeTableConstraintProviderTest.java** 类：

```
package org.acme.optaplanner.solver;

import java.time.DayOfWeek;
import java.time.LocalTime;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.Room;
import org.acme.optaplanner.domain.TimeTable;
import org.acme.optaplanner.domain.Timeslot;
import org.junit.jupiter.api.Test;
import org.optaplanner.test.api.score.stream.ConstraintVerifier;

@QuarkusTest
class TimeTableConstraintProviderTest {

    private static final Room ROOM = new Room("Room1");
    private static final Timeslot TIMESLOT1 = new Timeslot(DayOfWeek.MONDAY,
LocalTime.of(9,0), LocalTime.NOON);
    private static final Timeslot TIMESLOT2 = new Timeslot(DayOfWeek.TUESDAY,
LocalTime.of(9,0), LocalTime.NOON);
```

```
    @Inject
    ConstraintVerifier<TimeTableConstraintProvider, TimeTable> constraintVerifier;

    @Test
    void roomConflict() {
        Lesson firstLesson = new Lesson(1, "Subject1", "Teacher1", "Group1");
        Lesson conflictingLesson = new Lesson(2, "Subject2", "Teacher2", "Group2");
        Lesson nonConflictingLesson = new Lesson(3, "Subject3", "Teacher3", "Group3");

        firstLesson.setRoom(ROOM);
        firstLesson.setTimeslot(TIMESLOT1);

        conflictingLesson.setRoom(ROOM);
        conflictingLesson.setTimeslot(TIMESLOT1);

        nonConflictingLesson.setRoom(ROOM);
        nonConflictingLesson.setTimeslot(TIMESLOT2);

        constraintVerifier.verifyThat(TimeTableConstraintProvider::roomConflict)
                .given(firstLesson, conflictingLesson, nonConflictingLesson)
                .penalizesBy(1);
    }

}
```

请注意，**ConstraintVerifier** 在测试过程中如何忽略约束权重，即使这些约束 **weight** 在 **ConstraintProvider** 中是硬编码的。这是因为约束权重在进行生产环境前定期有所改变。这样，约束加权调整不会破坏单元测试。

10.8.2. 测试 school timetable solver

这个示例在 **Red Hat build of Quarkus** 上测试红帽构建的 **OptaPlanner school timetable** 项目。它使用 **JUnit** 测试来生成测试数据集，并将其发送到 **TimeTableController** 以解决。

流程

1. 使用以下内容创建 **src/test/java/com/example/rest/TimeTableResourceTest.java** 类：

```
package com.exmaple.optaplanner.rest;

import java.time.DayOfWeek;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.List;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import com.exmaple.optaplanner.domain.Room;
```

```
import com.exmaple.optaplanner.domain.Timeslot;
import com.exmaple.optaplanner.domain.Lesson;
import com.exmaple.optaplanner.domain.TimeTable;
import com.exmaple.optaplanner.rest.TimeTableResource;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@QuarkusTest
public class TimeTableResourceTest {

    @Inject
    TimeTableResource timeTableResource;

    @Test
    @Timeout(600_000)
    public void solve() {
        TimeTable problem = generateProblem();
        TimeTable solution = timeTableResource.solve(problem);
        assertFalse(solution.getLessonList().isEmpty());
        for (Lesson lesson : solution.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(solution.getScore().isFeasible());
    }

    private TimeTable generateProblem() {
        List<Timeslot> timeslotList = new ArrayList<>();
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30),
LocalTime.of(9, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>();
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
        lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
        lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
```

```
        lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
        return new TimeTable(timeslotList, roomList, lessonList);
    }

}
```

此测试会验证在解决后，所有课程都会分配给一个时间窗和房间。它还会验证它是否发现可行的解决方案（没有硬限制）。

2. 在 **src/main/resources/application.properties** 文件中添加 **test** 属性：

```
# The solver runs only for 5 seconds to avoid a HTTP timeout in this simple implementation.
# It's recommended to run for at least 5 minutes ("5m") otherwise.
quarkus.optaplanner.solver.termination.spent-limit=5s

# Effectively disable this termination in favor of the best-score-limit
%test.quarkus.optaplanner.solver.termination.spent-limit=1h
%test.quarkus.optaplanner.solver.termination.best-score-limit=0hard/*soft
```

通常，该解决方法在 **200** 毫秒内找到可行的解决方案。注意 **application.properties** 文件如何在测试过程中覆盖解决程序终止，以便在找到可行的解决方案 **(0hard/*soft)** 时立即终止。这可避免硬编码解决程序时间，因为单元测试可能会在任意硬件上运行。这种方法可确保测试用时足够长来查找可行的解决方案，即使在较慢的系统上也是如此。但是，即使在快速系统中，它也无法运行时间比严格要长的时间更长。

10.9. 日志记录

完成 **OptaPlanner school** 时间项目的红帽构建后，您可以使用日志信息帮助您微调 **ConstraintProvider** 中的限制。查看 **info** 日志文件中的分数计算速度，以评估对您的限制更改的影响。以 **debug** 模式运行应用程序，显示应用程序接受的每个步骤，或使用 **trace** 日志记录记录每一步和每次移动。

流程

1. 运行 **school timetable** 应用以获得固定时间，如五分钟。

2. 查看 日志文件中的 分数计算速度，如下例所示：

```
... Solving ended: ..., score calculation speed (29455/sec), ...
```

3. 更改约束，再次运行计划应用程序相同的时间，并查看 日志文件中 记录的分数计算速度。

4. 以 **debug** 模式运行应用程序，以记录应用程序所做的每个步骤：

   - 要从命令行运行调试模式，请使用 **-D** 系统属性。

   - 要永久启用调试模式，请在 **application.properties** 文件中添加以下行：

     ```
     quarkus.log.category."org.optaplanner".level=debug
     ```

     以下示例显示了在 **debug** 模式下 日志文件 的输出：

     ```
     ... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode (REPRODUCIBLE), random (JDK with seed 0).
     ...     CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15), picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
     ...     CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15), picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
     ...
     ```

5. 使用 **trace logging** 显示每个步骤以及每个步骤的每一个移动。

10.10. 将数据库与您的 QUARKUS OPTAPLANNER SCHOOL 时间相集成

创建 **Quarkus OptaPlanner school** 时间应用程序后，您可以将其与数据库集成，并创建一个基于 **Web** 的用户界面来显示时间稳定。

先决条件

- 您有一个 **Quarkus OptaPlanner school timetable** 应用程序。

流程

1. 使用 **Hibernate** 和 **Panache** 将 **Timeslot**、**Room** 和 **Lesson** 实例存储在数据库中。如需更多信息，请参阅使用 **Panache** 的简化 **Hibernate ORM**。

2. 通过 **REST** 公开实例。如需更多信息，请参阅 编写 **JSON REST** 服务。

3. 更新 **TimeTable Resource** 类，以读取和写入一个事务中的可时间实例：

```
package org.acme.optaplanner.rest;

import javax.inject.Inject;
import javax.transaction.Transactional;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;

import io.quarkus.panache.common.Sort;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.Room;
import org.acme.optaplanner.domain.TimeTable;
import org.acme.optaplanner.domain.Timeslot;
import org.optaplanner.core.api.score.ScoreManager;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.solver.SolverManager;
import org.optaplanner.core.api.solver.SolverStatus;

@Path("/timeTable")
public class TimeTableResource {

    public static final Long SINGLETON_TIME_TABLE_ID = 1L;

    @Inject
    SolverManager<TimeTable, Long> solverManager;
    @Inject
    ScoreManager<TimeTable, HardSoftScore> scoreManager;

    // To try, open http://localhost:8080/timeTable
    @GET
    public TimeTable getTimeTable() {
        // Get the solver status before loading the solution
        // to avoid the race condition that the solver terminates between them
        SolverStatus solverStatus = getSolverStatus();
        TimeTable solution = findById(SINGLETON_TIME_TABLE_ID);
        scoreManager.updateScore(solution); // Sets the score
        solution.setSolverStatus(solverStatus);
        return solution;
    }

    @POST
    @Path("/solve")
    public void solve() {
        solverManager.solveAndListen(SINGLETON_TIME_TABLE_ID,
                this::findById,
                this::save);
    }
```

```
    public SolverStatus getSolverStatus() {
        return solverManager.getSolverStatus(SINGLETON_TIME_TABLE_ID);
    }

    @POST
    @Path("/stopSolving")
    public void stopSolving() {
        solverManager.terminateEarly(SINGLETON_TIME_TABLE_ID);
    }

    @Transactional
    protected TimeTable findById(Long id) {
        if (!SINGLETON_TIME_TABLE_ID.equals(id)) {
            throw new IllegalStateException("There is no timeTable with id (" + id + ").");
        }
        // Occurs in a single transaction, so each initialized lesson references the same
timeslot/room instance
        // that is contained by the timeTable's timeslotList/roomList.
        return new TimeTable(
                Timeslot.listAll(Sort.by("dayOfWeek").and("startTime").and("endTime").and("id")),
                Room.listAll(Sort.by("name").and("id")),
                Lesson.listAll(Sort.by("subject").and("teacher").and("studentGroup").and("id")));
    }

    @Transactional
    protected void save(TimeTable timeTable) {
        for (Lesson lesson : timeTable.getLessonList()) {
            // TODO this is awfully naive: optimistic locking causes issues if called by the
SolverManager
            Lesson attachedLesson = Lesson.findById(lesson.getId());
            attachedLesson.setTimeslot(lesson.getTimeslot());
            attachedLesson.setRoom(lesson.getRoom());
        }
    }

}
```

这个示例包含一个可 时间 实例。但是，您可以并行为多个学校启用多租户和处理 定时 实例。

getTimeTable（） 方法返回来自数据库的最新时间表。它使用自动注入的 **ScoreManager** 方法来计算该时间表的分数，并使其可用于 **UI**。

solve（） 方法启动一个作业，以解决当前可时间表，并将时间窗和房分配存储在数据库中。它使用 **SolverManager.solveAndListen**（） 方法侦听中间最佳解决方案，并相应地更新数据库。**UI** 使用它来显示后端仍在解决过程中的进度。

4.

更新 **TimeTableResourceTest** 类，以反映 **solve**（） 方法立即返回并轮询最新的解决方案，直到解决解决为止：

```
package org.acme.optaplanner.rest;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.TimeTable;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.optaplanner.core.api.solver.SolverStatus;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@QuarkusTest
public class TimeTableResourceTest {

    @Inject
    TimeTableResource timeTableResource;

    @Test
    @Timeout(600_000)
    public void solveDemoDataUntilFeasible() throws InterruptedException {
        timeTableResource.solve();
        TimeTable timeTable = timeTableResource.getTimeTable();
        while (timeTable.getSolverStatus() != SolverStatus.NOT_SOLVING) {
            // Quick polling (not a Test Thread Sleep anti-pattern)
            // Test is still fast on fast machines and doesn't randomly fail on slow machines.
            Thread.sleep(20L);
            timeTable = timeTableResource.getTimeTable();
        }
        assertFalse(timeTable.getLessonList().isEmpty());
        for (Lesson lesson : timeTable.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(timeTable.getScore().isFeasible());
    }

}
```

5. 在这些 **REST** 方法上构建 **Web UI**，以提供时间表的视觉表示。

6. 查看 *快速入门源代码*。

10.11. 使用 MICROMETER 和 PROMETHEUS 来监控 SCHOOL TIMETABLE OPTAPLANNER QUARKUS 应用程序

OptaPlanner 通过 Micrometer 公开指标数据，这是 Java 应用程序的指标检测库。您可以将 Micrometer 与 Prometheus 搭配使用以监控 school timetable 应用程序中 OptaPlanner solver。

先决条件

- *您已创建了 Quarkus OptaPlanner school timetable 应用程序。*
- *已安装 Prometheus。有关安装 Prometheus 的详情，请查看 Prometheus 网站。*

流程

1. *将 Micrometer Prometheus 依赖项添加到 school timetable pom.xml 文件中：*

```
<dependency>
 <groupId>io.quarkus</groupId>
 <artifactId>quarkus-micrometer-registry-prometheus</artifactId>
</dependency>
```

2. *启动 school timetable 应用程序：*

```
mvn compile quarkus:dev
```

3. *在 Web 浏览器中打开 http://localhost:8080/q/metric。*

第 11 章 RED HAT BUILD OF OPTAPLANNER ON RED HAT BUILD OF QUARKUS: A VACCINATION APPOINTMENT 调度程序快速启动指南

您可以使用 **OptaPlanner vaccination appointment** 调度程序快速启动，以开发一个有效和公平的 **vaccination** 调度。**vaccination appointment** 调度程序使用 **artificial** 智能(AI)来根据多个限制和优先级排列人并分配时间插槽。

先决条件

- 安装了 **OpenJDK 11** 或更高版本。红帽构建的 **Open JDK** 可从红帽客户门户网站的 **Software Downloads** 页面获取（需要登录）。
- 已安装 **Apache Maven 3.6** 或更高版本。**Maven** 可以从 **Apache Maven Project** 网站获得。
- 使用 **IntelliJ IDEA**、**VS Code**、**Eclipse** 或 **NetBeans** 等 **IDE**。
- 您已创建了 **Quakus OptaPlanner** 项目，如 **第 6 章 OptaPlanner 和 Quarkus 入门** 所述。

11.1. OPTAPLANNER VACCINATION APPOINTMENT 调度程序如何工作

调度 **appointments** 的方法主要有两种。系统可以允许个人选择一个选项插槽（用户选择）或者系统分配一个插槽并告知个人何时和位置参加（系统自动自动分配分配）。**OptaPlanner vaccination appointment** 调度程序使用 **system-automatically-assigns** 方法。通过 **OptaPlanner vaccination appointment** 调度程序，您可以创建一个应用程序，其中人们为系统提供信息，系统会分配一个 **appointment**。

这个方法的特性：

- **appointment** 插槽根据优先级分配。
- 系统根据预配置的规划限制分配最佳的 **appointment** 时间和位置。
- 该系统不再被大量用户对受制约数量有限。

这个方法可以通过使用计划限制为每个人创建分数来解决 **vaccinating** 的问题。个人的分数决定何时获得指示。个人得分越大，他们收到之前的发言的机会越大。

11.1.1. OptaPlanner vaccination appointment 调度程序限制

OptaPlanner vaccination appointment 调度程序限制可以是 **hard**、**medium** 或 **soft**：

- 硬约束不能被破坏。如果有任何硬约束被破坏，则计划不可行，且无法执行：
  - 容量：在任何位置，不要随时通过手册 **vaccine** 容量。
  - **Vaccine** 最大年龄：如果 **vaccine** 具有最长期限，则不像首次进行的 **vaccine** 最长期限更久时对其进行管理，而不是 **vaccine** 最长期限。确保为人赋予一个适合其年龄的 **vaccine** 类型。例如，不要为 **vaccine** 分配 **75** 年旧个人，其最长期限限制为 **65** 年。
  - 所需的 **vaccine type**：使用所需的 **vaccine** 类型。例如，**vaccine** 的第二代的 **vaccine** 类型必须与第一个 **dose** 相同。
  - 就绪日期：管理指定日期或之后的 **vaccine**。例如，如果某个人收到第二个操作，则在建议尽快进行 **vaccine** 类型之前，不要对其进行管理，例如在第一个操作后 **26** 天。
  - 到期日期：管理指定日期之前或之前的 **vaccine**。例如，如果某个人收到第二个操作，请在推荐的 **vaccination** 最终因特定 **vaccine** 日期前对其进行管理，例如在第一个操作后的三个月。
  - 限制最大差距离：将每个人分配给最接近其一组 **vaccination** 中心之一。这通常是三个数据中心中的一个。这种限制的计算时间是差差时间，而不是距离距离，因此那些在维系地区持有的人通常比农区的差时限度要低。
- **Medium** 约束决定，当没有足够容量为每个人分配点时，他们不会得到指出。这被称为约束规划：
  - **schedule second dose vaccinations**：不要使任何第二个确实 **vaccination** 没有被分配，除非理想的日期不在计划窗口之外。

- *根据优先级评级计划人员：每个人具有优先级评级。这通常是其年龄，但如果它们来说，这要高得多，例如一个健康护程序。只有具有最低优先级的人们才是未分配的。在下次运行时将考虑它们。这个约束比上一个约束的软限制，因为第二个操作总是优先级在优先级评级上。*

- *软限制不应该有问题：*
  - *首选 **vaccination** 中心：如果一个人拥有首选的 **vaccination** 中心，请在该中心给出一个选择点。*
  - *距离：让人必须转入其分配的 **vaccination** 中心的距离。*
  - *理想日期：管理 **vaccine on** 或接近指定日期。例如，如果某个人收到第二个操作，则根据特定 **vaccine** 的基本日期对其进行管理，例如在第 **28** 天之后。这个约束比距离约束的软，以避免在国家间发送一半，使其更接近其理想的日期。*
  - *优先级评级：在规划窗口之前调度优先级较高的人。这个约束比距离约束的软，以避免在国家之间发送一半。这个约束也比理想日期约束的软，因为第二个操作的优先级高于优先级评级。*

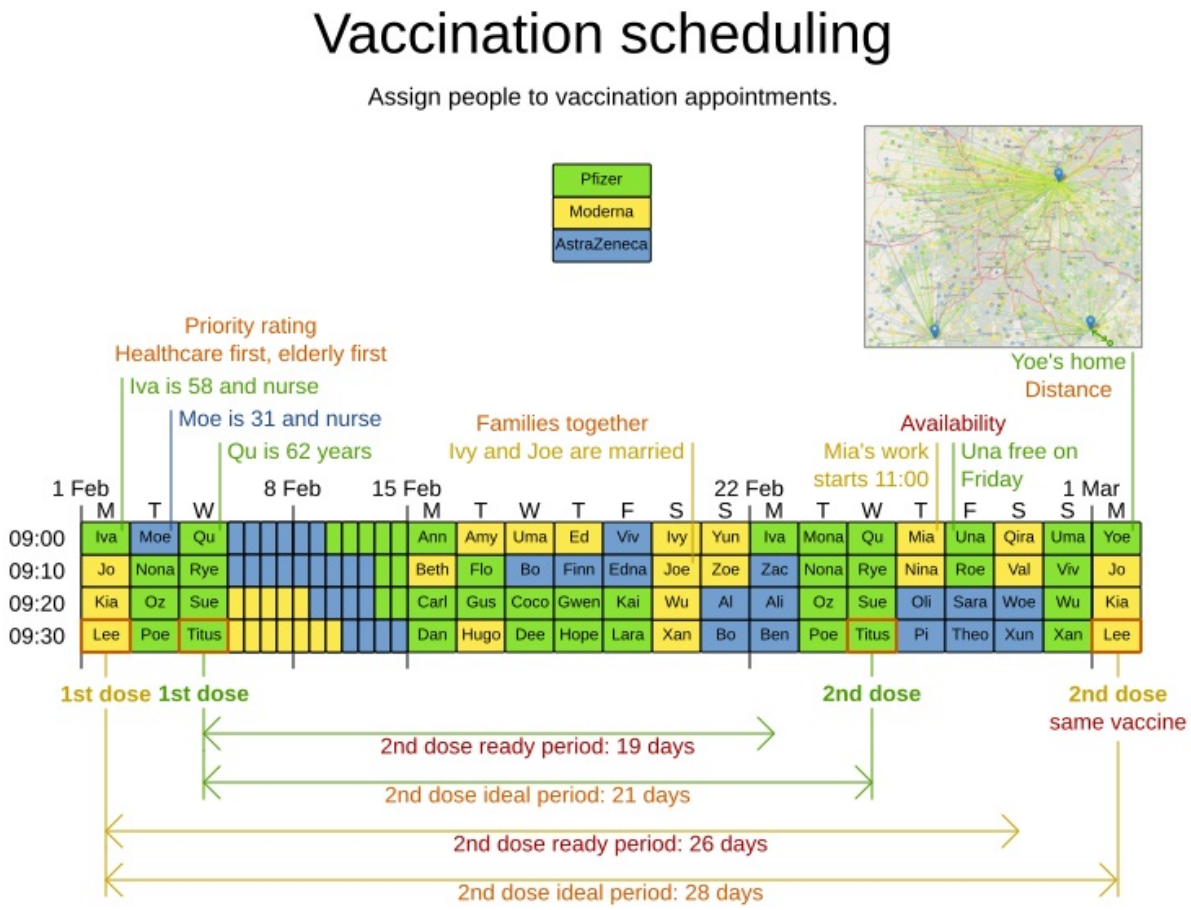
*硬约束的加法是相对于其他硬约束的权重。软限制会高于其他软限制。但是，硬约束总是优先于中和软限制。如果硬约束被破坏，则计划不可行。但是，如果没有硬的限制，则考虑软限制和中等限制来确定优先级。因为只有更多的人会超过可用代表的插槽，所以您必须优先选择。第二个确实需要首先被分配，以避免在以后创建不堪重负您的系统的积压。之后，会根据优先级的等级分配用户。每个人都从优先级评级开始，是其年龄。这样做会优先考虑旧人，而非年轻的人。之后，处于特定优先级组的人接收，例如几百个额外的点。这与其组的优先级不同。例如，**nurses** 可能会接收额外的 **1000** 点。这样，旧的 **nurses** 的优先级高于年轻的 **nurses**，而年轻的 **nurses** 则高于非 **nurses** 的人。下表演示了这个概念：*

*表 **11.1.** 优先级等级表*

age	作业	优先级评级
60	nurse	1060
33	nurse	1033
71	弃用	71
52	办公室 worker	52

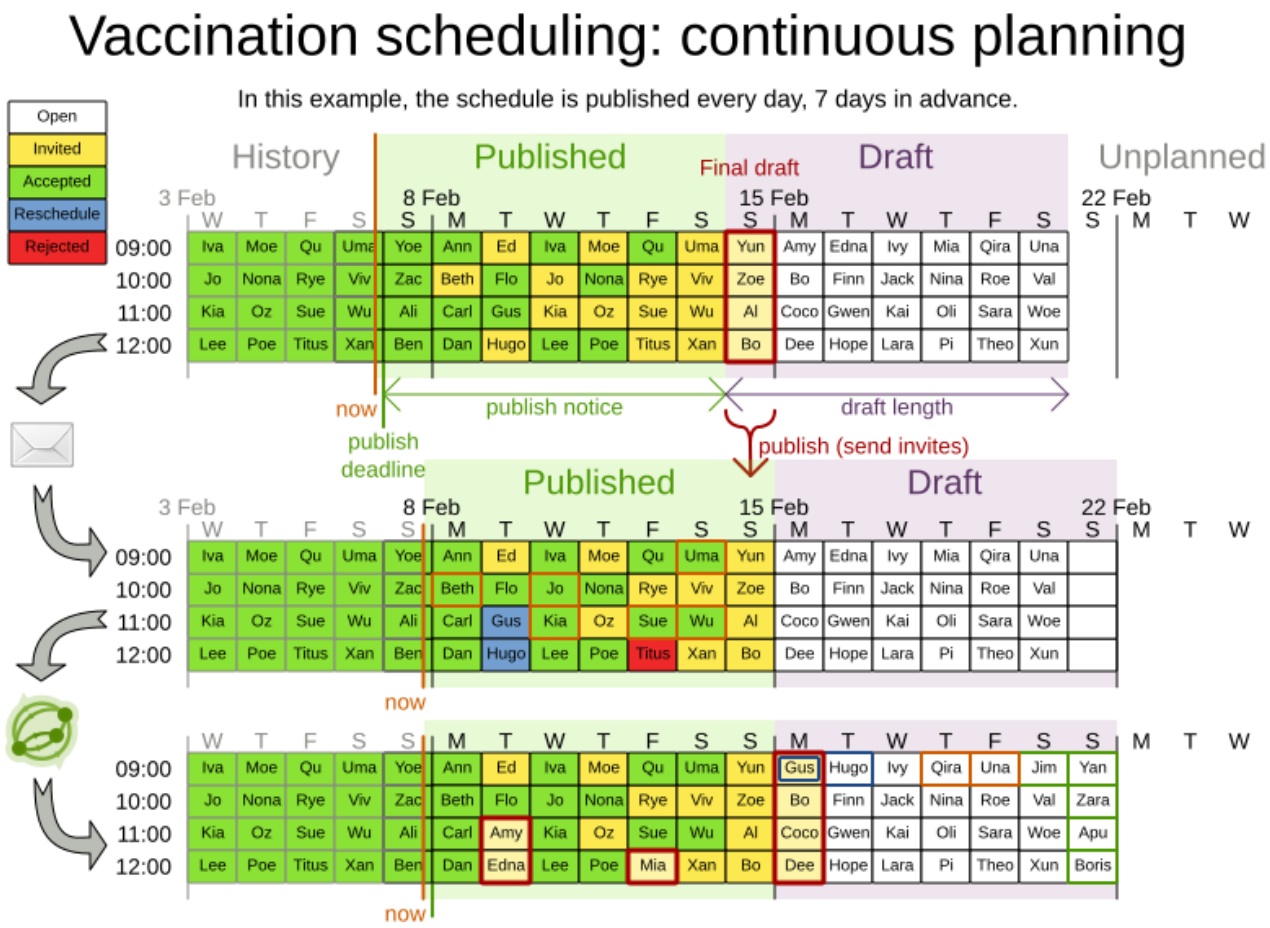
11.1.2. OptaPlanner solver

OptaPlanner 的核心是解决方法，引擎用于提取问题数据集并覆盖规划限制和配置。问题数据集包含人们、**vaccines** 和 **vaccination** 中心的所有信息。解决器通过各种数据组合进行工作，最终决定一个经过优化的指示计划，与分配给特定中心的 **vaccination appointments** 的人员进行优化。以下示例显示一个临时解决方案创建的调度：



