Red Hat build of Keycloak 24.0

迁移指南

Last Updated: 2024-06-28

法律通告

摘要

本指南由红帽构建的 Keycloak 迁移指南组成。

目录

第 1 章 将 RED HAT SINGLE SIGN-ON 7.6 迁移到 RED HAT BUILD OF KEYCLOAK

本指南的目的是记录成功将 Red Hat Single Sign-On 7.6 迁移到 Red Hat build of Keycloak 24.0 所需的步骤。以下元素的说明迁移：

- Red Hat Single Sign-On 7.6 server
- 在 OpenShift 上部署 Operator
- OpenShift 上的模板部署
- 由 Red Hat Single Sign-On 7.6 保护的应用程序
- 自定义供应商
- 自定义主题

本指南还包括将上游 Keycloak 迁移到红帽构建的 Keycloak 24.0 的指南。在开始迁移前，您可以考虑安装一个新的红帽构建的 Keycloak 实例，以熟悉本发行版本的更改。请参阅 Red Hat build of Keycloak 入门指南。

第 2 章 迁移 RED HAT SINGLE SIGN-ON 7.6 服务器

本节提供有关迁移从 ZIP 发行版部署的单机服务器的说明。Red Hat build of Keycloak 24.0 使用 Quarkus 构建，它替换了 Red Hat Single Sign-On 7.6 使用的 Red Hat JBoss Enterprise Application Platform (JBoss EAP)。

对服务器的主要更改如下：

- 改进的配置体验，它得到简化，并支持设置配置选项的灵活性。
- 服务器的 RPM 发行版不再可用

2.1. 先决条件

- 之前的 Red Hat Single Sign-On 7.6 实例被关闭，使其不使用红帽构建的 Keycloak 使用的数据库实例。
- 您备份了数据库。
- OpenJDK17 已安装。
- 您已查看了 发行注记。

2.2. 迁移过程概述

以下小节提供了这些迁移步骤的说明：

1. 下载红帽构建的 Keycloak。
2. 迁移配置。
3. 迁移数据库。
4. 启动 Red Hat build of Keycloak 服务器。

2.3. 下载 RED HAT BUILD OF KEYCLOAK

Red Hat build of Keycloak 服务器下载 ZIP 文件包含脚本和二进制文件，用于运行红帽构建的 Keycloak 服务器。

1. 从红帽客户门户下载红帽构建的 Keycloak 服务器分发 文件。
2. 使用 unzip 命令解包 ZIP 文件。

2.4. 迁移配置

配置红帽构建的 Keycloak 服务器的新统一方法是通过配置选项。Red Hat Single Sign-On 7.6 配置机制（如 standalone.xml、jboss-cli 等）不再适用。

每个选项可以通过以下配置源定义：

- CLI 参数
- 环境变量

- 配置文件
- Java KeyStore 文件

如果通过不同的配置源指定相同的配置选项，则使用上述列表中的第一个源。

所有配置选项都位于所有不同的配置源中，其主要区别是密钥的格式。例如，以下有四个配置数据库主机名的方法：

Source	格式
CLI 参数	--db-url-host cliValue
环境变量	KC_DB_URL_HOST=envVarValue
配置文件	db-url-host=confFileValue
Java KeyStore 文件	kc.db-url-host=keystoreValue

kc.sh --help 命令以及红帽构建的 Keycloak 文档提供了所有可用配置选项的完整列表，其中选项按缓存、数据库等类别分组。此外，每个区域都存在单独的章节，如配置数据库的章节。https://access.redhat.com/documentation/zh-cn/red_hat_build_of_keycloak/24.0/html-single/server_guide//db-

其他资源

- 配置 Keycloak

2.4.1. 迁移数据库配置

与 Red Hat Single Sign-On 7.6 不同，红帽构建的 Keycloak 内置支持支持的数据库，不再需要手动安装和配置数据库驱动程序。例外是 Oracle 和 Microsoft SQL Server，它仍需要手动安装驱动程序。

在配置中，请考虑现有 Red Hat Single Sign-On 7.6 安装中的 datasource 子系统，并将这些配置映射到红帽构建的 Keycloak 中数据库配置类别中提供的选项。例如，以前的配置如下所示：

```
<datasource jndi-name="java:jboss/datasources/KeycloakDS" pool-name="KeycloakDS"
enabled="true" use-java-context="true" statistics-enabled="true">
    <connection-url>jdbc:postgresql://mypostgres:5432/mydb?
currentSchema=myschema</connection-url>
    <driver>postgresql</driver>
    <pool>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>50</max-pool-size>
    </pool>
    <security>
        <user-name>myuser</user-name>
        <password>myuser</password>
    </security>
</datasource>
```

在 Red Hat build of Keycloak 中，使用 CLI 参数的等效配置将是：

```
kc.sh start
  --db postgres
  --db-url-host mypostgres
  --db-url-port 5432
  --db-url-database mydb
  --db-schema myschema
  --db-pool-min-size 5 --db-pool-max-size 50
  --db-username myser --db-password myuser
```

注意

考虑将数据库凭据存储在安全 KeyStore 配置源中。

其他资源

- 配置数据库，其中还包含安装 Oracle 和 Microsoft SQL Server JDBC 驱动程序的说明
- 使用 Java KeyStore 文件设置敏感选项，它提供如何安全地存储数据库凭据的说明。

2.4.2. 迁移 HTTP 和 TLS 配置

默认情况下，HTTP 被禁用，每当使用生产模式（由 **start** 选项代表）时，都需要 TLS 配置。

您可以使用 **--http-enabled=true** 配置选项启用 HTTP，但不建议使用它，除非红帽构建的 Keycloak 服务器位于完全隔离的网络中，且没有内部或外部攻击者能够观察网络流量的风险。

Red Hat build of Keycloak 实例具有不同的上下文根(URL 路径)，因为它在 Red Hat Single Sign-On 7.6 默认附加 **/auth** 时使用服务器的根目录。要模拟旧行为，可以使用 **--http-relative-path=/auth** 配置选项。默认端口保持不变，但也可以通过 **--http-port 和-- https-port** 选项更改。

有两种方法可用于配置 TLS，可以通过 PEM 格式的文件或 Java 密钥存储来配置。例如，Java Keystore 的以前的配置如下所示：

```
<tls>
 <key-stores>
          <key-store name="applicationKS">
              <credential-reference
  clear-text="password"/>
  <implementation type="JKS"/>
              <file
  path="/path/to/application.keystore”/>
          </key-store>
    </key-stores>
    <key-managers>
       <key-manager name="applicationKM"
        key-store="applicationKS">
            <credential-reference
         clear-text="password"/>
       </key-manager>
    </key-managers>
    <server-ssl-contexts>
       <server-ssl-context name="applicationSSC"
    key-manager="applicationKM"/>
    </server-ssl-contexts>
</tls>
```

在 Red Hat build of Keycloak 中，使用 CLI 参数的等效配置如下：

```
kc.sh start
  --https-key-store-file /path/to/application.keystore
  --https-key-store-password password
```

在 Red Hat build of Keycloak 中，您可以通过以 PEM 格式提供证书来配置 TLS，如下所示：

```
kc.sh start
  --https-certificate-file /path/to/certfile.pem
  --https-certificate-key-file /path/to/keyfile.pem
```

其他资源

- 配置 