11.1.3. 持续规划

持续计划是同时管理一个或多个即将举办的规划期的技术，并经常重复该过程每月、每周、每天、每小时或更频繁地重复该过程。计划窗口按指定间隔递增。下图显示了两周计划窗口，每天更新：



两周规划窗口被分为一半。第一周处于已发布的状态，第二周处于草案状态。人们被分配到计划窗口发布和部分的发言。但是，只有计划窗口发布的一部分的用户才会获得其发言通知。其他的 ***appointments*** *仍可在下一次运行时轻松更改。这样做可让* ***OptaPlanner*** *灵活地在再次运行问题时更改草案部分中的提示（如果需要）。例如，如果一个需要第二个操作的人准备了星期一，* ***OptaPlanner*** *认为是星期三的准备日期，如果您可以证明* ***OptaPlanner***，*则可以为星期一给出一个草案。*

您可以确定规划窗口的大小，但只了解问题空间的大小。问题空间就是创建计划的所有元素。您提前计划的时间越大，问题就越大。

11.1.4. 固定计划实体

如果您每天持续规划，在两个周的时间内将出现指出，该周期已分配给个人。要确保 ***appointments*** *没有双引号，* ***OptaPlanner*** *将现有的* ***appointments*** *标记为固定值分配。固定(* ***pinning)*** *用于定位一个或多个特定的分配，并强制* ***OptaPlanner*** *围绕这些固定分配进行调度。固定规划实体（如* ***appointment***）*在解决过程中不会改变。*

某个实体是固定还是不由 ***appointment*** *状态决定。* ***appointment*** *可以具有五个状态：* ***Open***、***Invited***、***Accepted***、***Rejected*** *或* ***Rescheduled***。

注意

您实际上不会在快速启动演示代码中直接看到这些状态，因为 **OptaPlanner** 引擎只对 **appointment** 是否固定。

您需要能够规划已调度的 **appointments**。固定具有 **Invited** 或 **Accepted** 状态的 **appointment**。没有固定状态，带有 **Open**、**Reschedule** 和 **Rejected** 状态没有固定状态，并可用于调度。

在这个示例中，当解决器在已发布和草案范围内搜索整个两周计划窗口时。除非计划的输入数据外，解决者还考虑了任何未固定实体、**Open**、**Reschedule** 或 **Rejected** 状态的任何未固定实体。如果每天运行解决方法，您会在运行解决者之前看到在计划中添加的新日期。

请注意，新天的 **appointments** 已被分配，在计划窗口的草案中调度了 **Amy** 和 **Edna**。这是因为 **Gus** 和 **Hugo** 请求重新调度。这不会造成任何混淆，因为 **Amy** 和 **Edna** 不会被通知其草案日期。现在，由于他们在规划窗口的已发布部分有指出，因此将收到通知并要求接受或拒绝其提示，并且现在固定了。

11.2. 下载并运行 OPTAPLANNER VACCINATION APPOINTMENT 调度程序

下载 **OptaPlanner vaccination appointment** 调度程序快速启动存档，以 **Quarkus** 开发模式启动存档，并在浏览器中查看应用程序。**Quarkus** 开发模式允许您在运行过程中进行更改和更新应用程序。

流程

1. 进入红帽客户门户网站中的 **Software Downloads** 页面（需要登录），然后从下拉列表中选择产品和版本：
   - 产品： 流程自动化管理器
   - 版本： **7.13.4**
2. 下载 **Red Hat Process Automation Manager 7.13.4 Kogito 和 OptaPlanner 8 Decision Services Quickstarts (rhpam-7.13.4-kogito-and-optaplanner-quickstarts.zip)**。
3. 提取 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts.zip** 文件。

4. 导航到 **optaplanner-quickstarts-8.13.0.Final-redhat-00013** 目录。

5. 导航到 **optaplanner-quickstarts-8.13.0.Final-redhat-00013/use-cases/vaccination-scheduling** 目录。

6. 输入以下命令在开发模式中启动 **OptaPlanner vaccination appointment** 调度程序：

```
$ mvn quarkus:dev
```

7. 要查看 **OptaPlanner vaccination appointment** 调度程序，在 **web** 浏览器中打开以下 **URL**。

```
http://localhost:8080/
```

8. 要运行 **OptaPlanner vaccination appointment** 调度程序，点 **Solve**。

9. 对源代码进行更改，然后按 **F5** 键刷新浏览器。请注意，您所做的更改现已可用。

11.3. 软件包并运行 OPTAPLANNER VACCINATION APPOINTMENT 调度程序

当您完成了对 **OptaPlanner vaccination appointment** 调度程序的开发工作后，以 **quarkus:dev** 模式运行应用程序，作为传统的 **jar** 文件运行。

先决条件

- 您已下载了 **OptaPlanner vaccination appointment** 调度程序快速启动。更多信息请参阅 *第 11.2 节 “下载并运行 OptaPlanner vaccination appointment 调度程序”*。

流程

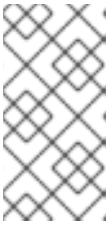
1. 进入 **/use-cases/vaccination-scheduling** 目录。

2. 要编译 **OptaPlanner vaccination appointment** 调度程序，请输入以下命令：

```
$ mvn package
```

3. 要运行编译的 **OptaPlanner vaccination appointment** 调度程序，请输入以下命令：

```
$ java -jar ./target/*-runner.jar
```



注意

要在端口 **8081** 上运行应用程序，请在前面的命令中添加 **-Dquarkus.http.port=8081**。

4. 要启动 **OptaPlanner vaccination appointment** 调度程序，在 **web** 浏览器中打开以下 **URL**。

```
http://localhost:8080/
```

11.4. 运行 OPTAPLANNER VACCINATION APPOINTMENT 调度程序，作为原生可执行文件

要利用 **Quarkus** 提供的小型内存占用和访问速度，在 **Quarkus** 原生模式下编译 **OptaPlanner vaccination appointment** 调度程序。

流程

1. 安装 **GraalVM** 和 **native-image** 工具。如需更多信息，请参阅 **Quarkus** 网站上 *配置 GraalVMI*。

2. 进入 **/use-cases/vaccination-scheduling** 目录。

3. 要原生编译 **OptaPlanner vaccination appointment** 调度程序，请输入以下命令：

```
$ mvn package -Dnative -DskipTests
```

4. 要运行原生可执行文件，请输入以下命令：

```
$ ./target/*-runner
```

5. 要启动 **OptaPlanner vaccination appointment** 调度程序，在 **web** 浏览器中打开以下 **URL**。

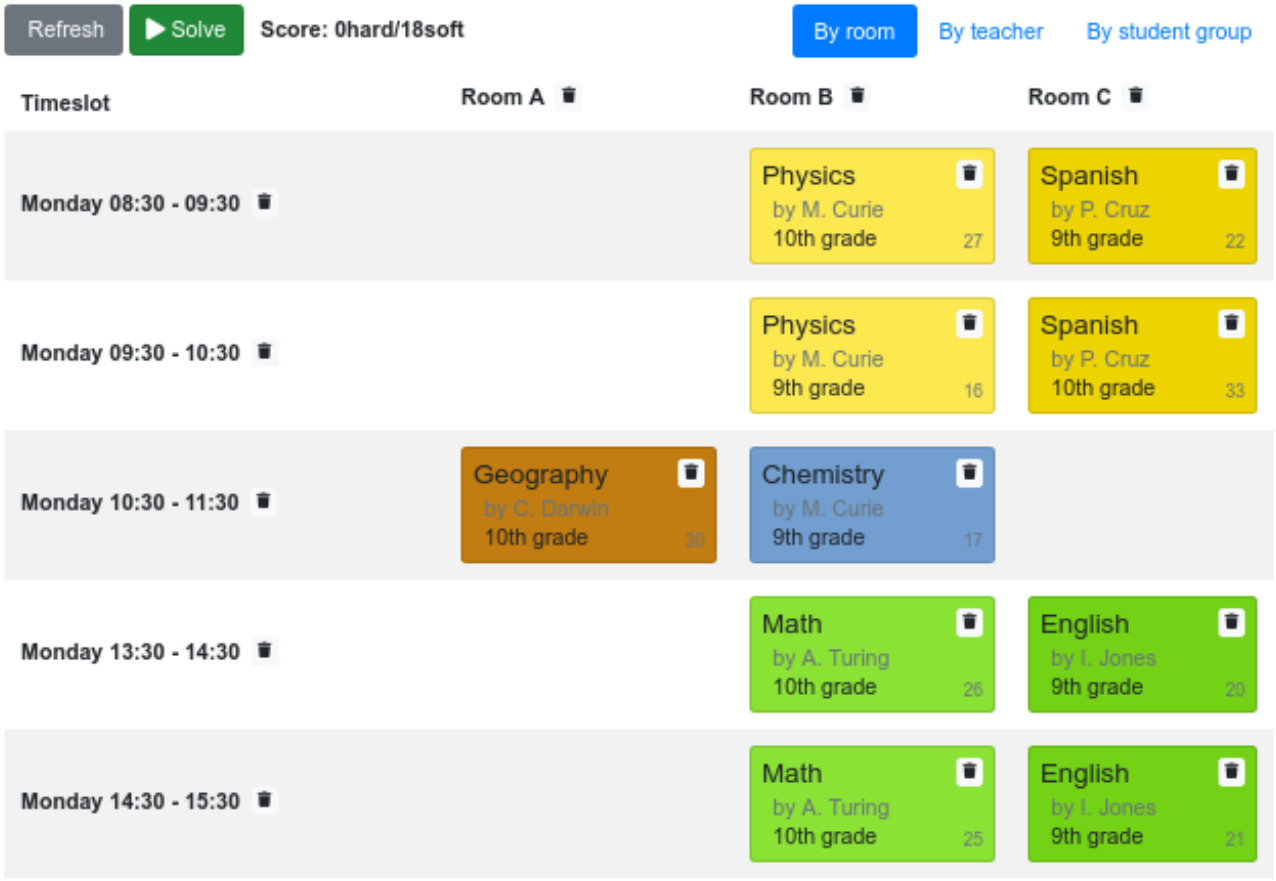
```
http://localhost:8080/
```

11.5. 其他资源

- **Vaccination appointment** 调度视频

第 12 章 RED HAT BUILD OF OPTAPLANNER ON SPRING BOOT: A SCHOOL TIMETABLE QUICK START GUIDE

本指南指导您了解使用 **OptaPlanner** 的约束解决 **artificial** 智能**(AI)**的 **Spring Boot** 应用程序的过程。您将构建一个 **REST** 应用程序，针对学生和老师优化院校时间。



您的服务会自动将 **Lesson** 实例分配给 **Timeslot** 和 **Room** 实例，方法是使用 **AI** 遵循以下硬和软 调度限制 ：

- 个空间最多可以同时有一处。
- 老师可以同时在上课时进行教学。
- 学员最多可以同时参加。
- 教授者更倾向于在单一房间学习。

- 指导者更倾向于在课间学习顺序课时和不类差距。

数学会上讲，该学校的时间范围是一个 **NP-hard** 问题。这意味着很难扩展。简单地说，通过带有 **brute** 强制的可能组合将花费数百万年时间，而在超级计算机中，甚至在超级计算机中也要花费数百万年时间。幸运的是，**AI** 约束解决诸如 **OptaPlanner** 等高级算法具有在合理的时间内提供最接近的解决方案。被视为有合理的时间，它取决于问题的目标。

先决条件

- 安装了 **OpenJDK 11** 或更高版本。红帽构建的 **Open JDK** 可从红帽客户门户网站的 **Software Downloads** 页面获取（需要登录）。
- 已安装 **Apache Maven 3.6** 或更高版本。**Maven** 可以从 **Apache Maven Project** 网站获得。
- 使用 **IntelliJ IDEA**、**VS Code**、**Eclipse** 或 **NetBeans** 等 **IDE**。

12.1. 下载并构建 SPRING BOOT SCHOOL 快速入门

要查看红帽使用 **Spring Boot** 产品构建 **OptaPlanner** 的 **school** 时间项目的已完成示例，请从红帽客户门户网站下载入门应用程序。

流程

1. 进入红帽客户门户网站中的 **Software Downloads** 页面（需要登录），然后从下拉列表中选择产品和版本：
   - 产品： 流程自动化管理器
   - 版本： **7.13.4**
2. 下载 **Red Hat Process Automation Manager 7.13.4 Kogito 和 OptaPlanner 8 Decision Services Quickstarts (rhpam-7.13.4-kogito-and-optaplanner-quickstarts.zip)**。
3. 提取 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts.zip** 文件。

4. 下载 **Red Hat Process Automation Manager 7.13.4 Kogito 和 OptaPlanner 8 Decision Services Maven Repositroy (rhpam-7.13.4-kogito-maven-repository.zip)**。

5. 提取 **rhpam-7.13.4-kogito-maven-repository.zip** 文件。

6. 将 **rhpam-7.13.4-kogito-maven-repository/maven-repository** 子目录的内容复制到 **~/.m2/repository** 目录中。

7. 导航到 **optaplanner-quickstarts-8.13.0.Final-redhat-00013/ Technology/java-spring-boot** 目录。

8. 输入以下命令来构建 **Spring Boot school timetabling** 项目：

```
mvn clean install -DskipTests
```

9. 要构建 **Spring Boot school timetabling** 项目，请输入以下命令：

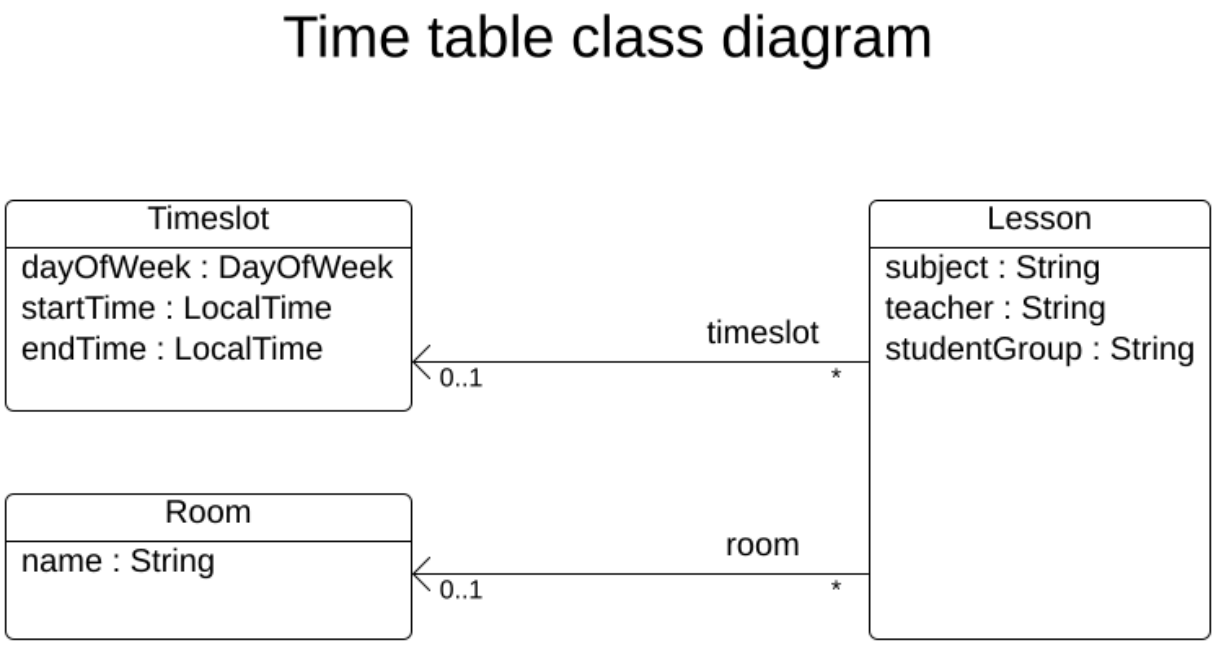
```
mvn spring-boot:run -DskipTests
```

10. 要查看项目，在网页浏览器中输入以下 **URL**：

```
http://localhost:8080/
```

12.2. 对域对象建模

红帽构建的 **OptaPlanner** 时间表项目的目标是将每节分配给一个时间段和一个房间。为此，请添加三个类：**Timeslot**、**Lesson** 和 **Room**，如下图所示：



Timeslot

***Timeslot** 类表示在未教授课程的时间间隔，例如： **Monday 10:30 - 11:30** 或 **Tuesday 13:30 - 14:30 - 14:30**。在这个示例中，所有时间插槽都有相同的持续时间，且在 **lunch** 或其他中断期间没有时间插槽。*

*时间段没有日期，因为高校的计划只每周重复。无法进行持续 规划。次性被称为 问题事实，因为在解决过程中不会有定时实例变化。此类类不需要任何特定于 **OptaPlanner** 的注解。*

room

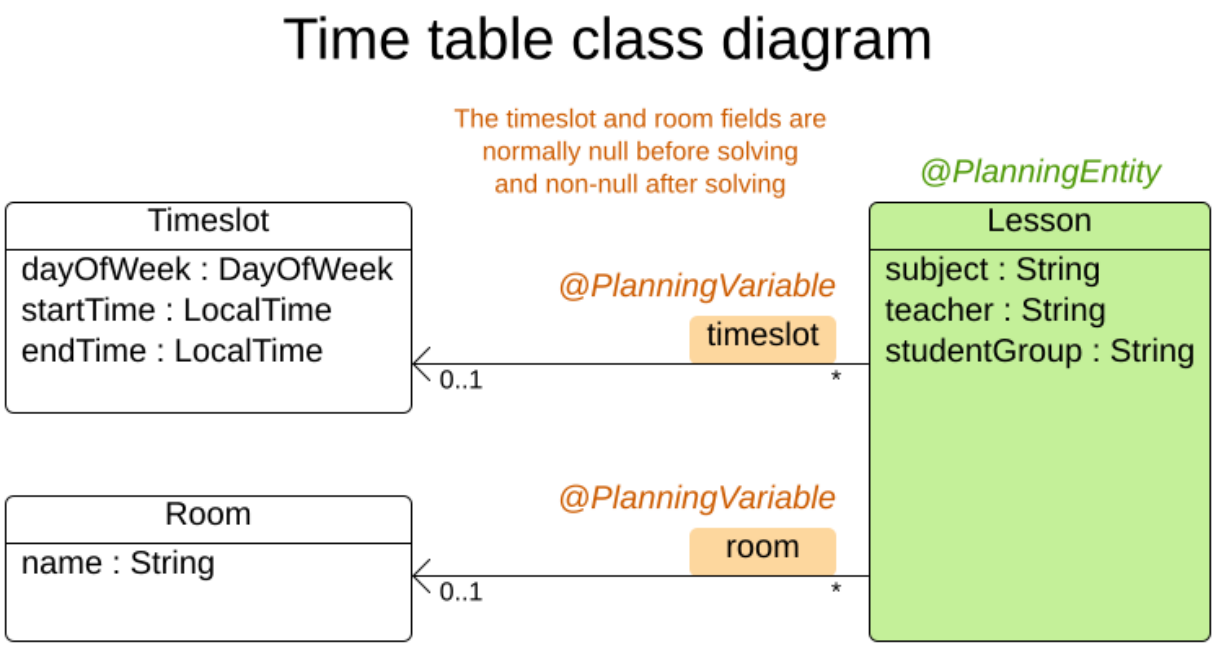
***Room** 类代表了课程讲授的方式，例如 **Room A** 或 **Room B**。在本例中，所有空间都没有容量限制，它们可容纳所有课时。*

*房间 实例在解决过程中不会改变，因此 **Room** 也是 问题。*

lesson

*在课时，由课课课表示，向班级教授一个主题，例如： **A.Turing for a 9th grade** 或 **Chemistry by M.Curie** 为 **10th grade**。如果一个主题每周都会为同一学员组讲授多次，则有多个小的、只能由 **id** 区分的实例。例如，第 **9** 分之时，每周有 **6** 个数学。*

*在寻求过程中，**OptaPlanner** 改变了 **lesson** 类的 **timelot** 和 **room** 字段，从而将每个较小的时间分配给一个时间段和一个房间。 因为 **OptaPlanner** 更改了这些字段，所以 **lesson** 是一个 规划实体 ：*



*上图中的大多数字段包含输入数据，但 **orange** 字段除外。在输入数据中 未分配的 时间和房间 字段 **(null)**，并在输出数据中分配（而不是 **null**）。**OptaPlanner** 在解决过程中更改这些字段。这些字段称为规划变量。为了 **OptaPlanner** 可以识别它们，**timelot** 和 **room** 字段都需要 **@PlanningVariable** 注解。它们包含类**(Lesson)**需要 **@PlanningEntity** 注释。*

流程

1. *创建 **src/main/java/com/example/domain/Timeslot.java** 类：*

```
package com.example.domain;

import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;

    private Timeslot() {
    }

    public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
        this.dayOfWeek = dayOfWeek;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    @Override
```

```
    public String toString() {
        return dayOfWeek + " " + startTime.toString();
    }

    // ********************************
    // Getters and setters
    // ********************************

    public DayOfWeek getDayOfWeek() {
        return dayOfWeek;
    }

    public LocalTime getStartTime() {
        return startTime;
    }

    public LocalTime getEndTime() {
        return endTime;
    }

}
```

请注意，**toString**（）方法保留了输出短片，以便可以更轻松地读取 **OptaPlanner** 的 **DEBUG** 或 **abrt** 日志，如稍后所示。

2. 创建 **src/main/java/com/example/domain/Room.java** 类：

```
package com.example.domain;

public class Room {

    private String name;

    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public String getName() {
        return name;
```

```
    }

}
```

3. 创建 **src/main/java/com/example/domain/Lesson.java** 类：

```
package com.example.domain;

import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
    }

    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // ********************************
    // Getters and setters
    // ********************************

    public Long getId() {
        return id;
    }

    public String getSubject() {
        return subject;
    }

    public String getTeacher() {
```

```
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {
        return timeslot;
    }

    public void setTimeslot(Timeslot timeslot) {
        this.timeslot = timeslot;
    }

    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
        this.room = room;
    }

}
```

Lesson 类具有 **@PlanningEntity** 注释，因此 **OptaPlanner** 知道这个类在解决过程中发生了变化，因为它包含一个或多个计划变量。

time lot 字段具有 **@PlanningVariable** 注释，因此 **OptaPlanner** 知道它可以更改其值。为了查找要分配给此字段的潜在的 **Timeslot** 实例，**OptaPlanner** 使用 **valueRangeProviderRefs** 属性连接到一个提供 **List<Timeslot>** 的值供应商。有关值范围供应商的信息，请参阅 第 **12.4** 节 “在规划解决方案中收集域对象”。

出于同样原因，**room** 字段还具有 **@PlanningVariable** 注释。

12.3. 定义限制并计算分数

当解决问题时，分数 代表特定解决方案的质量。得分越高。红帽构建的 **OptaPlanner** 寻找最佳解决方案，这是可用时间获得最高分数的解决方案。这可能是 最佳解决方案。

因为可时间的示例用例具有硬和软约束，因此请使用 **HardSoftScore** 类来代表分数：

- 硬限制不得被破坏。例如： 一个房间可以有以上时间。

- *软限制不应该被破坏。例如：更喜欢在单个房里的教学方式。*

硬约束的加法是相对于其他硬约束的权重。软限制会高于其他软限制。硬限制始终大于软限制，无论其对应的权重如何。

*要计算分数，您可以实施 **EasyScoreCalculator** 类：*

```
public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

   @Override
   public HardSoftScore calculateScore(TimeTable timeTable) {
      List<Lesson> lessonList = timeTable.getLessonList();
      int hardScore = 0;
      for (Lesson a : lessonList) {
         for (Lesson b : lessonList) {
            if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
                  && a.getId() < b.getId()) {
               // A room can accommodate at most one lesson at the same time.
               if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                  hardScore--;
               }
               // A teacher can teach at most one lesson at the same time.
               if (a.getTeacher().equals(b.getTeacher())) {
                  hardScore--;
               }
               // A student can attend at most one lesson at the same time.
               if (a.getStudentGroup().equals(b.getStudentGroup())) {
                  hardScore--;
               }
            }
         }
      }
      int softScore = 0;
      // Soft constraints are only implemented in the "complete" implementation
      return HardSoftScore.of(hardScore, softScore);
   }

}
```

不幸的是，这个解决方案无法进行很好的扩展，因为它是非增量：每次分配给不同的时间段或房间，都会重新评估所有课程来计算新分数。

*更好的解决方法是创建一个 **src/main/java/com/example/solver/TimeTableConstraintProvider.java** 类来执行增量分数计算。此类使用 **OptaPlanner** 的 **ConstraintStream API**，它由 **Java 8 Streams** 和 **SQL** 产生。**ConstraintProvider** 缩放比 **EasyScoreCalculator** 更好的度量：**O(n)**而不是 **O(n)**。*

流程

*创建以下 **src/main/java/com/example/solver/TimeTableConstraintProvider.java** 类：*

```
package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
                // Hard constraints
                roomConflict(constraintFactory),
                teacherConflict(constraintFactory),
                studentGroupConflict(constraintFactory),
                // Soft constraints are only implemented in the "complete" implementation
        };
    }

    private Constraint roomConflict(ConstraintFactory constraintFactory) {
        // A room can accommodate at most one lesson at the same time.

        // Select a lesson ...
        return constraintFactory.from(Lesson.class)
                // ... and pair it with another lesson ...
                .join(Lesson.class,
                        // ... in the same timeslot ...
                        Joiners.equal(Lesson::getTimeslot),
                        // ... in the same room ...
                        Joiners.equal(Lesson::getRoom),
                        // ... and the pair is unique (different id, no reverse pairs)
                        Joiners.lessThan(Lesson::getId))
                // then penalize each pair with a hard weight.
                .penalize("Room conflict", HardSoftScore.ONE_HARD);
    }

    private Constraint teacherConflict(ConstraintFactory constraintFactory) {
        // A teacher can teach at most one lesson at the same time.
        return constraintFactory.from(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getTeacher),
                        Joiners.lessThan(Lesson::getId))
                .penalize("Teacher conflict", HardSoftScore.ONE_HARD);
    }

    private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
```

```
        // A student can attend at most one lesson at the same time.
        return constraintFactory.from(Lesson.class)
                .join(Lesson.class,
                        Joiners.equal(Lesson::getTimeslot),
                        Joiners.equal(Lesson::getStudentGroup),
                        Joiners.lessThan(Lesson::getId))
                .penalize("Student group conflict", HardSoftScore.ONE_HARD);
    }

}
```

12.4. 在规划解决方案中收集域对象

TimeTable 实例嵌套了单个 **dataset** 的所有 **Timeslot**、**Room** 和 **Lesson** 实例。另外，由于它包含了 **allon**，每个都具有特定规划变量状态，所以它是一个 规划解决方案，并有分数：

- 如果课程仍然没有被签名，那么它是一个 未初始化 的解决方案，例如分数为 **-4init/0hard/0soft** 的解决方案。
- 如果损坏硬限制，那么它是一种 不可预见的 解决方案，例如分数为 **-2hard/-3soft** 的解决方案。
- 如果遵循所有硬约束，那么它是一个可行的解决方案，例如，分数为 **0hard/-7soft** 的解决方案。

TimeTable 类有一个 **@PlanningSolution** 注释，因此红帽构建的 **OptaPlanner** 知道该类包含所有输入和输出数据。

具体来说，这个类是问题的输入：

- 带有所有时间窗的 **timeslotList** 字段
  - 这是问题事实列表，因为它们在解决过程中不会改变。
- 包含所有房间有一个 **roomList** 字段

- 这是问题事实列表，因为它们在解决过程中不会改变。

- 带有所有节节的 **lessonList** 字段
  - 这是计划实体列表，因为他们在解决过程中发生了变化。
  - 在每个课 中 ：
    - **timeslot** 和 **room** 字段的值通常为 **null**，因此取消分配。它们是规划变量。
    - 其他字段（如 主题、**S quill r** 和 **studentGroup** ）已被填写。这些字段是问题属性。

但是，这个类也是解决方案的输出：

- 每个 **lessonList** 字段在解决后都有非 **null time lot** 和 **room** 字段
- 代表输出解决方案质量 的分数 字段，如 **0hard/-5soft**

流程

创建 **src/main/java/com/example/domain/TimeTable.java** 类：

```
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;

@PlanningSolution
public class TimeTable {
```

```
    @ValueRangeProvider(id = "timeslotRange")
    @ProblemFactCollectionProperty
    private List<Timeslot> timeslotList;

    @ValueRangeProvider(id = "roomRange")
    @ProblemFactCollectionProperty
    private List<Room> roomList;

    @PlanningEntityCollectionProperty
    private List<Lesson> lessonList;

    @PlanningScore
    private HardSoftScore score;

    private TimeTable() {
    }

    public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
            List<Lesson> lessonList) {
        this.timeslotList = timeslotList;
        this.roomList = roomList;
        this.lessonList = lessonList;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public List<Timeslot> getTimeslotList() {
        return timeslotList;
    }

    public List<Room> getRoomList() {
        return roomList;
    }

    public List<Lesson> getLessonList() {
        return lessonList;
    }

    public HardSoftScore getScore() {
        return score;
    }

}
```

值范围供应商

timeslotList *字段是一个值范围* **provider**。*它拥有* **OptaPlanner** *可以从中选择的* **Timeslot** *实例，分配给* **lesson** *实例的* **timeslot** *字段。***time lotList** *字段具有* **@ValueRangeProvider** *注释，可通过在* **Lesson** *中将* **id** *与* **@PlanningVariable** *的* **valueRangeProviderRefs** *匹配。*

遵循相同的逻辑，**roomList** 字段也有 **@ValueRangeProvider** 注释。

问题事实和规划实体属性

另外，**OptaPlanner** 需要知道它可能会更改哪些较小的实例，以及如何检索由 **TimeTableConstraintProvider** 计算的 **Timeslot** 和 **Room** 实例。

time lotList 和 **roomList** 字段具有 **@ProblemFactCollectionProperty** 注释，因此您的 **TimeTableConstraintProvider** 可以从这些实例中进行选择。

lessonList 有一个 **@PlanningEntityCollectionProperty** 注解，因此 **OptaPlanner** 可在解决过程中更改它们，并且您的 **TimeTableConstraintProvider** 也可从那些中选择。

12.5. 创建 TIMETABLE 服务

现在，您已准备好将所有内容放在一起并创建 **REST** 服务。但解决 **REST** 线程中的规划问题会导致 **HTTP** 超时问题。因此，**Spring Boot** 启动程序注入 **SolverManager**，它将在一个单独的线程池中运行 **solvers**，并可以并行解决多个数据集。

流程

创建 **src/main/java/com/example/solver/TimeTableController.java** 类：

```
package com.example.solver;

import java.util.UUID;
import java.util.concurrent.ExecutionException;

import com.example.domain.TimeTable;
import org.optaplanner.core.api.solver.SolverJob;
import org.optaplanner.core.api.solver.SolverManager;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/timeTable")
public class TimeTableController {

    @Autowired
    private SolverManager<TimeTable, UUID> solverManager;
```

```
    @PostMapping("/solve")
    public TimeTable solve(@RequestBody TimeTable problem) {
        UUID problemId = UUID.randomUUID();
        // Submit the problem to start solving
        SolverJob<TimeTable, UUID> solverJob = solverManager.solve(problemId, problem);
        TimeTable solution;
        try {
            // Wait until the solving ends
            solution = solverJob.getFinalBestSolution();
        } catch (InterruptedException | ExecutionException e) {
            throw new IllegalStateException("Solving failed.", e);
        }
        return solution;
    }

}
```

在本例中，初始实现会等待解决方法完成，这仍然可能导致 **HTTP** 超时。完整的 实现可避免 **HTTP** 超时更明确。

12.6. 设置解决者终止时间

如果您的计划应用程序没有终止设置或终止事件，它理论上会永久运行，并最终会导致 **HTTP** 超时错误。要防止这种情况发生，请使用 **optaplanner.solver.termination.spent-limit** 参数指定应用程序终止的时间长度。在大多数应用程序中，将时间至少设置为五分钟**(5m)**。但是，在 **Timetable** 示例中，将解决时间限制为 **5** 秒，这不足以避免 **HTTP** 超时。

流程

使用以下内容创建 **src/main/resources/application.properties** 文件：

```
quarkus.optaplanner.solver.termination.spent-limit=5s
```

12.7. 使应用程序可执行

完成 **OptaPlanner Spring Boot timetable** 项目的红帽构建后，将所有内容打包成由标准 **Java main**（） 方法驱动的单个可执行 **JAR** 文件。

先决条件

- 您已完成了 **OptaPlanner Spring Boot timetable** 项目。

流程

1. 使用以下内容创建 **TimeTableSpringBootApp.java** 类：

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class TimeTableSpringBootApp {

    public static void main(String[] args) {
        SpringApplication.run(TimeTableSpringBootApp.class, args);
    }

}
```

2. 将 **Spring Initializr** 创建的 **src/main/java/com/example/DemoApplication.java** 类替换为 **TimeTableSpringBootApp.java** 类。

3. 将 **TimeTableSpringBootApp.java** 类作为常规 **Java** 应用程序的主类运行。

12.7.1. 试用 timetable 应用程序

启动红帽构建的 **OptaPlanner Spring Boot timetable** 应用程序后，您可以使用您想要的任何 **REST** 客户端测试 **REST** 服务。这个示例使用 **Linux curl** 命令发送 **POST** 请求。

先决条件

- **OptaPlanner Spring Boot timetable** 应用程序正在运行。

流程

使用以下命令：

```
$ curl -i -X POST http://localhost:8080/timeTable/solve -H "Content-Type:application/json" -d
'{"timeslotList":[{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"}],"roomList":[{"name":"Room
A"},{"name":"Room B"}],"lessonList":[{"id":1,"subject":"Math","teacher":"A. Turing","studentGroup":"9th
grade"},{"id":2,"subject":"Chemistry","teacher":"M. Curie","studentGroup":"9th grade"},
{"id":3,"subject":"French","teacher":"M. Curie","studentGroup":"10th grade"},
{"id":4,"subject":"History","teacher":"I. Jones","studentGroup":"10th grade"}]}'
```

大约 **5** 秒后，在 **application.properties** 中定义的终止时间花费了时间，服务会返回类似以下示例的输出：

```
HTTP/1.1 200
Content-Type: application/json
...

{"timeslotList":...,"roomList":...,"lessonList":[{"id":1,"subject":"Math","teacher":"A.
Turing","studentGroup":"9th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":{"name":"Room A"}},
{"id":2,"subject":"Chemistry","teacher":"M. Curie","studentGroup":"9th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":{"name":"Room A"}},
{"id":3,"subject":"French","teacher":"M. Curie","studentGroup":"10th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"08:30:00","endTime":"09:30:00"},"room":{"name":"Room B"}},
{"id":4,"subject":"History","teacher":"I. Jones","studentGroup":"10th grade","timeslot":
{"dayOfWeek":"MONDAY","startTime":"09:30:00","endTime":"10:30:00"},"room":{"name":"Room
B"}}],"score":"0hard/0soft"}
```

请注意，应用程序被分配了这四个的时间插槽之一，以及两个房间的一个。另请注意，它符合所有硬约束。例如，**M. Curie** 的两节课时间不同。

在服务器端，**info** 日志显示在 **5** 秒内执行什么 **OptaPlanner**：

```
... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score calculation
speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score calculation
speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed (28524/sec),
phase total (2), environment mode (REPRODUCIBLE).
```

12.7.2. 测试应用

良好的应用程序包括测试覆盖。这个示例测试了 **OptaPlanner Spring Boot** 应用程序的 **Timetable Red Hat build**。它使用 **JUnit** 测试来生成测试数据集，并将其发送到 **TimeTableController** 以解决。

流程

使用以下内容创建 **src/test/java/com/example/solver/TimeTableControllerTest.java** 类：

```
package com.example.solver;

import java.time.DayOfWeek;
import java.time.LocalTime;
import java.util.ArrayList;
```

```
import java.util.List;

import com.example.domain.Lesson;
import com.example.domain.Room;
import com.example.domain.TimeTable;
import com.example.domain.Timeslot;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@SpringBootTest(properties = {
        "optaplanner.solver.termination.spent-limit=1h", // Effectively disable this termination in
favor of the best-score-limit
        "optaplanner.solver.termination.best-score-limit=0hard/*soft"})
public class TimeTableControllerTest {

    @Autowired
    private TimeTableController timeTableController;

    @Test
    @Timeout(600_000)
    public void solve() {
        TimeTable problem = generateProblem();
        TimeTable solution = timeTableController.solve(problem);
        assertFalse(solution.getLessonList().isEmpty());
        for (Lesson lesson : solution.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(solution.getScore().isFeasible());
    }

    private TimeTable generateProblem() {
        List<Timeslot> timeslotList = new ArrayList<>();
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30), LocalTime.of(9,
30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>();
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
```

```
        lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
        lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
        lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
        lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
        return new TimeTable(timeslotList, roomList, lessonList);
    }

}
```

此测试会验证在解决后，所有课程都会分配给一个时间窗和房间。它还会验证它是否发现可行的解决方案（没有硬限制）。

通常，该解决方法在 **200** 毫秒内找到可行的解决方案。请注意，**@SpringBootTest** 注释 的属性 如何覆盖解决器终止，以便在找到可行的解决方案**(0hard/*soft)**时立即终止。这可避免硬编码解决程序时间，因为单元测试可能会在任意硬件上运行。这种方法可确保测试用时足够长来查找可行的解决方案，即使在较慢的系统上也是如此。但是，即使在快速系统中，它也无法运行时间比严格要长的时间更长。

12.7.3. 日志记录

完成 **OptaPlanner Spring Boot timetable** 应用程序的构建后，您可以使用日志信息来帮助微调 **ConstraintProvider** 中的约束。查看 **info** 日志文件中的分数计算速度，以评估对您的限制更改的影响。以 **debug** 模式运行应用程序，显示应用程序接受的每个步骤，或使用 **trace** 日志记录记录每一步和每次移动。

流程

1. 运行定时应用程序的时间，例如五分钟。

2. 查看 日志文件中的 分数计算速度，如下例所示：

```
... Solving ended: ..., score calculation speed (29455/sec), ...
```

3. 更改约束，再次运行计划应用程序相同的时间，并查看 日志文件中 记录的分数计算速度。

4. 以 **debug** 模式运行应用程序以记录每个步骤：

- 要从命令行运行调试模式，请使用 **-D** 系统属性。

- 要更改 **application.properties** 文件中的日志记录，请在该文件中添加以下行：

```
logging.level.org.optaplanner=debug
```

以下示例显示了在 **debug** 模式下 日志文件 的输出：

```
... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode (REPRODUCIBLE), random (JDK with seed 0).
...     CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15), picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
...     CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15), picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
...
```

5. 使用 **trace logging** 显示每个步骤以及每个步骤的每一个移动。

12.8. 添加数据库和 UI 集成

在使用 **Spring Boot** 创建红帽构建的 **OptaPlanner** 应用程序示例后，添加数据库和 **UI** 集成。

前提条件

- 您已创建了 **OptaPlanner Spring Boot timetable** 示例。

流程

1. 为 **Timeslot**、**Room** 和 **Lesson** 创建 **Java Persistence API(JPA)**存储库。有关创建 **JPA** 软件仓库的详情，请参考 **Spring** 网站上 使用 **JPA** 访问数据。

2. 通过 **REST** 公开 **JPA** 存储库。有关公开软件仓库的详情，请参阅 **Spring** 网站上 使用 **REST** 访问 **JPA** 数据。

3.

*构建一个可 时间的***Repository** *常见问题解答，以在单个事务中读取和写入一个* ***TimeTable***。

4. *调整* ***TimeTableController***，*如下例所示：*

```
package com.example.solver;

import com.example.domain.TimeTable;
import com.example.persistence.TimeTableRepository;
import org.optaplanner.core.api.score.ScoreManager;
import org.optaplanner.core.api.solver.SolverManager;
import org.optaplanner.core.api.solver.SolverStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/timeTable")
public class TimeTableController {

    @Autowired
    private TimeTableRepository timeTableRepository;
    @Autowired
    private SolverManager<TimeTable, Long> solverManager;
    @Autowired
    private ScoreManager<TimeTable> scoreManager;

    // To try, GET http://localhost:8080/timeTable
    @GetMapping()
    public TimeTable getTimeTable() {
        // Get the solver status before loading the solution
        // to avoid the race condition that the solver terminates between them
        SolverStatus solverStatus = getSolverStatus();
        TimeTable solution =
timeTableRepository.findById(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
        scoreManager.updateScore(solution); // Sets the score
        solution.setSolverStatus(solverStatus);
        return solution;
    }

    @PostMapping("/solve")
    public void solve() {

solverManager.solveAndListen(TimeTableRepository.SINGLETON_TIME_TABLE_ID,
                timeTableRepository::findById,
                timeTableRepository::save);
    }

    public SolverStatus getSolverStatus() {
        return
solverManager.getSolverStatus(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
```

```
    }

    @PostMapping("/stopSolving")
    public void stopSolving() {

solverManager.terminateEarly(TimeTableRepository.SINGLETON_TIME_TABLE_ID);
    }

}
```

为简单起见，此代码只处理一个 可维护的 实例，但要直接启用多租户，并并行处理不同高中中不同中级的多个可 时间 实例。

getTimeTable（） 方法返回来自数据库的最新时间表。它使用 **ScoreManager** （自动注入）来计算该时间表的分数，以便 **UI** 可以显示分数。

solve（） 方法启动一个作业，以解决当前可时间表，并将时间窗和房分配存储在数据库中。它使用 **SolverManager.solveAndListen**（） 方法侦听中间最佳解决方案，并相应地更新数据库。这可让 **UI** 显示后端仍在解决过程中的进度。

5. 现在，**solve**（） 方法会立即返回，请调整 **TimeTableControllerTest**，如下例所示：

```
package com.example.solver;

import com.example.domain.Lesson;
import com.example.domain.TimeTable;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;
import org.optaplanner.core.api.solver.SolverStatus;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@SpringBootTest(properties = {
        "optaplanner.solver.termination.spent-limit=1h", // Effectively disable this
termination in favor of the best-score-limit
        "optaplanner.solver.termination.best-score-limit=0hard/*soft"})
public class TimeTableControllerTest {

    @Autowired
    private TimeTableController timeTableController;

    @Test
    @Timeout(600_000)
    public void solveDemoDataUntilFeasible() throws InterruptedException {
```

```
        timeTableController.solve();
        TimeTable timeTable = timeTableController.getTimeTable();
        while (timeTable.getSolverStatus() != SolverStatus.NOT_SOLVING) {
            // Quick polling (not a Test Thread Sleep anti-pattern)
            // Test is still fast on fast systems and doesn't randomly fail on slow systems.
            Thread.sleep(20L);
            timeTable = timeTableController.getTimeTable();
        }
        assertFalse(timeTable.getLessonList().isEmpty());
        for (Lesson lesson : timeTable.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(timeTable.getScore().isFeasible());
    }

}
```

6. 轮询最新的解决方案，直到解决解决为止。

7. 为了视觉化呈现时间，请在这些 **REST** 方法之上构建具有吸引力的 **Web UI**。

12.9. 使用 MICROMETER 和 PROMETHEUS 来监控 SCHOOL TIMETABLE OPTAPLANNER SPRING BOOT 应用程序

OptaPlanner 通过 **Micrometer** 公开指标数据，这是 **Java** 应用程序的指标检测库。您可以将 **Micrometer** 与 **Prometheus** 搭配使用以监控 **school timetable** 应用程序中 **OptaPlanner solver**。

先决条件

- 您已创建了 **Spring Boot OptaPlanner school** 时间应用程序。
- 已安装 **Prometheus**。有关安装 **Prometheus** 的详情，请查看 **Prometheus** 网站。

流程

1. 导航至 技术/**java-spring-boot** 目录。
2. 将 **Micrometer Prometheus** 依赖项添加到 **school timetable pom.xml** 文件中：

```
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
<dependency>
 <groupId>io.micrometer</groupId>
 <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

3. 在 **application.properties** 文件中添加以下属性：

```
management.endpoints.web.exposure.include=metrics,prometheus
```

4. 启动 **school timetable** 应用程序：

```
mvn spring-boot:run
```

5. 在 **Web** 浏览器中打开 **http://localhost:8080/actuator/prometheus**。

第 13 章 红帽构建的 OPTAPLANNER 和 JAVA：一个生态快速入门指南

本指南指导您了解通过 **OptaPlanner** 的约束解决 **artificial** 智能**(AI)**创建简单 **Java** 应用程序的过程。您将构建能为学生和老师优化学校的命令行应用程序：

```
...
INFO  Solving ended: time spent (5000), best score (0hard/9soft), ...
INFO
INFO  |            | Room A     | Room B     | Room C     |
INFO  |------------|------------|------------|------------|
INFO  | MON 08:30  | English    | Math       |            |
INFO  |            | I. Jones   | A. Turing  |            |
INFO  |            | 9th grade  | 10th grade |            |
INFO  |------------|------------|------------|------------|
INFO  | MON 09:30  | History    | Physics    |            |
INFO  |            | I. Jones   | M. Curie   |            |
INFO  |            | 9th grade  | 10th grade |            |
INFO  |------------|------------|------------|------------|
INFO  | MON 10:30  | History    | Physics    |            |
INFO  |            | I. Jones   | M. Curie   |            |
INFO  |            | 10th grade | 9th grade  |            |
INFO  |------------|------------|------------|------------|
...
INFO  |------------|------------|------------|------------|
```

您的应用程序会自动将 **Lesson** 实例分配给 **Timeslot** 和 **Room** 实例，方法是使用 **AI** 遵循硬和软调度限制，例如：

- 个空间最多可以同时有一处。
- 老师可以同时在上课时进行教学。
- 学员最多可以同时参加。
- 老师喜欢在同一房间教授所有课程。
- 指导者更倾向于在课间学习顺序课时和不类差距。
- 学员对相同主题的后续课类不同。

数学会上讲，该学校的时间范围是一个 **NP-hard** 问题。这意味着很难扩展。只需浏览所有可能的组合，需要数以百万计的、非内部的数据集，即使在超级计算机中也是如此。幸运的是，**AI** 约束解决诸如 **OptaPlanner** 等高级算法具有在合理的时间内提供最接近的解决方案。

先决条件

- 已安装了 **OpenJDK(JDK)11**。红帽构建的 **Open JDK** 可从红帽客户门户网站的 **Software Downloads** 页面获取（需要登录）。
- 已安装 **Apache Maven 3.6** 或更高版本。**Maven** 可以从 **Apache Maven Project** 网站获得。
- **IDE**，如 **IntelliJ IDEA**、**VS Code** 或 **Eclipse**

13.1. 创建 MAVEN 或 GRADLE 构建文件并添加依赖项

您可以将 **Maven** 或 **Gradle** 用于 **OptaPlanner unitable** 应用程序。创建构建文件后，添加以下依赖项：

- **OptaPlanner-core** （编译范围）以解决 **school timetable** 问题
- **OptaPlanner-test** （测试 **JUnit** 范围）测试 **schooltabling** 约束
- 一个实现（如 **logback-classic** （**runtime** 范围）来查看 **OptaPlanner** 采用的步骤

流程

1. 创建 **Maven** 或 **Gradle** 构建文件。
2. 将 **optaplanner-core**、**optaplanner-test** 和 **logback-classic** 依赖项添加到构建文件中：
   - 对于 **Maven**，在 **pom.xml** 文件中添加以下依赖项：

```
<dependency>
```

```
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-core</artifactId>
 </dependency>

 <dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-test</artifactId>
  <scope>test</scope>
 </dependency>

 <dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>1.2.3</version>
 </dependency>
```

以下示例显示了完整的 ***pom.xml*** *文件。*

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>org.acme</groupId>
 <artifactId>optaplanner-hello-world-school-timetabling-quickstart</artifactId>
 <version>1.0-SNAPSHOT</version>

 <properties>
  <maven.compiler.release>11</maven.compiler.release>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

  <version.org.optaplanner>8.13.0.Final-redhat-00013</version.org.optaplanner>
  <version.org.logback>1.2.3</version.org.logback>

  <version.compiler.plugin>3.8.1</version.compiler.plugin>
  <version.surefire.plugin>3.0.0-M5</version.surefire.plugin>
  <version.exec.plugin>3.0.0</version.exec.plugin>
 </properties>

 <dependencyManagement>
  <dependencies>
   <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-bom</artifactId>
    <version>${version.org.optaplanner}</version>
    <type>pom</type>
    <scope>import</scope>
   </dependency>
   <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>${version.org.logback}</version>
```

```
    </dependency>
  </dependencies>
</dependencyManagement>

<dependencies>
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-core</artifactId>
  </dependency>
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <scope>runtime</scope>
  </dependency>

  <!-- Testing -->
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${version.compiler.plugin}</version>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${version.surefire.plugin}</version>
    </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <version>${version.exec.plugin}</version>
      <configuration>
        <mainClass>org.acme.schooltimetabling.TimeTableApp</mainClass>
      </configuration>
    </plugin>
  </plugins>
</build>

<repositories>
  <repository>
    <id>jboss-public-repository-group</id>
    <url>https://repository.jboss.org/nexus/content/groups/public/</url>
    <releases>
      <!-- Get releases only from Maven Central which is faster. -->
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
```

```
    </repository>
  </repositories>
</project>
```

- 对于 **Gradle**，请在 **gradle.build** 文件中添加以下依赖项：

```
dependencies {
    implementation "org.optaplanner:optaplanner-core:${optaplannerVersion}"
    runtimeOnly "ch.qos.logback:logback-classic:${logbackVersion}"

    testImplementation "org.optaplanner:optaplanner-test:${optaplannerVersion}"
}
```

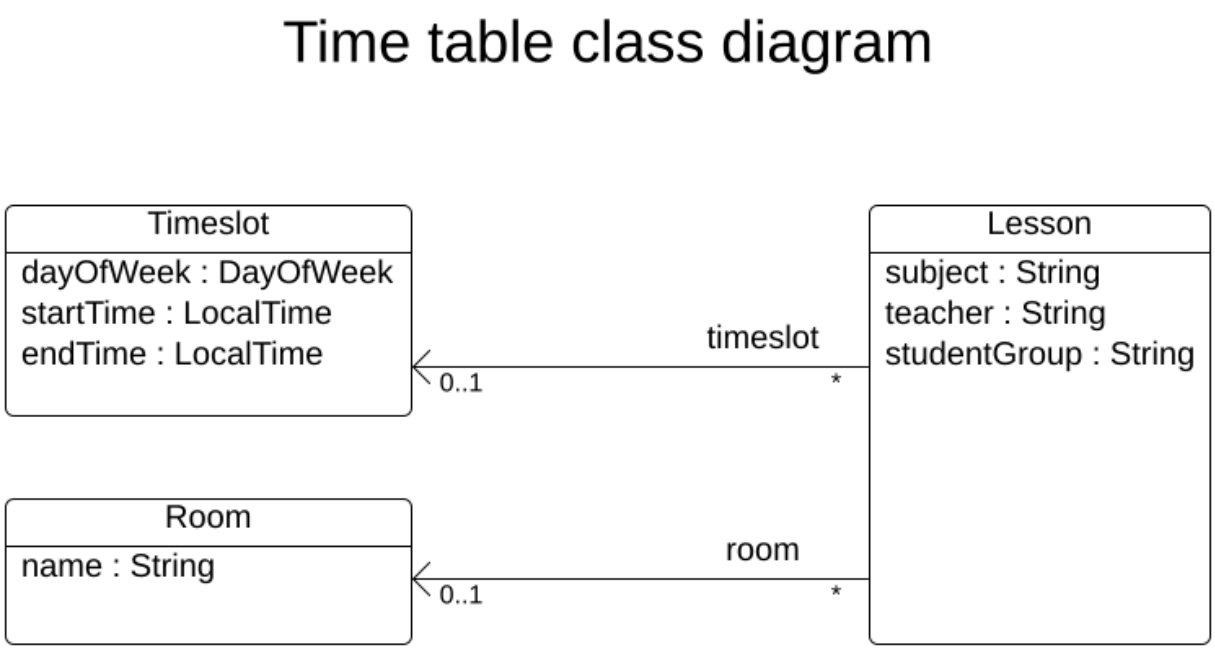
以下示例显示了完成的 **gradle.build** 文件。

```
plugins {
    id "java"
    id "application"
}

def optaplannerVersion = "{project-version}"
def logbackVersion = "1.2.3"

group = "org.acme"
version = "0.1.0-SNAPSHOT"

repositories {
    mavenCentral()
}

dependencies {
    implementation "org.optaplanner:optaplanner-core:${optaplannerVersion}"
    runtimeOnly "ch.qos.logback:logback-classic:${logbackVersion}"

    testImplementation "org.optaplanner:optaplanner-test:${optaplannerVersion}"
}

java {
    sourceCompatibility = JavaVersion.VERSION_11
    targetCompatibility = JavaVersion.VERSION_11
}

compileJava {
    options.encoding = "UTF-8"
    options.compilerArgs << "-parameters"
}

compileTestJava {
    options.encoding = "UTF-8"
}
```

```
application {
    mainClass = "org.acme.schooltimetabling.TimeTableApp"
}

test {
    // Log the test execution results.
    testLogging {
        events "passed", "skipped", "failed"
    }
}
```

13.2. 对域对象建模

红帽构建的 **OptaPlanner** 时间表项目的目标是将每节分配给一个时间段和一个房间。为此，请添加三个类： **Timeslot**、**Lesson** 和 **Room**，如下图所示：



Timeslot

Timeslot 类表示在未教授课程的时间间隔，例如： **Monday 10:30 - 11:30** 或 **Tuesday 13:30 - 14:30 - 14:30**。在这个示例中，所有时间插槽都有相同的持续时间，且在 **lunch** 或其他中断期间没有时间插槽。

时间段没有日期，因为高校的计划只每周重复。无法进行持续 规划。次性被称为 问题事实，因为在解决过程中不会有定时实例变化。此类类不需要任何特定于 **OptaPlanner** 的注解。

room

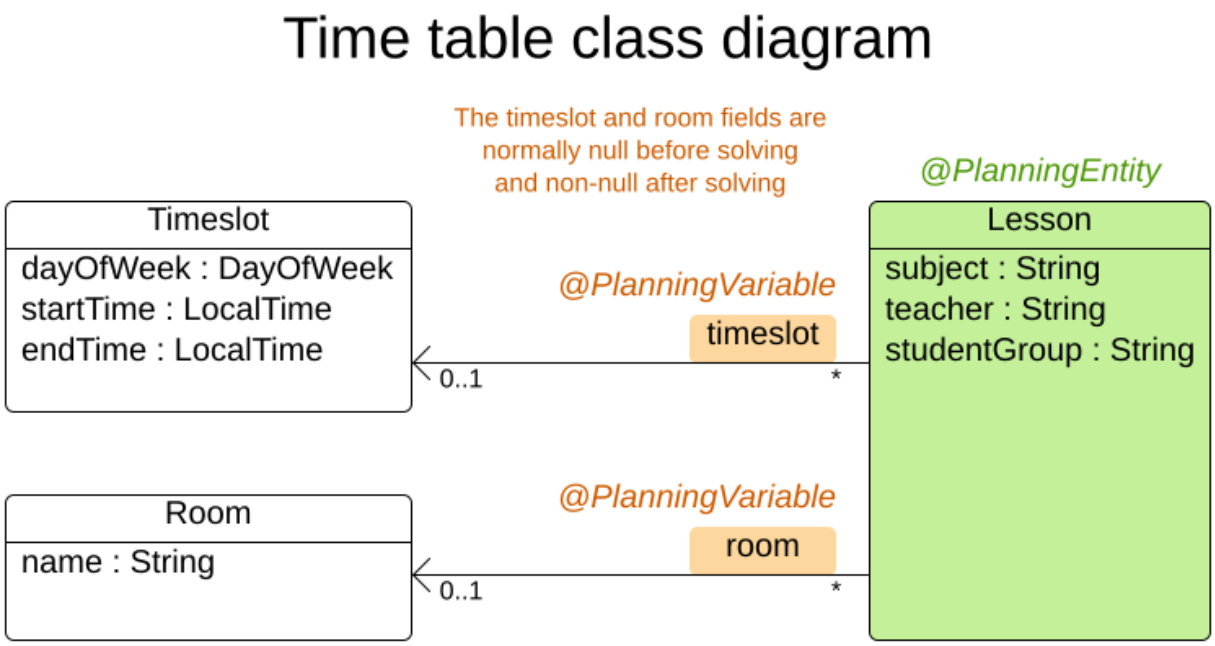
Room 类代表了课程讲授的方式，例如 Room A 或 Room B。在本例中，所有空间都没有容量限制，它们可容纳所有课时。

房间 实例在解决过程中不会改变，因此 Room 也是 问题。

lesson

在课时，由课课课表示，向班级教授一个主题，例如： A.Turing for a 9th grade 或 Chemistry by M.Curie 为 10th grade。如果一个主题每周都会为同一学员组讲授多次，则有多个小的、只能由 id 区分的实例。例如，第 9 分之时，每周有 6 个数学。

在寻求过程中，OptaPlanner 改变了 lesson 类的 timelot 和 room 字段，从而将每个较小的时间分配给一个时间段和一个房间。 因为 OptaPlanner 更改了这些字段，所以 lesson 是一个 规划实体 ：



上图中的大多数字段包含输入数据，但 orange 字段除外。在输入数据中 未分配的 时间和房间 字段 (null)，并在输出数据中分配（而不是 null）。OptaPlanner 在解决过程中更改这些字段。这些字段称为 规划变量。为了 OptaPlanner 可以识别它们，timelot 和 room 字段都需要 @PlanningVariable 注解。它们包含类(Lesson)需要 @PlanningEntity 注释。

流程

1. *创建 src/main/java/com/example/domain/Timeslot.java 类：*

```
package com.example.domain;
```

```
import java.time.DayOfWeek;
import java.time.LocalTime;

public class Timeslot {

    private DayOfWeek dayOfWeek;
    private LocalTime startTime;
    private LocalTime endTime;

    private Timeslot() {
    }

    public Timeslot(DayOfWeek dayOfWeek, LocalTime startTime, LocalTime endTime) {
        this.dayOfWeek = dayOfWeek;
        this.startTime = startTime;
        this.endTime = endTime;
    }

    @Override
    public String toString() {
        return dayOfWeek + " " + startTime.toString();
    }

    // ********************************
    // Getters and setters
    // ********************************

    public DayOfWeek getDayOfWeek() {
        return dayOfWeek;
    }

    public LocalTime getStartTime() {
        return startTime;
    }

    public LocalTime getEndTime() {
        return endTime;
    }

}
```

*请注意，**toString**（） 方法保留了输出短片，以便可以更轻松地读取 **OptaPlanner** 的 **DEBUG** 或 **abrt** 日志，如稍后所示。*

2. *创建 **src/main/java/com/example/domain/Room.java** 类：*

```
package com.example.domain;

public class Room {

    private String name;
```

```
    private Room() {
    }

    public Room(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public String getName() {
        return name;
    }

}
```

3. 创建 *src/main/java/com/example/domain/Lesson.java* 类：

```
package com.example.domain;

import org.optaplanner.core.api.domain.entity.PlanningEntity;
import org.optaplanner.core.api.domain.variable.PlanningVariable;

@PlanningEntity
public class Lesson {

    private Long id;

    private String subject;
    private String teacher;
    private String studentGroup;

    @PlanningVariable(valueRangeProviderRefs = "timeslotRange")
    private Timeslot timeslot;

    @PlanningVariable(valueRangeProviderRefs = "roomRange")
    private Room room;

    private Lesson() {
    }

    public Lesson(Long id, String subject, String teacher, String studentGroup) {
        this.id = id;
        this.subject = subject;
        this.teacher = teacher;
        this.studentGroup = studentGroup;
    }
```

```
    @Override
    public String toString() {
        return subject + "(" + id + ")";
    }

    // ********************************
    // Getters and setters
    // ********************************

    public Long getId() {
        return id;
    }

    public String getSubject() {
        return subject;
    }

    public String getTeacher() {
        return teacher;
    }

    public String getStudentGroup() {
        return studentGroup;
    }

    public Timeslot getTimeslot() {
        return timeslot;
    }

    public void setTimeslot(Timeslot timeslot) {
        this.timeslot = timeslot;
    }

    public Room getRoom() {
        return room;
    }

    public void setRoom(Room room) {
        this.room = room;
    }

}
```

Lesson 类具有 **@PlanningEntity** 注释，因此 **OptaPlanner** 知道这个类在解决过程中发生了变化，因为它包含一个或多个计划变量。

time lot 字段具有 **@PlanningVariable** 注释，因此 **OptaPlanner** 知道它可以更改其值。为了查找要分配给此字段的潜在的 **Timeslot** 实例，**OptaPlanner** 使用 **valueRangeProviderRefs** 属性连接到一个提供 **List<Timeslot>** 的值供应商。有关值范围供应商的信息，请参阅 **第 13.4** 节 “在规划解决方案中收集域对象”。

出于同样原因，**room** 字段还具有 **@PlanningVariable** 注释。

13.3. 定义限制并计算分数

当解决问题时，分数 代表特定解决方案的质量。得分越高。红帽构建的 **OptaPlanner** 寻找最佳解决方案，这是可用时间获得最高分数的解决方案。这可能是 最佳解决方案。

因为可时间的示例用例具有硬和软约束，因此请使用 **HardSoftScore** 类来代表分数：

- 硬限制不得被破坏。例如： 一个房间可以有以上时间。
- 软限制不应该被破坏。例如： 更喜欢在单个房里的教学方式。

硬约束的加法是相对于其他硬约束的权重。软限制会高于其他软限制。硬限制始终大于软限制，无论其对应的权重如何。

要计算分数，您可以实施 **EasyScoreCalculator** 类：

```
public class TimeTableEasyScoreCalculator implements EasyScoreCalculator<TimeTable> {

   @Override
   public HardSoftScore calculateScore(TimeTable timeTable) {
      List<Lesson> lessonList = timeTable.getLessonList();
      int hardScore = 0;
      for (Lesson a : lessonList) {
         for (Lesson b : lessonList) {
            if (a.getTimeslot() != null && a.getTimeslot().equals(b.getTimeslot())
                  && a.getId() < b.getId()) {
               // A room can accommodate at most one lesson at the same time.
               if (a.getRoom() != null && a.getRoom().equals(b.getRoom())) {
                  hardScore--;
               }
               // A teacher can teach at most one lesson at the same time.
               if (a.getTeacher().equals(b.getTeacher())) {
                  hardScore--;
               }
               // A student can attend at most one lesson at the same time.
               if (a.getStudentGroup().equals(b.getStudentGroup())) {
                  hardScore--;
               }
            }
```

```
            }
        }
        int softScore = 0;
        // Soft constraints are only implemented in the "complete" implementation
        return HardSoftScore.of(hardScore, softScore);
    }

}
```

不幸的是，这个解决方案无法进行很好的扩展，因为它是非增量：每次分配给不同的时间段或房间，都会重新评估所有课程来计算新分数。

更好的解决方法是创建一个 **src/main/java/com/example/solver/TimeTableConstraintProvider.java** 类来执行增量分数计算。此类使用 **OptaPlanner** 的 **ConstraintStream API**，它由 **Java 8 Streams** 和 **SQL** 产生。**ConstraintProvider** 缩放比 **EasyScoreCalculator** 更好的度量：**O(n)**而不是 **O(n)**。

流程

创建以下 **src/main/java/com/example/solver/TimeTableConstraintProvider.java** 类：

```
package com.example.solver;

import com.example.domain.Lesson;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;
import org.optaplanner.core.api.score.stream.Constraint;
import org.optaplanner.core.api.score.stream.ConstraintFactory;
import org.optaplanner.core.api.score.stream.ConstraintProvider;
import org.optaplanner.core.api.score.stream.Joiners;

public class TimeTableConstraintProvider implements ConstraintProvider {

    @Override
    public Constraint[] defineConstraints(ConstraintFactory constraintFactory) {
        return new Constraint[] {
                // Hard constraints
                roomConflict(constraintFactory),
                teacherConflict(constraintFactory),
                studentGroupConflict(constraintFactory),
                // Soft constraints are only implemented in the "complete" implementation
        };
    }

    private Constraint roomConflict(ConstraintFactory constraintFactory) {
        // A room can accommodate at most one lesson at the same time.

        // Select a lesson ...
        return constraintFactory.from(Lesson.class)
                // ... and pair it with another lesson ...
                .join(Lesson.class,
                        // ... in the same timeslot ...
```

```
                Joiners.equal(Lesson::getTimeslot),
                // ... in the same room ...
                Joiners.equal(Lesson::getRoom),
                // ... and the pair is unique (different id, no reverse pairs)
                Joiners.lessThan(Lesson::getId))
        // then penalize each pair with a hard weight.
        .penalize("Room conflict", HardSoftScore.ONE_HARD);
}

private Constraint teacherConflict(ConstraintFactory constraintFactory) {
    // A teacher can teach at most one lesson at the same time.
    return constraintFactory.from(Lesson.class)
            .join(Lesson.class,
                Joiners.equal(Lesson::getTimeslot),
                Joiners.equal(Lesson::getTeacher),
                Joiners.lessThan(Lesson::getId))
            .penalize("Teacher conflict", HardSoftScore.ONE_HARD);
}

private Constraint studentGroupConflict(ConstraintFactory constraintFactory) {
    // A student can attend at most one lesson at the same time.
    return constraintFactory.from(Lesson.class)
            .join(Lesson.class,
                Joiners.equal(Lesson::getTimeslot),
                Joiners.equal(Lesson::getStudentGroup),
                Joiners.lessThan(Lesson::getId))
            .penalize("Student group conflict", HardSoftScore.ONE_HARD);
}

}
```

13.4. 在规划解决方案中收集域对象

TimeTable 实例嵌套了单个 **dataset** 的所有 **Timeslot**、**Room** 和 **Lesson** 实例。另外，由于它包含了 **allon**，每个都具有特定规划变量状态，所以它是一个 规划解决方案，并有分数：

- 如果课程仍然没有被签名，那么它是一个 未初始化 的解决方案，例如分数为 **-4init/0hard/0soft** 的解决方案。
- 如果损坏硬限制，那么它是一种 不可预见的 解决方案，例如分数为 **-2hard/-3soft** 的解决方案。
- 如果遵循所有硬约束，那么它是一个可行的解决方案，例如，分数为 **0hard/-7soft** 的解决方案。

TimeTable 类有一个 **@PlanningSolution** 注释，因此红帽构建的 **OptaPlanner** 知道该类包含所有输

入和输出数据。

具体来说，这个类是问题的输入：

- 带有所有时间窗的 **timeslotList** 字段
  - 这是问题事实列表，因为它们在解决过程中不会改变。
- 包含所有房间有一个 **roomList** 字段
  - 这是问题事实列表，因为它们在解决过程中不会改变。
- 带有所有节节的 **lessonList** 字段
  - 这是计划实体列表，因为他们在解决过程中发生了变化。
  - 在每个课 中 ：
    - **timeslot** 和 **room** 字段的值通常为 **null**，因此取消分配。它们是规划变量。
    - 其他字段（如 主题、**S quill r** 和 **studentGroup** ）已被填写。这些字段是问题属性。

但是，这个类也是解决方案的输出：

- 每个 **lessonList** 字段在解决后都有非 **null time lot** 和 **room** 字段
- 代表输出解决方案质量 的分数 字段，如 **0hard/-5soft**

流程

*创建 **src/main/java/com/example/domain/TimeTable.java** 类：*

```
package com.example.domain;

import java.util.List;

import org.optaplanner.core.api.domain.solution.PlanningEntityCollectionProperty;
import org.optaplanner.core.api.domain.solution.PlanningScore;
import org.optaplanner.core.api.domain.solution.PlanningSolution;
import org.optaplanner.core.api.domain.solution.ProblemFactCollectionProperty;
import org.optaplanner.core.api.domain.valuerange.ValueRangeProvider;
import org.optaplanner.core.api.score.buildin.hardsoft.HardSoftScore;

@PlanningSolution
public class TimeTable {

    @ValueRangeProvider(id = "timeslotRange")
    @ProblemFactCollectionProperty
    private List<Timeslot> timeslotList;

    @ValueRangeProvider(id = "roomRange")
    @ProblemFactCollectionProperty
    private List<Room> roomList;

    @PlanningEntityCollectionProperty
    private List<Lesson> lessonList;

    @PlanningScore
    private HardSoftScore score;

    private TimeTable() {
    }

    public TimeTable(List<Timeslot> timeslotList, List<Room> roomList,
            List<Lesson> lessonList) {
        this.timeslotList = timeslotList;
        this.roomList = roomList;
        this.lessonList = lessonList;
    }

    // ********************************
    // Getters and setters
    // ********************************

    public List<Timeslot> getTimeslotList() {
        return timeslotList;
    }

    public List<Room> getRoomList() {
        return roomList;
    }

    public List<Lesson> getLessonList() {
        return lessonList;
```

```
    }

    public HardSoftScore getScore() {
        return score;
    }

}
```

值范围供应商

timeslotList *字段是一个值范围* **provider**。*它拥有* **OptaPlanner** *可以从中选择的* **Timeslot** *实例，分配给* **lesson** *实例的* **timeslot** *字段。***time lotList** *字段具有* **@ValueRangeProvider** *注释，可通过在* **Lesson** *中将* **id** *与* **@PlanningVariable** *的* **valueRangeProviderRefs** *匹配。*

*遵循相同的逻辑，***roomList** *字段也有* **@ValueRangeProvider** *注释。*

问题事实和规划实体属性

*另外，***OptaPlanner** *需要知道它可能会更改哪些较小的实例，以及如何检索由* **TimeTableConstraintProvider** *计算的* **Timeslot** *和* **Room** *实例。*

***time lotList** *和* **roomList** *字段具有* **@ProblemFactCollectionProperty** *注释，因此您的* **TimeTableConstraintProvider** *可以从这些实例中进行选择。*

***lessonList** *有一个* **@PlanningEntityCollectionProperty** *注解，因此* **OptaPlanner** *可在解决过程中更改它们，并且您的* **TimeTableConstraintProvider** *也可从那些中选择。*

13.5. TIMETABLEAPP.JAVA 类

在创建了 **school timetable** *应用的所有组件后，您将把它们放到* **TimeTableApp.java** *类中。*

***main**（） 方法执行以下任务：*

1. *创建* **Solv er onnectionFactory y**，*为每个数据集构建* **Solver**。
2. *加载数据集。*

3. 通过 **Solv er.solve**（）解决 它。

4. 视觉化呈现该数据集的解决方案。

通常，一个应用程序有一个 **SolverFactory**，用于为每个问题数据设置一个新的 **Solver** 实例，用于解决每个问题数据。**SolverFactory** 是线程安全，但 **Solver** 不是。对于 **school timetable** 应用，只有一个数据集，因此只有一个 **Solver** 实例。

以下是完成的 **TimeTableApp.java** 类：

```
package org.acme.schooltimetabling;

import java.time.DayOfWeek;
import java.time.Duration;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import org.acme.schooltimetabling.domain.Lesson;
import org.acme.schooltimetabling.domain.Room;
import org.acme.schooltimetabling.domain.TimeTable;
import org.acme.schooltimetabling.domain.Timeslot;
import org.acme.schooltimetabling.solver.TimeTableConstraintProvider;
import org.optaplanner.core.api.solver.Solver;
import org.optaplanner.core.api.solver.SolverFactory;
import org.optaplanner.core.config.solver.SolverConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class TimeTableApp {

    private static final Logger LOGGER = LoggerFactory.getLogger(TimeTableApp.class);

    public static void main(String[] args) {
        SolverFactory<TimeTable> solverFactory = SolverFactory.create(new SolverConfig()
                .withSolutionClass(TimeTable.class)
                .withEntityClasses(Lesson.class)
                .withConstraintProviderClass(TimeTableConstraintProvider.class)
                // The solver runs only for 5 seconds on this small data set.
                // It's recommended to run for at least 5 minutes ("5m") otherwise.
                .withTerminationSpentLimit(Duration.ofSeconds(10)));

        // Load the problem
        TimeTable problem = generateDemoData();
```

```
        // Solve the problem
        Solver<TimeTable> solver = solverFactory.buildSolver();
        TimeTable solution = solver.solve(problem);

        // Visualize the solution
        printTimetable(solution);
    }

    public static TimeTable generateDemoData() {
        List<Timeslot> timeslotList = new ArrayList<>(10);
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30), LocalTime.of(9,
30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(8, 30), LocalTime.of(9,
30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>(3);
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        long id = 0;
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
        lessonList.add(new Lesson(id++, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(id++, "Biology", "C. Darwin", "9th grade"));
        lessonList.add(new Lesson(id++, "History", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Physics", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "French", "M. Curie", "10th grade"));
```

```
        lessonList.add(new Lesson(id++, "Geography", "C. Darwin", "10th grade"));
        lessonList.add(new Lesson(id++, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(id++, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "10th grade"));

        return new TimeTable(timeslotList, roomList, lessonList);
    }

    private static void printTimetable(TimeTable timeTable) {
        LOGGER.info("");
        List<Room> roomList = timeTable.getRoomList();
        List<Lesson> lessonList = timeTable.getLessonList();
        Map<Timeslot, Map<Room, List<Lesson>>> lessonMap = lessonList.stream()
                .filter(lesson -> lesson.getTimeslot() != null && lesson.getRoom() != null)
                .collect(Collectors.groupingBy(Lesson::getTimeslot,
Collectors.groupingBy(Lesson::getRoom)));
        LOGGER.info("|            | " + roomList.stream()
                .map(room -> String.format("%-10s", room.getName())).collect(Collectors.joining(" |
")) + " |");
        LOGGER.info("|" + "------------|".repeat(roomList.size() + 1));
        for (Timeslot timeslot : timeTable.getTimeslotList()) {
            List<List<Lesson>> cellList = roomList.stream()
                    .map(room -> {
                        Map<Room, List<Lesson>> byRoomMap = lessonMap.get(timeslot);
                        if (byRoomMap == null) {
                            return Collections.<Lesson>emptyList();
                        }
                        List<Lesson> cellLessonList = byRoomMap.get(room);
                        if (cellLessonList == null) {
                            return Collections.<Lesson>emptyList();
                        }
                        return cellLessonList;
                    })
                    .collect(Collectors.toList());

            LOGGER.info("| " + String.format("%-10s",
                    timeslot.getDayOfWeek().toString().substring(0, 3) + " " + timeslot.getStartTime()) +
" | "
                    + cellList.stream().map(cellLessonList -> String.format("%-10s",
                            cellLessonList.stream().map(Lesson::getSubject).collect(Collectors.joining(",
"))))
                            .collect(Collectors.joining(" | "))
                    + " |");
            LOGGER.info("|            | "
                    + cellList.stream().map(cellLessonList -> String.format("%-10s",
                            cellLessonList.stream().map(Lesson::getTeacher).collect(Collectors.joining(",
"))))
                            .collect(Collectors.joining(" | "))
                    + " |");
            LOGGER.info("|            | "
                    + cellList.stream().map(cellLessonList -> String.format("%-10s",

cellLessonList.stream().map(Lesson::getStudentGroup).collect(Collectors.joining(", "))))
                            .collect(Collectors.joining(" | "))
                    + " |");
            LOGGER.info("|" + "------------|".repeat(roomList.size() + 1));
```

```
        }
        List<Lesson> unassignedLessons = lessonList.stream()
                .filter(lesson -> lesson.getTimeslot() == null || lesson.getRoom() == null)
                .collect(Collectors.toList());
        if (!unassignedLessons.isEmpty()) {
            LOGGER.info("");
            LOGGER.info("Unassigned lessons");
            for (Lesson lesson : unassignedLessons) {
                LOGGER.info("  " + lesson.getSubject() + " - " + lesson.getTeacher() + " - " +
lesson.getStudentGroup());
            }
        }
    }

}
```

main（） 方法首先创建 **SolverFactory** ：

```
SolverFactory<TimeTable> solverFactory = SolverFactory.create(new SolverConfig()
        .withSolutionClass(TimeTable.class)
        .withEntityClasses(Lesson.class)
        .withConstraintProviderClass(TimeTableConstraintProvider.class)
        // The solver runs only for 5 seconds on this small data set.
        // It's recommended to run for at least 5 minutes ("5m") otherwise.
        .withTerminationSpentLimit(Duration.ofSeconds(5)));
```

SolverFactory 创建注册 **@PlanningSolution** 类、**@PlanningEntity** 类和 **ConstraintProvider** 类（您之前创建的所有组件）。

如果没有终止设置或 **terminationEarly**（） 事件，解决者会永久运行。为避免这种情况，解决问题将解决时间限制为 **5** 秒。

5 秒后，**main**（） 方法加载问题，可解决这个问题并打印解决方案：

```
        // Load the problem
        TimeTable problem = generateDemoData();

        // Solve the problem
        Solver<TimeTable> solver = solverFactory.buildSolver();
        TimeTable solution = solver.solve(problem);

        // Visualize the solution
        printTimetable(solution);
```

solve（） 方法不会立即返回。在返回最佳解决方案前，它会持续 **5** 秒。

OptaPlanner 返回可用终止时间里 的最佳解决方案。由于 NP 划分问题的性质，最佳解决方案可能不是最佳的解决方案，特别是对于更大的数据集。增加终止时间，以有可能获得更好的解决方案。

generateDemoData（） 方法会生成 school timetable 问题来解决。

printTimetable（） 方法给控制台而生动，因此易于视觉确定是否适合自己的时间表。

13.6. 创建并运行 SCHOOL TIMETABLE 应用程序

您已完成了 school timetable Java 应用程序的所有组件，您已准备好将它们放在 TimeTableApp.java 类中并运行它。

先决条件

- *您已创建了 school timetable 应用的所有必要组件。*

流程

1. *创建 src/main/java/org/acme/schooltimetabling/TimeTableApp.java 类：*

```
package org.acme.schooltimetabling;

import java.time.DayOfWeek;
import java.time.Duration;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

import org.acme.schooltimetabling.domain.Lesson;
import org.acme.schooltimetabling.domain.Room;
import org.acme.schooltimetabling.domain.TimeTable;
import org.acme.schooltimetabling.domain.Timeslot;
import org.acme.schooltimetabling.solver.TimeTableConstraintProvider;
import org.optaplanner.core.api.solver.Solver;
import org.optaplanner.core.api.solver.SolverFactory;
import org.optaplanner.core.config.solver.SolverConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class TimeTableApp {
```

```
    private static final Logger LOGGER =
LoggerFactory.getLogger(TimeTableApp.class);

    public static void main(String[] args) {
        SolverFactory<TimeTable> solverFactory = SolverFactory.create(new
SolverConfig()
                .withSolutionClass(TimeTable.class)
                .withEntityClasses(Lesson.class)
                .withConstraintProviderClass(TimeTableConstraintProvider.class)
                // The solver runs only for 5 seconds on this small data set.
                // It's recommended to run for at least 5 minutes ("5m") otherwise.
                .withTerminationSpentLimit(Duration.ofSeconds(10)));

        // Load the problem
        TimeTable problem = generateDemoData();

        // Solve the problem
        Solver<TimeTable> solver = solverFactory.buildSolver();
        TimeTable solution = solver.solve(problem);

        // Visualize the solution
        printTimetable(solution);
    }

    public static TimeTable generateDemoData() {
        List<Timeslot> timeslotList = new ArrayList<>(10);
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30),
LocalTime.of(9, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(8, 30),
LocalTime.of(9, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.TUESDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>(3);
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        long id = 0;
```

```
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "9th grade"));
        lessonList.add(new Lesson(id++, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(id++, "Biology", "C. Darwin", "9th grade"));
        lessonList.add(new Lesson(id++, "History", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Math", "A. Turing", "10th grade"));
        lessonList.add(new Lesson(id++, "Physics", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "French", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(id++, "Geography", "C. Darwin", "10th grade"));
        lessonList.add(new Lesson(id++, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(id++, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(id++, "Spanish", "P. Cruz", "10th grade"));

        return new TimeTable(timeslotList, roomList, lessonList);
    }

    private static void printTimetable(TimeTable timeTable) {
        LOGGER.info("");
        List<Room> roomList = timeTable.getRoomList();
        List<Lesson> lessonList = timeTable.getLessonList();
        Map<Timeslot, Map<Room, List<Lesson>>> lessonMap = lessonList.stream()
                .filter(lesson -> lesson.getTimeslot() != null && lesson.getRoom() != null)
                .collect(Collectors.groupingBy(Lesson::getTimeslot,
Collectors.groupingBy(Lesson::getRoom)));
        LOGGER.info("|            | " + roomList.stream()
                .map(room -> String.format("%-10s",
room.getName())).collect(Collectors.joining(" | ")) + " |");
        LOGGER.info("|" + "------------|".repeat(roomList.size() + 1));
        for (Timeslot timeslot : timeTable.getTimeslotList()) {
            List<List<Lesson>> cellList = roomList.stream()
                    .map(room -> {
                        Map<Room, List<Lesson>> byRoomMap = lessonMap.get(timeslot);
                        if (byRoomMap == null) {
                            return Collections.<Lesson>emptyList();
                        }
                        List<Lesson> cellLessonList = byRoomMap.get(room);
                        if (cellLessonList == null) {
                            return Collections.<Lesson>emptyList();
                        }
                        return cellLessonList;
                    })
                    .collect(Collectors.toList());

            LOGGER.info("| " + String.format("%-10s",
                    timeslot.getDayOfWeek().toString().substring(0, 3) + " " +
timeslot.getStartTime()) + " | "
                    + cellList.stream().map(cellLessonList -> String.format("%-10s",
```

```
cellLessonList.stream().map(Lesson::getSubject).collect(Collectors.joining(", "))))
                    .collect(Collectors.joining(" | "))
                + " |");
        LOGGER.info("|            | "
                + cellList.stream().map(cellLessonList -> String.format("%-10s",

cellLessonList.stream().map(Lesson::getTeacher).collect(Collectors.joining(", "))))
                    .collect(Collectors.joining(" | "))
                + " |");
        LOGGER.info("|            | "
                + cellList.stream().map(cellLessonList -> String.format("%-10s",

cellLessonList.stream().map(Lesson::getStudentGroup).collect(Collectors.joining(",
"))))
                    .collect(Collectors.joining(" | "))
                + " |");
        LOGGER.info("|" + "------------|".repeat(roomList.size() + 1));
    }
    List<Lesson> unassignedLessons = lessonList.stream()
            .filter(lesson -> lesson.getTimeslot() == null || lesson.getRoom() == null)
            .collect(Collectors.toList());
    if (!unassignedLessons.isEmpty()) {
        LOGGER.info("");
        LOGGER.info("Unassigned lessons");
        for (Lesson lesson : unassignedLessons) {
            LOGGER.info("  " + lesson.getSubject() + " - " + lesson.getTeacher() + " - "
lesson.getStudentGroup());
        }
    }
  }

}
```

2. 运行 **TimeTableApp** 类作为普通 **Java** 应用程序的主要类。以下输出应结果：

```
...
INFO  |            | Room A     | Room B     | Room C     |
INFO  |------------|------------|------------|------------|
INFO  | MON 08:30  | English    | Math       |            |
INFO  |            | I. Jones   | A. Turing  |            |
INFO  |            | 9th grade  | 10th grade |            |
INFO  |------------|------------|------------|------------|
INFO  | MON 09:30  | History    | Physics    |            |
INFO  |            | I. Jones   | M. Curie   |            |
INFO  |            | 9th grade  | 10th grade |            |
...
```

3. 验证控制台输出。它是否符合所有硬约束？如果您在 **TimeTableConstraintProvider** 中注释掉 **roomConflict** 约束，会发生什么？

info 日志显示 OptaPlanner 在 5 秒里执行的操作：

```
... Solving started: time spent (33), best score (-8init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
... Construction Heuristic phase (0) ended: time spent (73), best score (0hard/0soft), score calculation
speed (459/sec), step total (4).
... Local Search phase (1) ended: time spent (5000), best score (0hard/0soft), score calculation
speed (28949/sec), step total (28398).
... Solving ended: time spent (5000), best score (0hard/0soft), score calculation speed (28524/sec),
phase total (2), environment mode (REPRODUCIBLE).
```

13.7. 测试应用

良好的应用程序包括测试覆盖。测试您时间项目中的限制和解决方法。

13.7.1. 测试 school timetable 约束

要测试隔离中可时间项目的每个约束，请在单元测试中使用 ConstraintVerifier。这会测试每个约束的基点，与其他测试隔离开来，这在添加新限制时降低了维护。

此测试会验证约束 TimeTableConstraintProvider::roomConflict 在同一房间，且两节课中的课程具有相同的 timeslot，具有匹配权重 1。因此，如果约束 weight 为 10 个 位，它将分数减少 -10hard。

流程

创建 src/test/java/org/acme/optaplanner/solver/TimeTableConstraintProviderTest.java 类：

```
package org.acme.optaplanner.solver;

import java.time.DayOfWeek;
import java.time.LocalTime;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import org.acme.optaplanner.domain.Lesson;
import org.acme.optaplanner.domain.Room;
import org.acme.optaplanner.domain.TimeTable;
import org.acme.optaplanner.domain.Timeslot;
import org.junit.jupiter.api.Test;
import org.optaplanner.test.api.score.stream.ConstraintVerifier;

@QuarkusTest
class TimeTableConstraintProviderTest {
```

```
    private static final Room ROOM = new Room("Room1");
    private static final Timeslot TIMESLOT1 = new Timeslot(DayOfWeek.MONDAY,
LocalTime.of(9,0), LocalTime.NOON);
    private static final Timeslot TIMESLOT2 = new Timeslot(DayOfWeek.TUESDAY,
LocalTime.of(9,0), LocalTime.NOON);

    @Inject
    ConstraintVerifier<TimeTableConstraintProvider, TimeTable> constraintVerifier;

    @Test
    void roomConflict() {
        Lesson firstLesson = new Lesson(1, "Subject1", "Teacher1", "Group1");
        Lesson conflictingLesson = new Lesson(2, "Subject2", "Teacher2", "Group2");
        Lesson nonConflictingLesson = new Lesson(3, "Subject3", "Teacher3", "Group3");

        firstLesson.setRoom(ROOM);
        firstLesson.setTimeslot(TIMESLOT1);

        conflictingLesson.setRoom(ROOM);
        conflictingLesson.setTimeslot(TIMESLOT1);

        nonConflictingLesson.setRoom(ROOM);
        nonConflictingLesson.setTimeslot(TIMESLOT2);

        constraintVerifier.verifyThat(TimeTableConstraintProvider::roomConflict)
                .given(firstLesson, conflictingLesson, nonConflictingLesson)
                .penalizesBy(1);
    }

}
```

*请注意，**ConstraintVerifier** 在测试过程中如何忽略约束权重，即使这些约束 **weight** 在 **ConstraintProvider** 中是硬编码的。这是因为约束权重在进行生产环境前定期有所改变。这样，约束加权调整不会破坏单元测试。*

13.7.2. 测试 school timetable solver

*这个示例在 **Red Hat build of Quarkus** 上测试红帽构建的 **OptaPlanner school timetable** 项目。它使用 **JUnit** 测试来生成测试数据集，并将其发送到 **TimeTableController** 以解决。*

流程

1. *使用以下内容创建 **src/test/java/com/example/rest/TimeTableResourceTest.java** 类：*

```
package com.exmaple.optaplanner.rest;

import java.time.DayOfWeek;
import java.time.LocalTime;
import java.util.ArrayList;
```

```
import java.util.List;

import javax.inject.Inject;

import io.quarkus.test.junit.QuarkusTest;
import com.exmaple.optaplanner.domain.Room;
import com.exmaple.optaplanner.domain.Timeslot;
import com.exmaple.optaplanner.domain.Lesson;
import com.exmaple.optaplanner.domain.TimeTable;
import com.exmaple.optaplanner.rest.TimeTableResource;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Timeout;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertNotNull;
import static org.junit.jupiter.api.Assertions.assertTrue;

@QuarkusTest
public class TimeTableResourceTest {

    @Inject
    TimeTableResource timeTableResource;

    @Test
    @Timeout(600_000)
    public void solve() {
        TimeTable problem = generateProblem();
        TimeTable solution = timeTableResource.solve(problem);
        assertFalse(solution.getLessonList().isEmpty());
        for (Lesson lesson : solution.getLessonList()) {
            assertNotNull(lesson.getTimeslot());
            assertNotNull(lesson.getRoom());
        }
        assertTrue(solution.getScore().isFeasible());
    }

    private TimeTable generateProblem() {
        List<Timeslot> timeslotList = new ArrayList<>();
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(8, 30),
LocalTime.of(9, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(9, 30),
LocalTime.of(10, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(10, 30),
LocalTime.of(11, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(13, 30),
LocalTime.of(14, 30)));
        timeslotList.add(new Timeslot(DayOfWeek.MONDAY, LocalTime.of(14, 30),
LocalTime.of(15, 30)));

        List<Room> roomList = new ArrayList<>();
        roomList.add(new Room("Room A"));
        roomList.add(new Room("Room B"));
        roomList.add(new Room("Room C"));

        List<Lesson> lessonList = new ArrayList<>();
        lessonList.add(new Lesson(101L, "Math", "B. May", "9th grade"));
```

```
        lessonList.add(new Lesson(102L, "Physics", "M. Curie", "9th grade"));
        lessonList.add(new Lesson(103L, "Geography", "M. Polo", "9th grade"));
        lessonList.add(new Lesson(104L, "English", "I. Jones", "9th grade"));
        lessonList.add(new Lesson(105L, "Spanish", "P. Cruz", "9th grade"));

        lessonList.add(new Lesson(201L, "Math", "B. May", "10th grade"));
        lessonList.add(new Lesson(202L, "Chemistry", "M. Curie", "10th grade"));
        lessonList.add(new Lesson(203L, "History", "I. Jones", "10th grade"));
        lessonList.add(new Lesson(204L, "English", "P. Cruz", "10th grade"));
        lessonList.add(new Lesson(205L, "French", "M. Curie", "10th grade"));
        return new TimeTable(timeslotList, roomList, lessonList);
    }

}
```

此测试会验证在解决后，所有课程都会分配给一个时间窗和房间。它还会验证它是否发现可行的解决方案（没有硬限制）。

2. 在 **src/main/resources/application.properties** 文件中添加 **test** 属性：

```
# The solver runs only for 5 seconds to avoid a HTTP timeout in this simple implementation.
# It's recommended to run for at least 5 minutes ("5m") otherwise.
quarkus.optaplanner.solver.termination.spent-limit=5s

# Effectively disable this termination in favor of the best-score-limit
%test.quarkus.optaplanner.solver.termination.spent-limit=1h
%test.quarkus.optaplanner.solver.termination.best-score-limit=0hard/*soft
```

通常，该解决方法在 **200** 毫秒内找到可行的解决方案。注意 **application.properties** 文件如何在测试过程中覆盖解决程序终止，以便在找到可行的解决方案 **(0hard/*soft)** 时立即终止。这可避免硬编码解决程序时间，因为单元测试可能会在任意硬件上运行。这种方法可确保测试用时足够长来查找可行的解决方案，即使在较慢的系统上也是如此。但是，即使在快速系统中，它也无法运行时间比严格要长的时间更长。

13.8. 日志记录

完成 **OptaPlanner school** 时间项目的红帽构建后，您可以使用日志信息帮助您微调 **ConstraintProvider** 中的限制。查看 **info** 日志文件中的分数计算速度，以评估对您的限制更改的影响。以 **debug** 模式运行应用程序，显示应用程序接受的每个步骤，或使用 **trace** 日志记录记录每一步和每次移动。

流程

1. 运行 **school timetable** 应用以获得固定时间，如五分钟。

2. 查看 日志文件中的 分数计算速度，如下例所示：

```
... Solving ended: ..., score calculation speed (29455/sec), ...
```

3. 更改约束，再次运行计划应用程序相同的时间，并查看 日志文件中 记录的分数计算速度。

4. 以 **debug** 模式运行应用程序，以记录应用程序所做的每个步骤：

- 要从命令行运行调试模式，请使用 **-D** 系统属性。

- 要永久启用调试模式，请在 **application.properties** 文件中添加以下行：

```
quarkus.log.category."org.optaplanner".level=debug
```

以下示例显示了在 **debug** 模式下 日志文件 的输出：

```
... Solving started: time spent (67), best score (-20init/0hard/0soft), environment mode
(REPRODUCIBLE), random (JDK with seed 0).
...     CH step (0), time spent (128), score (-18init/0hard/0soft), selected move count (15),
picked move ([Math(101) {null -> Room A}, Math(101) {null -> MONDAY 08:30}]).
...     CH step (1), time spent (145), score (-16init/0hard/0soft), selected move count (15),
picked move ([Physics(102) {null -> Room A}, Physics(102) {null -> MONDAY 09:30}]).
...
```

5. 使用 **trace logging** 显示每个步骤以及每个步骤的每一个移动。

13.9. 使用 MICROMETER 和 PROMETHEUS 来监控 SCHOOL TIMETABLE OPTAPLANNER JAVA 应用程序

OptaPlanner 通过 **Micrometer** 公开指标数据，这是 **Java** 应用程序的指标检测库。您可以将 **Micrometer** 与 **Prometheus** 搭配使用以监控 **school timetable** 应用程序中 **OptaPlanner solver**。

先决条件

- 您已使用 **Java** 创建 **OptaPlanner school timetable** 应用程序。

- 已安装 **Prometheus**。有关安装 **Prometheus** 的详情，请查看 **Prometheus** 网站。

流程

1. 将 **Micrometer Prometheus** 依赖项添加到 **school timetable pom.xml** 文件中，其中 **<MICROMETER_VERSION >** 是您安装的 **Micrometer** 的版本：

```
<dependency>
 <groupId>io.micrometer</groupId>
 <artifactId>micrometer-registry-prometheus</artifactId>
 <version><MICROMETER_VERSION></version>
</dependency>
```

注意

还需要 **micrometer-core** 依赖项。但是，这个依赖项包含在 **optaplanner-core** 依赖项中，您不需要将其添加到 **pom.xml** 文件中。

2. 将以下导入语句添加到 **TimeTableApp.java** 类中。

```
import io.micrometer.core.instrument.Metrics;
import io.micrometer.prometheus.PrometheusConfig;
import io.micrometer.prometheus.PrometheusMeterRegistry;
```

3. 将下面几行添加到 **TimeTableApp.java** 类的主方法的顶部，以便 **Prometheus** 可以从 **com.sun.net.net.httpserver.HttpServer** 中获取数据：

```
PrometheusMeterRegistry prometheusRegistry = new
PrometheusMeterRegistry(PrometheusConfig.DEFAULT);

    try {
        HttpServer server = HttpServer.create(new InetSocketAddress(8080), 0);
        server.createContext("/prometheus", httpExchange -> {
            String response = prometheusRegistry.scrape();
            httpExchange.sendResponseHeaders(200, response.getBytes().length);
            try (OutputStream os = httpExchange.getResponseBody()) {
                os.write(response.getBytes());
            }
        });

        new Thread(server::start).start();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
```

```
        Metrics.addRegistry(prometheusRegistry);

        solve();
    }
```

4. 添加以下行，以控制解决时间。通过调整解决时间，您可以了解指标如何根据要解决的时间而变化。

```
withTerminationSpentLimit(Duration.ofMinutes(5)));
```

5. 启动 **school timetable** 应用。

6. 在 **Web** 浏览器中打开 **http://localhost:8080/prometheus**，查看 **Prometheus** 中的可时间应用程序。

7. 打开监控系统，查看您的 **OptaPlanner** 项目的指标。

   公开以下指标：

   - **optaplanner_solver_errors_total** ：从测量开始解决时出现的错误总数。
   - **optaplanner_solve_duration_seconds_active_count** ：当前解决的解决方法数量。
   - **optaplanner_solve_duration_seconds_max: run time of longest-running currently active solver.**
   - **optaplanner_solve_duration_seconds_duration_sum:**每个活动解决持续时间的总和。例如，如果存在两个活跃的 **solvers**，一个只运行三分钟，另一个为一分钟，总计解决时间为 **4** 分钟。

部分 V. RED HAT BUILD OF OPTAPLANNER STARTER 应用程序

Red Hat build of OptaPlanner 提供了以下入门应用程序，您可以在 Red Hat OpenShift Container Platform 上部署：

- staff staff Rostering starter 应用程序
- Vechile Route Planning starter 应用程序

OptaPlanner 启动程序应用程序比示例和快速启动指南的开发更多。它们专注于特定的用例，并使用可用于构建规划解决方案的最佳技术。

第 14 章 在 IDE 中使用 RED HAT BUILD OF OPTAPLANNER: 一个员工的 ROSTERING 示例

作为业务规则开发人员，您可以使用 IDE 来构建、运行和修改 optaweb-employee-rostering starter 应用程序，使用 Red Hat build of OptaPlanner 功能。

先决条件

- 您可以使用集成的开发环境，如 Red Hat CodeReady Studio 或 IntelliJ IDEA。
- 您对 Java 语言有了了解。
- 您对 React 和 TypeScript 的了解。需要此要求才能开发 OptaWeb UI。

14.1. 员工指定入门应用程序概述

员工名单的入门应用程序为员工分配组织内各种位置的员工。例如，您可以使用应用程序在 nurses、guard duty 跨多个位置转移，或在 worker 之间的装配行转移。

最佳员工名单必须考虑很多变量。例如，不同的位置上会需要不同的技能来改变。另外，某些员工对于某些时间段不可用，或者可能首选特定时间段。此外，员工也可以有合同，该合同限制员工可在单个时间段内工作的小时数。

对于这个启动程序应用程序，红帽构建的 OptaPlanner 规则都使用 hard 和 soft 约束。在优化过程中，计划引擎可能不会违反硬约束，例如，如果员工不可用（开单位置），或者某个员工无法在单一转移中的两个位置工作。计划引擎会尝试遵守软限制，例如员工首选项不做特定的转变，但如果最佳解决方案需要，则可能会违反它们。

14.2. 构建并运行员工名语入门应用程序

您可以从源代码构建员工级名入门应用程序，并将其作为 JAR 文件运行。

另外，您可以使用 IDE，如 Eclipse（包括 Red Hat CodeReady Studio）来构建和运行应用程序。

14.2.1. 准备部署文件

在构建和部署应用程序前，您必须下载并准备部署文件。

流程

1. 进入红帽客户门户网站中的 **Software Downloads** 页面（需要登录），然后从下拉列表中选择产品和版本：
   - 产品： 流程自动化管理器
   - 版本： **7.13.4**
2. 下载 **Red Hat Process Automation Manager 7.13.4 Kogito 和 OptaPlanner 8 Decision Services Quickstarts (rhpam-7.13.4-kogito-and-optaplanner-quickstarts.zip)**。
3. 提取 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts.zip** 文件。
4. 下载 **Red Hat Process Automation Manager 7.13 Maven Repository Kogito 和 OptaPlanner 8 Maven** 存储库 **(rhpam-7.13.4-kogito-maven-repository.zip)**。
5. 提取 **rhpam-7.13.4-kogito-maven-repository.zip** 文件。
6. 将 **rhpam-7.13.4-kogito-maven-repository/maven-repository** 子目录的内容复制到 **~/.m2/repository** 目录中。
7. 导航到 **optaweb-8.13.0.Final-redhat-00013/optaweb-employee-rostering** 目录。这个文件夹是本文档的后续部分中的基础文件夹。

注意

文件和文件夹名称可能比本文档中特别记录的版本号更高。

14.2.2. 运行 Employee Rostering starter application JAR 文件

*您可以从 **Red Hat Process Automation Manager 7.13.4 Kogito 和 OptaPlanner 8 Decision Services Quickstarts** 中包含的 **JAR** 文件运行 **Employee Rostering starter** 应用程序。*

先决条件

- *您已下载并提取 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts.zip** 文件，如 **第 14.2.1 节 “准备部署文件”** 所述。*
- *安装了 **Java Development Kit**。*
- *已安装 **Maven**。*
- *主机可以访问互联网。构建过程使用互联网从外部存储库下载 **Maven** 软件包。*

流程

1. *在命令终端中，更改为 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-employee-rostering** 目录。*
2. *使用以下命令：*

```
mvn clean install -DskipTests
```

3. *等待构建过程完成。*
4. *导航到 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-employee-rostering/optaweb-employee-rostering-standalone/target** 目录。*
5. *输入以下命令来运行 **Employee Rostering JAR** 文件：*

```
java -jar quarkus-app/quarkus-run.jar
```

注意

在构建时，**quarkus.datasource.db-kind** 参数的值默认设置为 **H2**。要使用不同的数据库，您必须重建独立模块，并在命令行中指定数据库类型。例如，要使用 **PostgreSQL** 数据库，请输入以下命令：

```
mvn clean install -DskipTests -Dquarkus.profile=postgres
```

6. 要访问应用程序，在网页浏览器中输入 **http://localhost:8080/**。

14.2.3. 使用 Maven 构建并运行 Employee Rostering starter 应用

您可以使用命令行构建并运行员工名单的入门应用程序。

如果您使用此步骤，数据将保存在内存中，并在服务器停止后丢失。要使用持久性存储的数据库服务器构建并运行应用程序，请参阅 第 14.2.4 节 “从命令行构建并运行带有持久数据存储的员工级入门应用程序”。

先决条件

- 您已准备了部署文件，如 第 14.2.1 节 “准备部署文件” 所述。
- 安装了 **Java Development Kit**。
- 已安装 **Maven**。
- 主机可以访问互联网。构建过程使用互联网从外部存储库下载 **Maven** 软件包。

流程

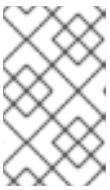
1. 进入 **optaweb-employee-rostering-backend** 目录。
2. 使用以下命令：

```
mvn quarkus:dev
```

3. 进入 **optaweb-employee-rostering-frontend** 目录。

4. 使用以下命令：

```
npm start
```



注意

如果使用 **npm** 启动服务器，则 **npm** 会监控代码更改。

5. 要访问应用程序，在网页浏览器中输入 **http://localhost:3000/**。

14.2.4. 从命令行构建并运行带有持久数据存储的员工级入门应用程序

如果使用命令行构建员工的入门应用程序并运行它，则可以为持久数据存储提供数据库服务器。

先决条件

- 您已准备了部署文件，如 第 14.2.1 节 “准备部署文件” 所述。
- 安装了 **Java Development Kit**。
- 已安装 **Maven**。
- 主机可以访问互联网。构建过程使用互联网从外部存储库下载 **Maven** 软件包。
- 您已部署 **MySQL** 或 **PostrgeSQL** 数据库服务器。

流程

1. 在命令终端中，导航到 **optaweb-employee-rostering-standalone/target** 目录。

2. 输入以下命令来运行 **Employee Rostering JAR** 文件：

```
java \
-Dquarkus.datasource.username=<DATABASE_USER> \
-Dquarkus.datasource.password=<DATABASE_PASSWORD> \
-Dquarkus.datasource.jdbc.url=<DATABASE_URL> \
-jar quarkus-app/quarkus-run.jar
```

在本例中，替换以下占位符：

- **<DATABASE_URL >：**要连接到数据库的 **URL**
- **<DATABASE_USER >** ：要连接到数据库的用户
- **<DATABASE_PASSWORD> : < DATABASE_USER>**的密码

注意

在构建时，**quarkus.datasource.db-kind** 参数的值默认设置为 **H2**。要使用不同的数据库，您必须重建独立模块，并在命令行中指定数据库类型。例如，要使用 **PostgreSQL** 数据库，请输入以下命令：

```
mvn clean install -DskipTests -Dquarkus.profile=postgres
```

14.2.5. 使用 IntelliJ IDEA 构建并运行员工名单入门应用程序

您可以使用 **IntelliJ IDEA** 构建并运行员工间候入门应用程序。

先决条件

- 您已下载了 **Employee Rostering** 源代码，它位于 **Employee Rostering GitHub** 页面中。

- 已安装 **Intellij IDEA**、**Maven** 和 **Node.js**。
- 主机可以访问互联网。构建过程使用互联网从外部存储库下载 **Maven** 软件包。

流程

1. **Start IntelliJ IDEA.**
2. 在 **IntelliJ IDEA** 主菜单中选择 **File** → **Open**。
3. 选择应用程序源的根目录并点 **OK**。
4. 在主菜单中选择 **Run** → **Edit Configuration**。
5. 在出现的窗口中，展开 **Templates** 并选择 **Maven**。此时会出现 **Maven** 侧栏。
6. 在 **Maven** 边栏中，从 **Working Directory** 菜单中选择 **optaweb-employee-rostering-backend**。
7. 在命令行中输入 **mvn quarkus:dev**。
8. 要启动后端，请单击 **OK**。
9. 在命令终端中，导航到 **optaweb-employee-rostering-frontend** 目录。
10. 输入以下命令启动前端：

```
npm start
```

11. 要访问应用程序，在网页浏览器中输入 **http://localhost:3000/**。

14.3. 员工指定入门应用程序的源代码概述

员工的入门应用程序由以下主要组件组成：

- 使用红帽构建的 **OptaPlanner** 并提供 **REST API**，实现 **rostering** 逻辑 的后端
- 使用 **React** 实施用户界面并通过 **REST API** 与 **backend** 模块交互的 **frontend** 模块

您可以独立构建和使用这些组件。特别是，您可以实施不同的用户界面，并使用 **REST API** 调用服务器。

除两个主要组件外，员工的 **rostering** 模板还包含随机源数据生成器（用于演示和测试目的）和基准测试应用程序。

模块和密钥类

staff rostering 模板的 **Java** 源代码包含多个 **Maven** 模块。这些模块各自包含一个单独的 **Maven** 项目文件(**pom.xml**)，但它们旨在在通用项目中进行构建。

模块包含多个文件，包括 **Java** 类。本文档列出了所有模块，以及包含员工计算计算关键信息的类和其他文件。

- **optaweb-employee-rostering-benchmark** 模块：包含生成随机数据并基准解决方案的额外应用程序。
- **optaweb-employee-rostering-distribution** 模块：包含 **README** 文件。
- **optaweb-employee-rostering-docs** 模块：包含文档文件。
- **optaweb-employee-rostering-frontend** 模块：使用在 **React** 中开发的用户界面包含客户端应用程序。
- **optaweb-employee-rostering-backend** 模块：包含使用 **OptaPlanner** 执行恶意计算的服务器应用程序。

- ***src/main/java/org.optaweb.employeerostering.service.rosterGenerator.java:*** *为演示和测试目的生成随机输入数据。如果您更改了所需的输入数据，请相应地更改生成器。*

- ***src/main/java/org.optaweb.employeerostering.domain.employee/EmployeeAvailability.java*** *：定义员工的可用性信息。对于每个时间插槽，员工都可以不可用，或者指定该员工的首选时间段。*

- ***src/main/java/org.optaweb.employeerostering.domain.employee/Employee.java*** *：定义员工。员工的名称、技术技能列表和合同工作。技能由技能对象表示。*

- ***src/main/java/org.optaweb.employeerostering.domain.roster/Roster.java:*** *定义计算的* ***rostering*** *信息。*

- ***src/main/java/org.optaweb.employeerostering.domain.shift/Shift.java*** *：定义可为其分配员工的转换。转换由一个时间窗和* ***spot*** *定义。例如，在外围的位置可能会发生* ***20 8AM-4PM*** *时间段* ***2*** *月* ***20 8AM-4PM*** *的位置。可以为特定的位置和时间插槽定义多个转换。在这种情况下，这个位置和时间插槽需要多个员工。*

- ***src/main/java/org.optaweb.employeeroeerostering.domain.skill/Skill.java*** *：定义员工可以拥有的技术。*

- ***src/main/java/org.optaweb.employeerostering.domain.spot/Spot.java*** *：定义可放置员工的位置。例如，* ***Kitchen*** *可以是* ***spot****。*

- ***src/main/java/org.optaweb.employeeroeerostering.domain.contract.java:*** *定义在各种时间段内为员工设置工作时间的合同。*

- ***src/main/java/org.optaweb.employeerostering.domain.tenant/Tenant.java:*** ***Defines a tenant.****每个租户代表一组独立的数据。个租户的数据更改不会影响任何其他租户。*

- ****view.java:*** *与域对象相关的类定义从其他信息计算的值集；客户端应用可以通过* ***REST API*** *读取这些值，但不写入它们。*

- ***service.java :**位于定义 **REST API** 的服务 软件包中的接口。服务器和客户端应用都单独定义对这些接口的实施。

- ***optaweb-employee-rostering-standalone** 模块：包含独立应用程序的装配配置。

14.4. 修改 EMPLOYEES ROSTERING STARTER 应用

*要修改 **employees rostering starter** 应用以满足您的需要，您必须更改管理优化过程的规则。您还必须确保数据结构包含所需的数据，并提供规则所需的计算结果。如果用户界面中不存在所需数据，还必须修改用户界面。*

以下过程概述了修改员工的入门应用的一般方法。

先决条件

- *您有一个成功构建应用程序的构建环境。*

- *您可以阅读和修改 **Java** 代码。*

流程

1. *计划所需的更改。回答以下问题：*

   - *必须避免 的其他场景是什么？这些场景是 硬性限制。*

   - *优化程序必须尽量避免 的其它 场景是什么？这些场景是 软限制。*

   - *如果每个方案发生在潜在解决方案中，则需要什么数据计算？*

   - *可以从用户在现有版本中输入的信息派生哪些数据？*

   - *哪些数据可以被硬编码？*

- *哪个数据必须由用户输入，且在当前版本中不会输入？*

2. *如果可以从当前数据计算任何所需数据或硬编码，请将计算或硬编码添加到现有的视图或实用程序类。如果数据必须在服务器端计算，请添加 **REST API** 端点来读取它。*

3. *如果用户必须输入所需的数据，将数据添加到代表数据实体（例如 **Employee** 类）的类中，添加 **REST API** 端点来读取和写入数据，并修改用户界面进入数据。*

4. *当所有数据都可用时，修改规则。对于大多数修改，您必须添加新的规则。规则位于 **optaweb-employee-rostering-backend** 模块的 **src/main/java/optaweb/employeerostering/service/solver/EmployeeRosteringConstraintProvider.java** 文件中。*

5. *修改应用程序后，构建并运行它。*

第 15 章 在 RED HAT OPENSHIFT CONTAINER PLATFORM 中部署和使用红帽构建的 OPTAPLANNER：一个员工的先备入门示例

作为业务规则开发人员，您可以快速部署 **Red Hat Process Automation Manager** 发行版本中包含的 **optaweb-employee-rostering starter** 项目，从而测试与红帽构建的 **OptaPlanner** 功能进行交互。

先决条件

- 您可以访问部署的 **OpenShift** 环境。详情请参阅您使用的 **OpenShift** 产品的文档。

15.1. 员工指定入门应用程序概述

员工名单的入门应用程序为员工分配组织内各种位置的员工。例如，您可以使用应用程序在 **nurses**、**guard duty** 跨多个位置转移，或在 **worker** 之间的装配行转移。

最佳员工名单必须考虑很多变量。例如，不同的位置上会需要不同的技能来改变。另外，某些员工对于某些时间段不可用，或者可能首选特定时间段。此外，员工也可以有合同，该合同限制员工可在单个时间段内工作的小时数。

对于这个启动程序应用程序，红帽构建的 **OptaPlanner** 规则都使用 **hard** 和 **soft** 约束。在优化过程中，计划引擎可能不会违反硬约束，例如，如果员工不可用（开单位置），或者某个员工无法在单一转移中的两个位置工作。计划引擎会尝试遵守软限制，例如员工首选项不做特定的转变，但如果最佳解决方案需要，则可能会违反它们。

15.2. 在 OPENSHIFT 上安装并启动员工间候入门应用程序

使用 **runOnOpenShift.sh** 脚本将 **Employee Rostering** 入门程序应用程序部署到 **Red Hat OpenShift Container Platform**。**runOnOpenShift.sh shell** 脚本在 **Red Hat Process Automation Manager 7.13.4 Kogito** 和 **OptaPlanner 8 Decision Services Quickstarts** 发行版本中提供。

runOnOpenShift.sh 脚本在本地构建并打包应用程序源代码，并将它上传到 **OpenShift** 环境以进行部署。这个方法需要 **Java Development Kit**、**Apache Maven** 和 **bash shell** 命令行。

15.2.1. 使用提供的脚本部署应用程序

您可以使用提供的脚本将 **Employee Rostering starter** 应用程序部署到 **Red Hat OpenShift Container Platform**。该脚本在本地构建并打包应用源代码，并将它上传到 **OpenShift** 环境以进行部

署。

先决条件

- 您可以使用 **oc** 命令行工具登录到目标 **OpenShift** 环境。有关此工具的更多信息，请参阅 **OpenShift Container Platform CLI** 参考。
- 安装了 **OpenJDK 11** 或更高版本。红帽构建的 **Open JDK** 可从红帽客户门户网站的 **Software Downloads** 页面获取（需要登录）。
- 已安装 **Apache Maven 3.6** 或更高版本。**Maven** 可以从 **Apache Maven Project** 网站获得。
- 您的本地系统中有 **bash shell** 环境。

流程

1. 进入红帽客户门户网站中的 **Software Downloads** 页面（需要登录），然后从下拉列表中选择产品和版本：
   - 产品：流程自动化管理器
   - 版本：**7.13.4**
2. 下载 **Red Hat Process Automation Manager 7.13 Maven Repository Kogito** 和 **OptaPlanner 8 Maven** 存储库 **(rhpam-7.13.4-kogito-maven-repository.zip)**文件。
3. 提取 **rhpam-7.13.4-kogito-maven-repository.zip** 文件。
4. 将 **rhpam-7.13.4-kogito-maven-repository/maven-repository** 子目录的内容复制到 **~/.m2/repository** 目录中。
5. 从红帽客户门户的软件 下载页面，下载 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts.zip** 文件。

6. 提取下载的存档。

7. 进入 **optaweb-employee-rostering** 文件夹。

8. 要构建 **Employee Rostering** 应用程序，请运行以下命令：

```
mvn clean install -DskipTests -DskipITs
```

9. 登录 **OpenShift** 帐户或 **Red Hat Code Ready Container** 实例。在以下示例中，**< account-url>** 是 **OpenShift** 帐户或 **Red Hat Code Ready Container** 实例的 **URL**，**& lt;login-token >** 是该帐户的登录令牌：

```
oc login <account-url> --token <login-token>
```

10. 创建一个新项目来托管 **Employee Rostering**：

```
oc new-project optaweb-employee-rostering
```

11. 运行置备脚本以构建和部署应用程序：

```
./runOnOpenShift.sh
```

完成编译和打包可能需要长达 **10** 分钟。这些进程在命令行输出中持续显示进度。

当操作完成后，会显示以下信息，其中 **< URL& gt;** 是部署的 **URL:**

```
You can access the application at <URL> once the deployment is done.
```

12. 为 **OpenShift** 帐户或 **Red Hat Code Ready Container** 实例输入您之前在流程中使用的 **URL**，以访问已部署的应用程序。首次启动最多可能需要一分钟，因为其他构建在 **OpenShift** 平台上完成。

注意

如果应用程序点击链接后没有打开一分钟，请对浏览器页面执行硬刷新。

15.3. 使用员工指定入门应用程序

您可以使用 **Web** 界面使用 **Employee Rostering** 入门应用程序。接口在 **ReactJS** 中开发。您还可以根据需要访问 **REST API** 来创建自定义用户界面。

15.3.1. 草稿和发布周期

在任何特定时刻，您可以使用应用程序为一个时间段内创建 **roster**，称为 草案 周期。默认情况下，草案周期的长度为 **3** 周。

截止到草案前一周的 **roster** 后，您可以 发布 **roster**。目前，当前期第一周的名册会成为 已发布的 期间。在已发布的时间内，**roster** 已被修复，您无法自动修改它（方式，紧急手动更改仍有可能）。然后，这个 **roster** 可以分配给员工，以便他们能够规划他们周围的时间。草案日期稍后会转移一个星期。

例如，假设设置了 **9** 月 **1** 日至 **9** 月 **21** 日的草案。您可以在此期限内自动创建员工名册。然后，当您发布 **roster** 时，周期最多会发布 **7** 年 **9** 月 **7** 日。新的草案周期为 **9** 月 **8** 日至 **9** 月 **28** 日。

有关发布 **roster** 的步骤，请参考 第 **15.3.12** 节 “发布动向语”。

15.3.2. 轮转模式

员工名单应用程序支持转换和员工的 轮转模式。

轮转模式是从两个天开始的任何时间的**"model"**周期。模式没有与特定日期绑定。

您可以为轮转每天创建时间存储桶。每次存储桶都设置变化的时间。（可选）模板可以包括转换的默认员工的名称。

当您发布 **roster** 时，应用程序会在草案中添加新周。此时，转移，如果适用，则默认员工名称从轮转模式复制到草案期间的新部分。

当达到轮转模式结束时，它会自动从开始重启。

如果您的组织中的周末切换模式与工作日变化模式不同，请使用一周的轮转模式或数周的轮转模式，例如 **14**、**21** 或 **28** 天。默认长度为 **28** 天。然后，模式总是在同一工作日重复，您可以为不同的工作日设置转换。

有关编辑轮转模式的步骤，请参阅 第 **15.3.13** 节 “查看并编辑轮转模式”。

15.3.3. employees Rostering 租户

Employee Rostering 应用程序支持多个 租户。每个租户都是一个独立的数据，包括输入和 **roster** 输出。更改一个租户的数据不会影响其他租户。您可以在租户间切换以使用多个独立的数据集，例如，为员工准备不同的位置。

安装后会存在多个示例租户，这代表一些典型的企业类型，如工厂或医院等。您可以选择这些租户中的任何一个，并对其进行修改以符合您的需要。您还可以创建新租户，以从空白处输入数据。

15.3.3.1. 更改一个 Employee Rostering 租户

您可以更改当前的租户。选择了不同的租户后，所有显示的信息都引用此租户，您进行的任何更改仅会影响到此租户。

流程

1. 在 **Employee Rostering application Web** 界面中，在浏览器窗口的右上角，单击 **Tenant** 列表。
2. 从列表中选择一个租户。

15.3.3.2. 创建租户

您可以创建新租户，从空白处输入数据。在创建租户时，您可以设置几个参数来确定应用如何为这个租户准备输出。

重要

在创建租户后，您无法更改租户参数。

流程

1. 要在 **Employee Rostering** 应用 **Web** 界面中创建一个新租户，请在浏览器窗口右上角单击设置**(gear)**图标，然后单击添加。

2. 设置以下值：

- 名称 ：新租户的名称。此名称显示在租户列表中。
- 计划开始日期 ：初始草案时间段的开始日期。发布 **roster** 后，此日期将变为已发布期限的开始日期。此日期的工作日始终保持在一周日，这些日期开始周期草案，任何特定公布的期限，以及首次使用轮转模式。因此，通常很方便地将开始日期设置为一周（星期一或周一）的开始。
- 草案长度（天 ）：周期的长度。草案周期在租户生命周期内保持相同的长度。
- **publish Notice(days)** ：发布通知周期的长度。我们希望提前为任何一天发布最终的名册，因此员工有足够的通知规划其发生时间的个人生命周期。在当前版本中，此设置不会以任何方式强制执行。
- 发布长度（天 ）：您每次发布出 **roster** 时都会发布（修复）的周期长度。在当前版本中，此设置在 **7** 天时被修复。
- 轮转长度（天）**:** 轮转模式长度。
- **timezone** ： **roster** 应用到的环境的时区。此时区用于决定显示用户界面的**"**当前**"**日期。

3. 点击 **Save**。

租户使用空白数据创建。

15.3.4. 掌握技能

您可设置在 **roster** 中的任何位置需要的所有 技能。例如，除了一般人力资源和其他人操作等技能外，**24** 小时的工员可能需要中断、服务、总线和托管技能。

流程

1. 在 **Employee Rostering application Web** 界面中，单击 **Skills** 选项卡。

   您可以看到浏览器窗口右上角的当前可见技能的数量，例如，**1- 15** 代表 **34**。您可以使用 **&lt ;** 和 **& gt;** 按钮来显示列表中的其他技能。

   您可以在搜索框中输入技能名称以 搜索 技能。

2. 完成以下步骤以添加新技能：

   a. 点击 **Add**。

   b. 在名称 处在文本字段中输入新技能的名称。

   c. 点 **Save** 图标。

3. 要编辑技能名称，请点击 技能旁边的编辑 技巧图标（铅笔图标）。

4. 要删除技能，请单击技能旁边的 删除技能 图标（垃圾形图标）。

注意

在每个租户中，技能名称必须是唯一的。如果技能与员工或点相关联，则您无法删除技能。

15.3.5. 输入位置

您必须进入 **spots** 列表， 这代表着业务中的各种位置。就职，点数包括该栏、总线站、前端计数器、各种工具包、服务站、服务区域和办公室。

对于每个位置，您可以从您在 **Skills** 选项卡输入的列表中选择一个或多个所需技能。应用程序 **rosters** 仅拥有该位置上所有必要技能的人员。如果位置没有所需技能，则应用程序会将任何员工降级为点。

流程

1. 要在 **Employee Rostering** 应用程序 **Web** 界面中输入或更改位置，点 **Spots** 选项卡。您可以在搜索框中输入任意部分来搜索 **spot**。

2. 完成以下步骤以添加新位置：

   a. 点 **Add Spot**。

   b. 在 名称 下面的文本字段中，输入新 **spot** 的名称。

   c. 可选：根据所需技能 集的下拉列表中选择一个或多个技能。

   d. 点 **Save** 图标。

3. 要编辑 **spot** 的名称和所需技能，点 **spot** 旁边的 **Edit Spot** 图标（铅笔图标）。

4. 要删除一个 **spot**，点 **spot** 旁边的 **Delete Spot** 图标**(trashcan shape)**。

注意

在每个租户中，位置名称必须是唯一的。当为其创建任何变化时，您无法删除该位置。

15.3.6. 输入合同列表

您必须进入业务为员工使用的所有合同类型列表。

合同决定了员工每天、日历、日历月或日历年可工作的最大时间。

在创建合同时，您可随意设置任何限制或任何限制。例如，一个部分员工可能不允许在一周内工作超过 **20** 小时，而一次性员工在一天和 **1800** 小时内可能限于 **10** 小时。其他合同可能不包括在工作时间内没有限制。

您必须在分钟内输入合同的所有工作时间限制。

流程

1. 要在 **Employee Rostering** 应用程序 **Web** 界面输入或更改 合同 列表，点 **contracts** 选项卡。

   您可以查看浏览器窗口右上角的当前可见合同的数量，例如，**1- 15** 代表 **34**。您可以使用 **&lt ;** 和 **& gt;** 按钮在列表中显示其他合同。

   您可以在搜索框中输入合同名称的任何部分来搜索合同。

2. 完成以下步骤以添加新合同：

   a. 点击 **Add**。

   b. 在名称 处文本字段中输入合同 名称。

   c. 在 **Maximum minutes** 下输入所需的时间限制：

   - 如果员工不能每天工作超过设定的时间，请在 **Per Day** 中启用复选框，然后在此复选框旁边的字段中输入分钟数。
   - 如果员工不能按日历星期工作，请在 **Per Week** 中启用复选框，然后在此复选框

旁边输入分钟数。

- 如果员工不能按每个日历月工作的间隔时间，请在 **Per Month** 中启用复选框，然后在此复选框旁边的字段输入分钟数。

- 如果员工不能以日历年为单位工作，请在日历年 内启用 复选框，然后在此复选框旁边的字段输入分钟数。

d. 点 **Save** 图标。

3. 要编辑合同的名称和时间限制，请点击合同名称旁边的 **Edit Contract** 图标（创建铅笔图标）。

4. 要删除合同，请点击 合同名称旁边的删除合同 图标（手形）。

注意

在每个租户中，合同名称必须是唯一的。如果合同已分配给任何员工，则您无法删除它。

15.3.7. 输入员工列表

您必须进入所有业务员工、拥有相关技能以及适用于他们的合约的合同。根据相关合同中的工作时间限制，应用程序认为这些员工能够发现其技能。

流程

1. 要在 **Employee Rostering** 应用程序 **Web** 界面中输入或更改员工列表，请单击 **Employees** 选项卡。

您可以看到浏览器窗口右上角的当前可见员工数量，例如，**1- 15** 代表 **34**。您可以使用 **&lt ;** 和 **& gt;** 按钮显示列表中的其他员工。

您可以在搜索框中输入员工名称的任何部分来搜索员工。

2. 完成以下步骤以添加新员工：

   a. 点击 **Add**。

   b. 在名称 下的文本字段中，输入员工 名称。

   c. 可选：在 **Skill** 集的下拉列表中选择一个或多个技能。

   d. 从合同下的下拉列表中选择 合同。

   e. 点 **Save** 图标。

3. 要编辑员工的名称和技能，请单击员工名称旁边的 **Edit Employee** 图标（列形）。

4. 要删除员工，请单击员工名称旁边的 删除 **Employee** 图标（垃圾形形）。

注意

在每个租户中，员工名称必须是唯一的。如果这些员工对任何变化的影响，则无法删除它们。

15.3.8. 设置员工可用性

您可以为特定时间范围设置员工可用性。

如果某个员工对于某个特定的时间范围 不可用，则在此时间段内，从不分配员工进行任何转换（例如，如果员工在精度或在度中调用 **sick** 或 监管），则不能分配给任何变化。不 所需 且需要的是特定时间范围内的员工首选项；应用程序尽可能适应它们。

流程

1. 要在 **Employee Rostering** 应用程序 **Web** 界面中查看并编辑员工可用性，请单击 **Availability Roster** 选项卡。

在窗口的左上角，您可以看到显示 **roster** 的日期。要查看其他周，您可以使用字段旁边的 **<** 和 **>** 按钮。或者，您可以单击日期字段并更改要查看包含此日期的每周的日期。

2. 要为员工创建可用性条目，请单击计划中的空空间，然后选择一个员工。最初，创建整个一天的 **Unavailable** 条目。

3. 要更改可用性条目，点该条目。您可以更改以下设置：

   - 从 和 到日期和时间：可用性条目应用到的时间范围。
   - **status**：您可以从下拉列表中选择 **Unavailable**、**Desired** 或 **Undesired** 状态。

   要保存该条目，请单击 **Apply**。

4. 要删除可用性条目，请单击 条目，然后单击删除 可用性。

您还可以通过将鼠标指针移到该条目上，然后点击该条目上显示的图标之一，以更改或删除可用性条目：

- 点击  图标将条目的状态设置为 **Unavailable**。
- 点击  图标将条目的状态设置为 **Undesired**。
- 点击  图标将条目的状态设置为 **Desired**。
- 点击 

图标删除该条目。

重要

如果员工已分配给更改，然后在这一转变过程中创建或更改可用性条目，则此分配不会自动更改。您必须再次创建员工转换 **roster**，以应用新的或更改的可用性条目。

15.3.9. 在 shift roster 中查看和编辑变化

Shift Roster 是所有位置以及所有可能的时间范围的列表。

如果一个员工在某个时间段内必须出现在 **spot** 中，就必须存在一个 转变。如果位置需要同时有多个员工，则可以为同一位置和时间范围创建多个转换。

每个转换都由位置（行）和时间范围（列）的交集表示。

当新的时间添加到草案周期中时，应用程序会从轮转模式将转换（和默认员工）从轮转模式复制到这一草案周期的新部分。您还可以在草稿时间内手动添加并编辑变化。

流程

1. 要查看并编辑 **Employee Rostering** 应用程序 **Web** 界面中的 **shift roster**，请单击 **Shift** 选项卡。

   在窗口的左上角，您可以看到显示 **roster** 的日期。要查看其他周，您可以使用字段旁边的 **<** 和 **>** 按钮。或者，您可以单击日期字段并更改要查看包含此日期的每周的日期。

2. 要添加转变，请点击计划的开放式区域。应用程序会添加一个变化，从点击的位置自动决定插槽和时间范围。

3. 要编辑转变，请点击转变。您可以为更改设置以下值：

   - 日期和时间 ：转换的确切时间和持续时间。

- 员工 ：分配给转变的员工。

- 固定 ：该员工是否 固定 为转换。如果员工固定，则自动员工名单无法改变员工的分配。固定员工不会自动复制到任何其他变化。

要保存更改，请单击" 应用 "。

4. 要删除转变，请单击转变，然后单击 **Delete shift**。

15.3.10. 创建和查看员工转换语

您可以使用应用程序为所有员工创建并查看最佳转变 **roster**。

流程

1. 要查看并编辑 **Employee Rostering** 应用程序 **Web** 界面中的 **shift roster**，请单击 **Shift** 选项卡。

2. 要创建最佳转变 **roster**，请单击 **Schedule**。应用程序需要 **30** 秒才能找到最佳解决方案。

结果

操作完成后，按 **Shift Roster** 视图包含最佳变速值。为草案周期创建一个新的 **roster**。该操作不会修改已发布的句点。

在窗口的左上角，您可以看到显示 **roster** 的日期。要查看其他周，您可以使用字段旁边的 **& lt ;** 和 **&gt ;** 按钮。或者，您可以单击日期字段并更改要查看包含此日期的每周的日期。

在草案期间，代表转换的框顺序是点线。在发布的时间段内，**Borders** 是不中断的行。

代表切换的方框的颜色显示每个切换的约束状态：

- 强绿色：**Soft** 约束匹配；例如，该转换在员工的"不良"时间内进行。

- ***Pale green***：没有约束中断。
- ***grey: Soft*** 约束问题；例如，该转换处于员工的"不所需"时间。
- 黄色：***Medium*** 约束中断，例如，没有员工被分配到转移。
- 红帽：硬盘约束损坏；例如，员工同时有两个变化。

15.3.11. 查看员工切换

您可以在以员工为中心的表中查看特定员工的分配转变。这些信息与 ***Shift Roster*** 相同，但查看格式可能会更方便地告知员工所分配转变。

流程

要在 ***Employee Rostering*** 应用程序 ***Web*** 界面中查看员工和转变表，请单击 ***Availability Roster*** 选项卡。

在窗口的左上角，您可以看到显示 ***roster*** 的日期。要查看其他周，您可以使用字段旁边的 ***& lt ;*** 和 ***&gt ;*** 按钮。或者，您可以单击日期字段并更改要查看包含此日期的每周的日期。

您可以查看浏览器窗口右上角的当前可见员工数量，例如，***1-10*** 为 ***34***。您可以使用数字旁边的 ***& lt;*** 和 ***>*** 按钮来显示列表中的其他员工。

在草案期间，代表转换的框顺序是点线。在发布的时间段内，***Borders*** 是不中断的行。

15.3.12. 发布动向语

当您发布转变的 ***roster*** 时，第一周的草稿期将发布。虽然紧急的手动更改仍有可能，但自动员工的指定时间不再会改变已发布的期限内任何变化。草案周期将在一周之后被转移。有关草案和发布周期的详情，请参考 第 ***15.3.1*** 节 “草稿和发布周期”。

流程

1. 要查看并编辑 ***Employee Rostering*** 应用程序 ***Web*** 界面中的 ***shift roster***，请单击 ***Shift*** 选

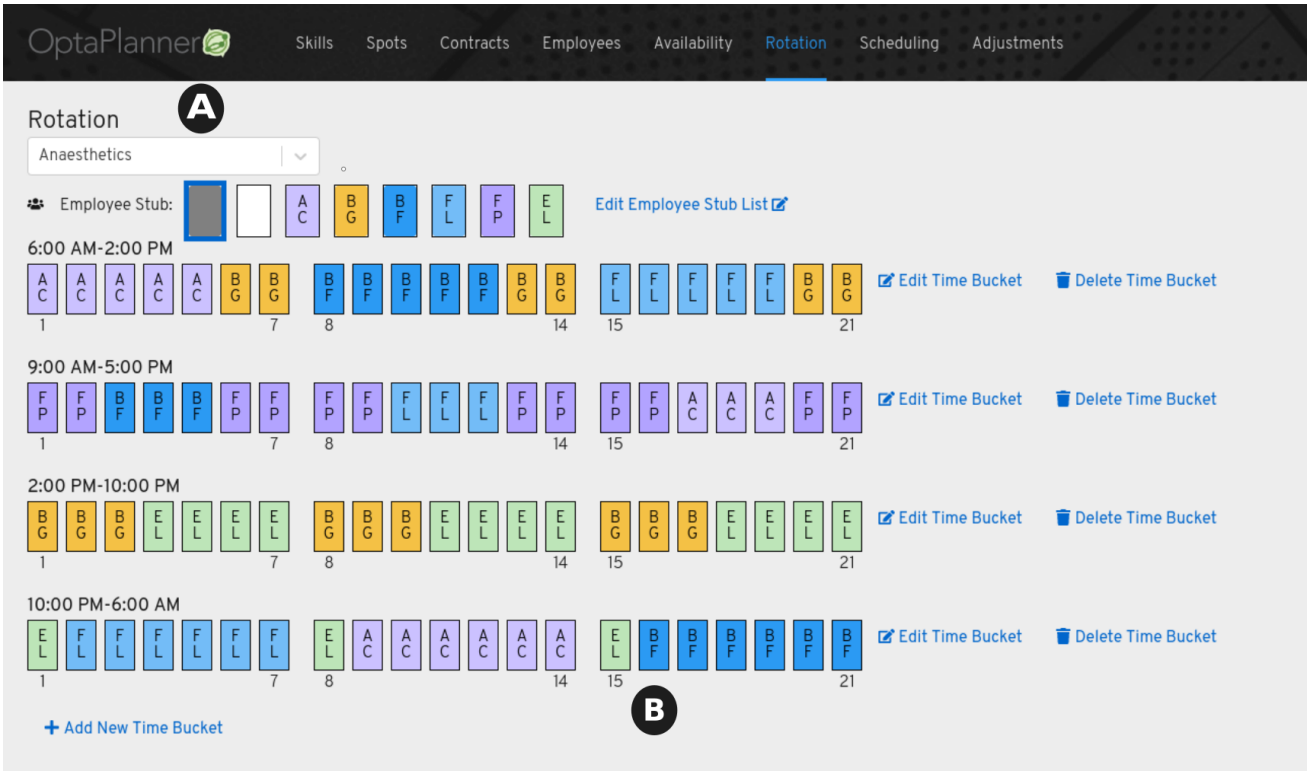
项卡。

2. 审查草案周期的第一周的转换代码，以确保它可以接受。

3. 单击 **Publish**。

15.3.13. 查看并编辑轮转模式

通过轮转模式，您可以添加、移动和删除转变，以便您有效地管理员工资源。它通过时间存储桶和座位进行定义。



- 时间存储桶描述了一个时间插槽（例如，**9:00 a.m.** 至 **5:00 p.m.**）用于特定位置或位置 （例如，**A**） （例如：**naestics**）超过两天，以及需要（例如，开票培训）所需的任何技能。

- 座位 **(B)** 是特定日期在特定时间 **bucket** 中的员工分配。

- 员工存根是一个图标，表示可用于分配给时间 **bucket** 的员工。员工存根列在 **Employee Stub** 列表中。

有关轮转模式的更多信息，请参阅 第 15.3.2 节 “轮转模式”。

流程

1. 点 **Rotation** 选项卡查看并编辑轮转模式。
2. 从 **Rotation** 菜单中选择 **spot**。
3. 点 **Add New Time Bucket**。此时会显示 **Creating Working Time Bucket** 对话框。
4. 指定开始和结束时间，选择任何额外的所需技能，选择本时间存储桶的天数，然后单击 **Save**。此时间 **bucket** 的未分配位置显示在 **Rotation** 页面中，按时间范围组织。
5. 要创建员工 **stub** 列表，以便您可以向轮转添加员工，单击 编辑 **Employee Stub** 列表。
6. 在 **Edit Employee Stub List** 对话框中，单击 **Add Employee**，并从列表中选择一个员工。
7. 添加此 **stub** 列表需要的所有员工，然后单击保存。员工在 **Rotation** 页面上显示时间存储桶之上。
8. 单击员工图标，从员工 **stub** 列表中选择员工。
9. 单击鼠标并拖动鼠标，将所选员工分配到座位。座位将用员工图标填充。

注意

一个时间存储桶只能为每天分配一个员工。要向同一时间存储桶添加多个员工，请复制时间存储桶，并根据需要更改员工名称。

10. 要置备计划，点 **Scheduling** 并选择您为其创建轮转的位置。

11. 单击 **Provision**，再指定日期范围。

12. 取消选择在此计划中包括的位置。

13. 单击所选 **spot** 旁边的箭头，然后取消选择希望在您的计划中使用的任何时间存储桶。

14. 单击 **Provision Shifts**。日历填充自时间存储桶生成的转换。

15. 要修改转变，请点击日历上的生成的转变。

第 16 章 部署和使用红帽构建的 OPTAPLANNER 载体路由规划入门应用程序

作为开发人员，您可以使用 OptaWeb Vehicle Routing starter 应用程序来优化您的载体发送。

先决条件

- 已安装了 OpenJDK(JDK)11。红帽构建的 Open JDK 可从红帽客户门户网站的 Software Downloads 页面获取（需要登录）。
- 已安装 Apache Maven 3.6 或更高版本。Maven 可以从 Apache Maven Project 网站获得。

16.1. 什么是 OPTAWEB VEHICLE 路由？

许多企业的主要目的是传输各种类型的车go。这些企业的目标是以最有效的方式从加载点向目标交付一个车点，并使用其载体车队。主要目标之一是尽量缩短以时间或距离度衡量的差差成本。

这种优化问题被称为载体路由问题(VRP)，并具有很多变体。

Red Hat build of OptaPlanner 可解决许多这些载体路由变化并提供解决方案示例。OptaPlanner 可让开发人员专注于对业务规则和需求建模，而不是学习 约束编程。OptaWeb Vehicle Routing 通过提供回答问题的入门应用程序，扩展了 OptaPlanner 的载路由功能：

- 我从哪里获得距离和出差时间？
- 如何视觉化 map 上的解决方案？
- 如何构建在云中运行的应用程序？

OptaWeb Vehicle Routing 使用 OpenStreetMap(OSM)数据文件。有关 OpenStreetMap 的详情，请查看 OpenStreetMap 网站。

在使用 OptaWeb Vehicle 路由时，请使用以下定义：

*区域 ：**sarth** 映射上的任意区域，由 **OSM** 文件表示。地区可以是国家/地区、城市、洲或一组经常一起使用的国家/地区。例如，**DACH** 区域包括德国(**DE**)、奥地利(**AT**)和瑞士(**CH**)。*

*国家/ 地区代码：由 **ISO-3166** 标准分配给国家的双字母代码。您可以使用国家代码过滤 **geosearch** 结果。由于您可以处理跨越多个国家（例如 **DACH** 区域）的区域，**OptaWeb Vehicle Routing** 接受国家代码列表，以便 **geosearch** 过滤可与此类区域一起使用。有关 国家代码列表，请参阅 **ISO 3166 country Codes***

***Geosearch** ：一个查询类型，在其中提供区域的地址或放置名称作为搜索关键字，并收到多个 **GPS** 位置。返回的位置数量取决于搜索关键字的唯一情况。因为大多数位置名称没有唯一的，所以通过在您工作区域的国家或国家/地区中包括内容，过滤出非相关结果。*

16.2. 下载并构建 OPTAWEB VEHICLE 路由部署文件

*在构建和部署 **OptaWeb Vehicle** 路由前，您必须下载并准备部署文件。*

流程

1. *进入红帽客户门户网站中的 **Software Downloads** 页面（需要登录），然后从下拉列表中选择产品和版本：*
   - *产品： 流程自动化管理器*
   - *版本： **7.13.4***
2. *下载 **Red Hat Process Automation Manager 7.13.4 Kogito 和 OptaPlanner 8 Decision Services Quickstarts (rhpam-7.13.4-kogito-and-optaplanner-quickstarts.zip)**。*
3. *提取 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts.zip** 文件。*
4. *下载 **Red Hat Process Automation Manager 7.13 Maven Repository Kogito 和 OptaPlanner 8 Maven** 存储库 **(rhpam-7.13.4-kogito-maven-repository.zip)**。*
5. *提取 **rhpam-7.13.4-kogito-maven-repository.zip** 文件。*

6. 将 **rhpam-7.13.4-kogito-maven-repository/maven-repository** 子目录的内容复制到 **~/.m2/repository** 目录中。

7. 导航到 **optaweb-8.13.0.Final-redhat-00013/optaweb-vehicle-routing** 目录。

8. 输入以下命令来构建 **OptaWeb Vehicle** 路由：

```
mvn clean package -DskipTests
```

16.3. 使用 RUNLOCALLY.SH 脚本在本地运行 OPTAWEB VEHICLE ROUTING

Linux 用户可以使用 **runLocally.sh Bash** 脚本来运行 **OptaWeb Vehicle** 路由。

注意

runLocally.sh 脚本不会在 **macOS** 中运行。如果您无法使用 **runLocally.sh** 脚本，请参阅 第 16.4 节 “手动配置并运行 OptaWeb Vehicle Routing”。

runLocally.sh 脚本自动执行以下设置步骤，否则必须手动执行：

- 创建数据目录。
- 从 **Geofabrik** 下载所选 **OpenStreetMap(OSM)**文件。
- 尝试自动将国家代码与每个下载的 **OSM** 文件关联。
- 如果独立 **JAR** 文件不存在，则构建项目。
- 使用单一地区参数或以交互方式选择地区，启动 **OptaWeb Vehicle** 路由。

有关执行 **runLocally.sh** 脚本的说明，请参见以下小节：

- 第 16.3.1 节 “以快速启动模式运行 OptaWeb Vehicle Routing runLocally.sh 脚本”
- 第 16.3.2 节 “以互动模式运行 OptaWeb Vehicle Routing runLocally.sh 脚本”
- 第 16.3.3 节 “以非互动模式运行 OptaWeb Vehicle Routing runLocally.sh 脚本”

16.3.1. 以快速启动模式运行 OptaWeb Vehicle Routing runLocally.sh 脚本

使用 OptaWeb Vehicle Routing 入门的最简单方法是运行不带任何参数的 runLocally.sh 脚本。

先决条件

- OptaWeb Vehicle Routing 已通过 Maven 成功构建，如 第 16.2 节 “下载并构建 OptaWeb Vehicle 路由部署文件” 所述。
- 可通过互联网访问。

流程

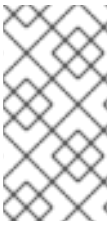
1. 在 rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-vehicle-routing 目录中输入以下命令。

```
./runLocally.sh
```

2. 如果提示您创建 .optaweb-vehicle-routing 目录，请输入 y。第一次运行脚本时，会提示您输入创建此目录。
3. 如果提示下载 OSM 文件，请输入 y。您第一次运行脚本时，OptaWeb Vehicle Routing 下载 Belgium OSM 文件。

   应用程序在下载 OSM 文件后启动。
4. 要打开 OptaWeb Vehicle Routing 用户界面，在 web 浏览器中打开以下 URL：

```
http://localhost:8080
```



注意

第一次运行脚本时，需要几分钟才能启动，因为 **OSM** 文件必须由 **GraphHopper** 导入，并存储为路路图。下一次运行 **runlocal.sh** 脚本时，加载时间会非常快。

后续步骤

第 16.6 节 “使用 OptaWeb Vehicle 路由”

16.3.2. 以互动模式运行 OptaWeb Vehicle Routing runLocally.sh 脚本

使用互动模式查看为每个地区分配了下载的 **OSM** 文件和国家代码的列表。您可以使用互动模式从 **Geofabrik** 下载其他 **OSM** 文件，而无需访问网站并选择下载目标。

先决条件

- **OptaWeb Vehicle Routing** 已通过 **Maven** 成功构建，如 第 16.2 节 “下载并构建 OptaWeb Vehicle 路由部署文件” 所述。
- 可通过互联网访问。

流程

1. 将目录更改为 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-vehicle-routing**。
2. 输入以下命令在互动模式中运行脚本：

```
./runLocally.sh -i
```

3. 在您选择的提示下，输入 **d** 以显示下载菜单。系统将显示之前下载的区域列表，然后是您可以下载的区域列表。
4. 可选：从之前下载的区域列表中选择区域：

a. 在下载的地区列表中，输入与区域关联的号码。

b. 按 **Enter** 键。

5. 可选：下载区域：

a. 输入与您要下载的区域关联的号码。例如，要选择欧洲的映射，请输入 **5**。

b. 要下载映射，请输入 **d**，然后按 **Enter** 键。

c. 要在映射中下载特定区域，请输入 **e**，然后输入与您要下载的区域关联的号码，然后按 **Enter** 键。

使用大型 **OSM** 文件

为获得最佳用户体验，请使用小地区（如欧洲或美国地区）。使用大于 **1 GB** 的 **OSM** 文件需要大量 **RAM** 大小，并需要大量时间（最多几小时）进行初始处理。

应用程序在下载 **OSM** 文件后启动。

6. 要打开 **OptaWeb Vehicle Routing** 用户界面，在 **web** 浏览器中打开以下 **URL**：

```
http://localhost:8080
```

后续步骤

第 16.6 节 “使用 OptaWeb Vehicle 路由”

16.3.3. 以非互动模式运行 OptaWeb Vehicle Routing runLocally.sh 脚本

以非互动模式使用 **OptaWeb Vehicle Routing**，通过单个命令启动 **OptaWeb Vehicle** 路由，其中包含您之前下载的 **OSM** 文件。当您想快速或进行演示时，这非常有用。

先决条件

- **OptaWeb Vehicle Routing** 已通过 **Maven** 成功构建，如 第 16.2 节 “下载并构建 **OptaWeb Vehicle** 路由部署文件” 所述。
- 要用于下载的区域的 **OSM** 文件。有关下载 **OSM** 文件的详情，请参考 第 16.3.2 节 “以互动模式运行 **OptaWeb Vehicle Routing runLocally.sh** 脚本”。
- 可通过互联网访问。

流程

1. 将目录更改为 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-vehicle-routing**。
2. 执行以下命令，其中 **<OSM_FILE_NAME>** 是您之前下载的 **OSM** 文件：

```
./runLocally.sh <OSM_FILE_NAME>
```

后续步骤

第 16.6 节 “使用 **OptaWeb Vehicle** 路由”

16.3.4. 更新数据目录

如果要使用不同的数据目录，您可以更新 **OptaWeb Vehicle Routing** 使用的数据目录。默认数据目录为 **$HOME/.optaweb-vehicle-routing**。

先决条件

- **OptaWeb Vehicle Routing** 已通过 **Maven** 成功构建，如 第 16.2 节 “下载并构建 **OptaWeb Vehicle** 路由部署文件” 所述。

流程

- 要使用不同的数据目录，请在当前数据目录中添加目录绝对路径到 **.DATA_DIR_LAST** 文件。

- 要更改与区域关联的国家代码，编辑当前数据目录中的 **country_codes** 目录中的对应文件。

  例如，如果您为 **SVRF** 下载了 **OSM** 文件，且该脚本无法猜测国家代码，将 **country_codes/stunnel-latest** 的内容设置为 **GB**。

- 要删除区域，请从数据目录中的 **openstreetmap** 目录删除对应的 **OSM** 文件，并删除 **graphhopper** 目录中的区域目录。

16.4. 手动配置并运行 OPTAWEB VEHICLE ROUTING

运行 **OptaWeb Vehicle Routing** 的最简单方法是使用 **runlocal.sh** 脚本。但是，如果您的系统中没有 **Bash**，您可以手动完成 **run.sh** 脚本执行的步骤。

先决条件

- **OptaWeb Vehicle Routing** 已通过 **Maven** 成功构建，如 *第 16.2 节 “下载并构建 OptaWeb Vehicle 路由部署文件”* 所述。

- 可通过互联网访问。

流程

1. 下载路由数据。

   路由引擎需要地理数据来计算在位置间传输的时间。在运行 **OptaWeb Vehicle** 路由前，您必须在本地文件系统中下载并存储 **OpenStreetMap(OSM)**数据文件。

   注意

   OSM 数据文件通常只在 **100 MB** 到 **1 GB** 之间下载，因此最好先下载这些文件，然后再构建或启动 **OptaWeb Vehicle** 路由应用程序。

a. 在 **Web** 浏览器中打开 **http://download.geofabrik.de/**。

b. 点 **Sub Region** 列表中的一个区域，如 。这时将打开 **subregion** 页面。

c. 在 **Sub Regions** 表中，为国家下载 **OSM** 文件**(.osm.pbf)**，例如 **Belgium**。

2. 创建数据结构。

OptaWeb Vehicle Routing 读取和写入文件系统上多种类型的数据。它从 **openstreetmap** 目录中读取 **OSM(OpenStreetMap)**文件，将路网络 图形 写入图形目录，并将用户数据保存在名为 **db** 的目录中。创建一个用于存储所有这些数据的新目录，以便更轻松地升级到未来 **OptaWeb Vehicle** 路由的更新版本，并继续处理之前创建的数据。

a. 创建 **$HOME/.optaweb-vehicle-routing** 目录。

b. 在 **$HOME/.optaweb-vehicle-routing** 目录中创建 **openstreetmap** 目录：

```
$HOME/.optaweb-vehicle-routing
└── openstreetmap
```

c. 将所有下载的 **OSM** 文件（扩展名 **.osm.pbf**）移动到 **openstreetmap** 目录。

剩余的目录结构由 **OptaWeb Vehicle Routing** 应用程序创建，在第一次运行时由 **OptaWeb Vehicle Routing** 应用程序创建。之后，您的目录结构类似以下示例：

```
$HOME/.optaweb-vehicle-routing
├── db
│   └── vrp.mv.db
├── graphhopper
│   └── belgium-latest
└── openstreetmap
    └── belgium-latest.osm.pbf
```

3. 将目录更改为 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-vehicle-routing/optaweb-vehicle-routing-standalone/target**。

4. 要运行 **OptaWeb Vehicle Routing**，请输入以下命令：

```
java \
-Dapp.demo.data-set-dir=$HOME/.optaweb-vehicle-routing/dataset \
-Dapp.persistence.h2-dir=$HOME/.optaweb-vehicle-routing/db \
-Dapp.routing.gh-dir=$HOME/.optaweb-vehicle-routing/graphhopper \
-Dapp.routing.osm-dir=$HOME/.optaweb-vehicle-routing/openstreetmap \
-Dapp.routing.osm-file=<OSM_FILE_NAME> \
-Dapp.region.country-codes=<COUNTRY_CODE_LIST> \
-jar quarkus-app/quarkus-run.jar
```

在这个命令中，替换以下变量：

- **<OSM_FILE_NAME >：**要使用的区域 **OSM** 文件以及之前下载的区域

- **<COUNTRY_CODE_LIST >：**用于过滤 **geosearch** 查询的国家代码的逗号分隔列表。有关国家代码的列表，请参阅 **ISO 3166** 国家 代码。

应用程序在下载 **OSM** 文件后启动。

在以下示例中，**OptaWeb Vehicle Routing** 下载中心美国（中央**-america-latest.osm.pbf**）的 **OSM** 映射，并在 **Belize(BZ)**和 **Guatemala(GT)**的搜索。

```
java \
-Dapp.demo.data-set-dir=$HOME/.optaweb-vehicle-routing/dataset \
-Dapp.persistence.h2-dir=$HOME/.optaweb-vehicle-routing/db \
-Dapp.routing.gh-dir=$HOME/.optaweb-vehicle-routing/graphhopper \
-Dapp.routing.osm-dir=$HOME/.optaweb-vehicle-routing/openstreetmap \
-Dapp.routing.osm-file=entral-america-latest.osm.pbf \
-Dapp.region.country-codes=BZ,GT \
-jar quarkus-app/quarkus-run.jar
```

5. 要打开 **OptaWeb Vehicle Routing** 用户界面，在 **web** 浏览器中打开以下 **URL**：

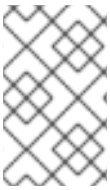
```
http://localhost:8080
```

后续步骤

第 **16.6** 节 “使用 **OptaWeb Vehicle** 路由”

16.5. 在 RED HAT OPENSHIFT CONTAINER PLATFORM 上运行 OPTAWEB VEHICLE ROUTING

Linux 用户可以使用 runOnOpenShift.sh Bash 脚本在 Red Hat OpenShift Container Platform 上安装 OptaWeb Vehicle 路由。



注意

runOnOpenShift.sh 脚本不会在 macOS 上运行。

先决条件

- 您可以访问 OpenShift 集群，并安装了 OpenShift 命令行界面(oc)。有关 Red Hat OpenShift Container Platform 的详情，请参阅安装 OpenShift Container Platform。
- OptaWeb Vehicle Routing 已通过 Maven 成功构建，如 第 16.2 节 “下载并构建 OptaWeb Vehicle 路由部署文件” 所述。
- 可通过互联网访问。

流程

1. 登录或启动 Red Hat OpenShift Container Platform 集群。

a. 输入以下命令，其中 <PROJECT_NAME> 是新项目的名称：

```
oc new-project <PROJECT_NAME>
```

b. 如有必要，将目录改为 rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-vehicle-routing。

c. 输入以下命令执行 runOnOpenShift.sh 脚本并下载 OpenStreetMap(OSM)文件：

```
./runOnOpenShift.sh <OSM_FILE_NAME> <COUNTRY_CODE_LIST>
<OSM_FILE_DOWNLOAD_URL>
```

在这个命令中，替换以下变量：

- **<OSM_FILE_NAME& gt; :** 从 **< OSM_FILE_DOWNLOAD_URL>** 下载的文件名。
- **<COUNTRY_CODE_LIST > :** 用于过滤 **geosearch** 查询的国家代码的逗号分隔列表。有关国家代码的列表，请参阅 **ISO 3166** 国家 代码。
- **<OSM_FILE_DOWNLOAD_URL > : PBF** 格式的 **OSM** 数据文件的 **URL** 可从 **OpenShift** 访问。该文件将在后端启动期间下载，并保存为 **/deployments/local/<OSM_FILE_NAME>**。

在以下示例中，**OptaWeb Vehicle Routing** 下载中心美国（中央**-america-latest.osm.pbf**）的 **OSM** 映射，并在 **Belize(BZ)**和 **Guatemala(GT)**的搜索。

```
./runOnOpenShift.sh central-america-latest.osm.pbf BZ,GT
http://download.geofabrik.de/europe/central-america-latest.osm.pbf
```

注意

如需有关 **runOnOpenShift.sh** 脚本的帮助，请输入 **./runOnOpenShift.sh --help**。

16.5.1. 使用本地更改更新部署的 OptaWeb Vehicle Routing 应用程序

在 **Red Hat OpenShift Container Platform** 上部署 **OptaWeb Vehicle Routing** 应用程序后，您可以更新后端和前端。

先决条件

- **OptaWeb Vehicle Routing** 已通过 **Maven** 成功构建并部署在 **OpenShift** 中。

流程

- 要更新后端，请执行以下步骤：

1. 更改源代码并使用 **Maven** 构建后端模块。

2. 将目录更改为 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-vehicle-routing**。

3. 输入以下命令启动 **OpenShift** 构建：

```
oc start-build backend --from-dir=. --follow
```

- 要更新前端，请执行以下步骤：

1. 更改源代码并使用 **npm** 实用程序构建前端模块。

2. 将目录改为 **source /optaweb-vehicle-routing-frontend**。

3. 输入以下命令启动 **OpenShift** 构建：

```
oc start-build frontend --from-dir=docker --follow
```

后续步骤

第 16.6 节 “使用 OptaWeb Vehicle 路由”

16.6. 使用 OPTAWEB VEHICLE 路由

在 **OptaWeb Vehicle Routing** 应用程序中，您可以在映射中标记多个位置。第一个位置假定为 **depot**。车道必须从此指南到您标记的其他所有位置。

您可以设置载体的数量和每个载车的执行容量。但是，该路由并不保证使用所有载体。应用程序使用所需数量的载体来达到最佳路由要求。

当前版本有一些限制：

- 每个发送到位置的传送都应占用一个载体容量。例如，一个容量为 **10** 的载体，在返回到 **depot** 之前，最多可访问 **10** 个位置。

- 不支持设置列车和位置的自定义名称。

16.6.1. 创建路由

要创建最佳路由，请使用 **OptaWeb Vehicle Routing** 用户界面的 **Demo** 选项卡。

先决条件

- **OptaWeb Vehicle Routing** 正在运行，您可以访问用户界面。

流程

1. 在 **OptaWeb Vehicle Routing** 中，点击 **Demo** 选项卡。
2. 使用映射上方的蓝色减号和加按钮来设置载体数量。每个车辆的默认容量都是 **10**。
3. 根据需要，使用 **map** 中的方块中的加号按钮来缩放。

注意

不要双击 **zoom**。双击 也会创建位置。

4. 点 **depot** 的位置。
5. 单击地图上的其他位置以进行发送点。
6. 如果要删除位置：
   a. 将鼠标光标悬停在位置上，以查看位置名称。
   b. 在屏幕左侧的列表中找到位置名称。

c. 点名称旁边的 **X** 图标。

每次添加或删除位置或更改载体数量时，应用程序都会创建并显示新的最佳路由。如果解决方案使用多个载体，则应用程序会以不同的颜色显示每个载体的路由。

16.6.2. 查看和设置其他详情

您可以使用 **OptaWeb Vehicle Routing** 用户界面中的其他标签页来查看和设置附加详情。

先决条件

- **OptaWeb Vehicle Routing** 正在运行，您可以访问用户界面。

流程

- 单击 **Vehicles** 选项卡，以查看、添加和删除车键，同时设置每个载体的容量。
- 单击 **Visits** 选项卡可以查看和删除位置。
- 单击 **Route** 选项卡，以选择每个载体并查看所选载体的路由。

16.6.3. 使用 OptaWeb Vehicle Routing 创建自定义数据集

有一个由几个大型 **Belgian** 城市组成的内置演示数据集。如果要在 **Load demo** 菜单中有更多演示，您可以准备自己的数据集。

流程

1. 在 **OptaWeb Vehicle Routing** 中，通过点击映射或使用 **geosearch** 添加 **depot** 和一个或多个 **accesss**。
2. 点 **Export**，并将该文件保存到数据集目录中。

注意

数据集目录是 **app.demo.data-set-dir** 属性中指定的目录。

如果应用程序通过 **runLocally.sh** 脚本运行，则数据设置目录设置为 **$HOME/.optaweb-vehicle-routing/dataset**。

否则，属性取自 **application.properties** 文件，默认为 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-vehicle-routing/optaweb-vehicle-routing-standalone/target/local/dataset**。

您可以编辑 **app.demo.data-set-dir** 属性来指定 **diffent** 数据目录。

3. 编辑 **YAML** 文件，并为数据集选择唯一名称。

4. 重启后端。

重新启动后端后，数据设置目录中的文件会出现在 **Load demo** 菜单中。

16.6.4. OptaWeb Vehicle 路由故障排除

如果 **OptaWeb Vehicle Routing** 意外行为，请按照以下步骤排除。

先决条件

- **OptaWeb Vehicle Routing** 正在运行，并意外行为。

流程

1. 要识别问题，请查看后端终端输出日志。

2. 要解决这个问题，请删除后端数据库：

a. 在后端终端窗口中按 **Ctrl+C** 来停止后端。

b. 删除 **optaweb-vehicle-routing/optaweb-vehicle-routing-backend/local/db** 目录。

c. 重启 **OptaWeb Vehicle** 路由。

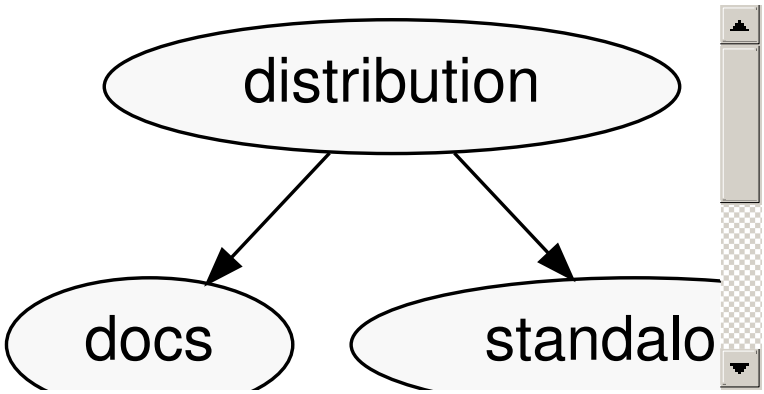
16.7. OPTAWEB VEHICLE ROUTING 开发指南

本节论述了如何在开发模式中配置和运行后端模块。

16.7.1. OptaWeb Vehicle Routing 项目结构

OptaWeb Vehicle Routing 项目是一个多模块 **Maven** 项目。

图 **16.1.** 模块依赖关系树图



后端和前端模块位于模块树的底部。这些模块包含应用源代码。

独立模块是组合后端和前端到单个可执行 **JAR** 文件的装配模块。

distribution 模块表示最终的 **assembly** 步骤。它使用独立应用程序和文档，并将它们打包成易于分发的存档中。

后端和前端是单独的项目，您可以单独构建和部署。实际上，它们使用完全不同的语言编写，并使用不同的工具进行构建。这两个项目都具有提供现代开发人员体验的工具，可在代码更改和正在运行的应用程序之间进行快速切换。

下面章节描述了如何在开发模式下运行后端和前端项目。

16.7.2. OptaWeb Vehicle Routing 后端模块

后端模块包含一个服务器端应用程序，它使用红帽构建的 **OptaPlanner** 优化载体路由。优化是一种 **CPU** 密集型计算，必须避免任何 **I/O** 操作才能将其完整性能。由于目标之一是尽量减少差差成本，因此 **OptaWeb Vehicle Routing** 在 **RAM** 内存中保留出差成本信息。在寻求时，**OptaPlanner** 需要了解用户输入的每个位置之间的差成本。此信息存储在名为 **distance matrix** 的结构中。

当您进入新位置时，**OptaWeb Vehicle Routing** 会计算新位置和所有其他位置之间的差额成本，并在距离列表中保存出差成本。**travel** 成本计算由 **GraphHopper** 路由引擎执行。

后端模块实现以下额外功能：

- 持久性
- 前端的 **websocket** 连接
- 数据集加载、导出和导入

如需了解更多有关后端代码架构的信息，请参阅 *第 16.8 节 “OptaWeb Vehicle 路由后端架构”*。

下面章节描述了如何在开发模式中配置和运行后端。

16.7.2.1. 运行 OptaWeb Vehicle Routing 后端模块

您可以在 **Quarkus** 开发模式下运行后端模块。

先决条件

- **OptaWeb Vehicle Routing** 被配置为 *第 16.4 节 “手动配置并运行 OptaWeb Vehicle Routing”* 所述。

流程

1. 将目录更改为 **rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-vehicle-routing/optaweb-vehicle-routing-backend**。

2. 要在 **development** 模式下运行后端，请输入以下命令：

```
mvn compile quarkus:dev
```

16.7.2.2. 从 IntelliJ IDEA Ultimate 运行 OptaWeb Vehicle Routing 后端模块

您可以使用 **IntelliJ IDEA Ulitmate** 运行 **OptaWeb Vehicle** 路由后端模块，以便更轻松地开发项目。**Intellij IDEA Ultimate** 包括 **Quarkus** 插件，它可自动为使用 **Quarkus** 框架的模块创建运行配置。

流程

使用 **optaweb-vehicle-routing-backend** 运行配置来运行后端。

其他资源

如需更多信息，请参阅 *运行 **Quarkus** 应用程序*。

16.7.2.3. Quarkus 开发模式

在开发模式中，如果存在对后端源代码或配置的更改，并且刷新运行前端的浏览器选项卡，后端会自动重启。

了解有关 ***Quarkus** 开发模式 的更多信息*。

16.7.2.4. 更改 OptaWeb Vehicle Routing 后端模块系统属性值

您可以临时或永久覆盖 **OptaWeb Vehicle Routing** 后端模块的默认系统属性值。

OptaWeb Vehicle Routing 后端模块系统属性存储在 **/src/main/resources/application.properties** 文件中。此文件受版本控制。使用它来永久存储默认配置属性值并定义 **Quarkus** 配置集。

先决条件

- *OptaWeb Vehicle Routing starter 应用程序已下载并提取。有关详情请参考 第 16.2 节 “下载并构建 OptaWeb Vehicle 路由部署文件”。*

流程

- *要临时覆盖默认系统属性值，请在运行 mvn 或 java 命令时包括 -D<PROPERTY>=<VALUE & gt; 参数，其中 <PROPERTY > 是您要更改的属性值，而 < VALUE > 是您要暂时分配给该属性的值。以下示例演示了如何在使用 Maven 在 dev 模式中编译 Quarkus 项目时，将 quarkus.http.port 系统属性的值临时更改为 8181 ：*

```
mvn compile quarkus:dev -Dquarkus.http.port=8181
```

这会临时更改 /src/main/resources/application.properties 文件中的属性值。

- *要永久更改配置值，例如存储特定于您的开发环境的配置，请将 env-example 文件的内容复制到 optaweb-vehicle-routing-backend/.env 文件。*

此文件不包括在版本控制中，因此克隆存储库时不存在此文件。您可以在 .env 文件中进行更改，而不影响 Git 工作树。

其他资源

有关 OptaWeb Vehicle Routing 配置属性的完整列表，请参阅 第 16.9 节 “OptaWeb Vehicle 路由后端配置属性”。

16.7.2.5. OptaWeb Vehicle Routing backend logging

OptaWeb Vehicle Routing 使用 SLF4J API 和 Logback 作为日志框架。如需更多信息，请参阅 Quarkus - 配置日志。

16.7.3. 使用 OptaWeb Vehicle Routing 前端模块

前端项目通过 Create React App 启动。创建 React App 提供了多个脚本和依赖项，可帮助开发并构建用于生产环境的应用程序。

先决条件

- *OptaWeb Vehicle Routing starter 应用程序已下载并提取。有关详情请参考 第 16.2 节 “下载并构建 OptaWeb Vehicle 路由部署文件”。*

流程

1. 在 Fedora 中，输入以下命令来设置开发环境：

```
sudo dnf install npm
```

有关安装 npm 的更多信息，请参阅下载并安装 Node.js 和 npm。

2. 将目录更改为 rhpam-7.13.4-kogito-and-optaplanner-quickstarts/optaweb-8.13.0.Final-redhat-00013/optaweb-vehicle-routing/optaweb-vehicle-routing-frontend。

3. 安装 npm 依赖项：

```
npm install
```

与 Maven 不同，npm 软件包管理器在项目目录下将依赖项安装到 node_modules 中，并且仅在您执行 npm install 时执行此操作。每当 package.json 更改后列出的依赖项时，例如，当您将更改拉取到 master 分支时，都必须在运行开发服务器前执行 npm install。

4. 输入以下命令来运行开发服务器：

```
npm start
```

5. 如果没有自动打开，在网页浏览器中打开 http://localhost:3000/。

默认情况下，npm start 命令会尝试在您的默认浏览器中打开这个 URL。

注意

如果您不希望 npm start 命令在每次运行时都打开一个新浏览器标签页，请导出 BROWSER=none 环境变量。您可以使用 .env.local 文件使这个首选项持久。要做到这一点，请输入以下命令：

```
echo BROWSER=none >> .env.local
```

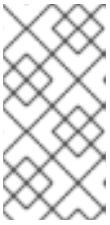
每当您在前端源代码中进行更改时，浏览器都会刷新该页面。在终端中运行的开发服务器进程会提取更改，并将编译和 **lint** 错误输出到控制台。

6. 输入以下命令来运行测试：

```
npm test
```

7. 在执行 **npm start** 或 **npm run build** 时，更改 **REACT_APP_BACKEND_URL** 环境变量的值，以指定 **npm** 要使用的后端项目的位置，例如：

```
REACT_APP_BACKEND_URL=http://10.0.0.123:8081
```



注意

在 **npm** 构建过程中，环境变量在 **JavaScript** 捆绑包内被硬编码，因此您必须在构建和部署前端前指定后端位置。

如需了解更多有关 **React** 环境变量的信息，请参阅添加 自定义环境变量。

8. 要构建前端，请输入以下命令之一：

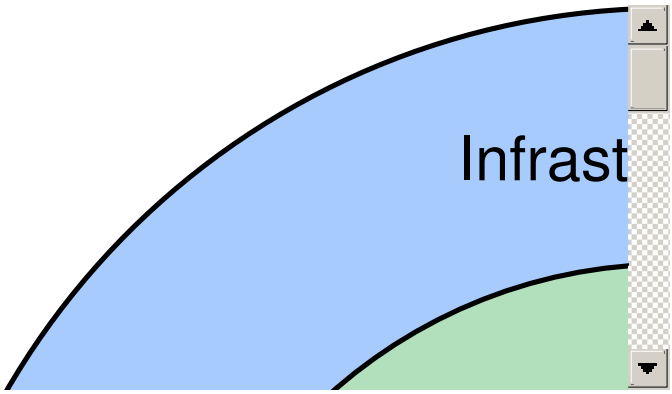
```
./mvnw install
```

```
mvn install
```

16.8. OPTAWEB VEHICLE 路由后端架构

对于应用程序来说，域模型和使用案例是必不可少的。**OptaWeb Vehicle Routing** 域模型位于架构中心，它包括在嵌入用例的应用程序层中。路由优化、距离计算、持久性和网络通信等功能被视为实施详情，并将其置于架构的最顶层。

图 **16.2.** 应用程序层图



16.8.1. 代码机构

后端代码由三个层进行组织，在上图中显示。

```
org.optaweb.vehiclerouting
├── domain
├── plugin          # Infrastructure layer
│   ├── persistence
│   ├── planner
│   ├── routing
│   └── rest
└── service         # Application layer
    ├── demo
    ├── distance
    ├── error
    ├── location
    ├── region
    ├── reload
    ├── route
    └── vehicle
```

service 软件包包含实施用例的应用层。**plugin** 软件包包含基础架构层。

每个层中的代码进一步由函数组织。这意味着每个服务或插件都有自己的软件包。

16.8.2. 依赖项规则

编译时依赖项只能从外部层指向中心。遵循此规则有助于使领域模型独立于底层框架和其他实施细节，并更加精确地模拟业务实体的行为。通过演示和永久性推向外围，测试业务实体和使用案例的行为更加容易。

域没有依赖项。

服务仅依赖于域。如果服务需要发送结果（例如到数据库或客户端），它会使用输出边界接口。其实施由 上下文和依赖项注入 (CDI)容器注入。

插件以两种方式依赖服务。首先，根据用户输入或从优化引擎发出的路由更新等事件调用服务。服务注入插件，将其构造和依赖项解析到 IoC 容器的负担中。第二，插件实施服务输出边界接口来处理用例结果，例如对数据库进行永久性更改或向 Web UI 发送响应。

16.8.3. 域软件包

domain 软件包包含为此项目域建模 的业务对象，如 Location、Vehicle、Route。这些对象严格以业务为导向，并且不得受到任何工具和框架的影响，例如对象关系映射工具和 Web 服务框架。

16.8.4. service 软件包

service 软件包包含实施 用例 的类。用例描述了您要执行的操作，例如添加新的位置、更改载体容量或查找地址的协调。监管用例的业务规则使用域对象来表达。

服务通常需要与外层中的插件交互，如持久性、Web 和优化。为了满足层之间的依赖关系规则，服务和插件之间的交互以定义服务依赖项的接口表示。插件可通过提供实现服务边界接口的 bean 来满足服务的依赖项。CDI 容器创建了插件 bean 的实例，并在运行时将其注入服务。这是 control 原则的 inversion of example。

16.8.5. plugin 软件包

插件 软件包包含基础架构功能，如优化、持久性、路由和网络。

16.9. OPTAWEB VEHICLE 路由后端配置属性

您可以设置下表中列出的 OptaWeb Vehicle Routing 应用程序属性。

属性	类型	示例	描述
app.demo.data-set-dir	相对或绝对路径	/home/user/.optaweb-vehicle-routing/dataset	从这个目录中载入自定义数据集。默认为 local/dataset。

属性	类型	示例	描述
app.persistence.h2-dir	相对或绝对路径	**/home/user/.optaweb-vehicle-routing/db**	H2 用于存储数据库文件的目录。默认为 **local/db**。
app.region.country-codes	ISO 3166-1 alpha-2 国家代码列表	**美国,GB,IE,DE,AT,CH** 可能是空的	限制 geosearch 结果。
app.routing.engine	Enumeration	**air,graphhopper**	路由引擎实施。默认为 **graphhopper**。
app.routing.gh-dir	相对或绝对路径	**/home/user/.optaweb-vehicle-routing/graphhopper**	GraphHopper 用来存储移动网络图形的目录。默认为 **local/graphhopper**。
app.routing.osm-dir	相对或绝对路径	**/home/user/.optaweb-vehicle-routing/openstreetmap**	包含 OSM 文件的目录。默认为 **local/openstreetmap**。
app.routing.osm-file	文件名	**belgium-latest.osm.pbf**	由 GraphHopper 加载的 OSM 文件的名称。该文件必须放在 **app.routing.osm-dir** 下。
optaplanner.solver.termination.spent-limit	**java.time.Duration**	• **1m** • **150s** • **P2dT21h** (**PnDTnHnMn.nS**)	解决者在发生位置更改后应运行多长时间。
server.address	IP 地址或主机名	**10.0.0.123**, **my-vrp.geo-1.openshiftapps.com**	要绑定服务器的网络地址。
server.port	端口号	**4000**, **8081**	服务器 HTTP 端口。

*附录 **A.** 版本信息*

*文档最后于 **2023** 年 **9** 月 **5** 日星期二更新。*

附录 B. 联系信息

Red Hat Process Automation Manager 文档团队 ：brms-docs@redhat.com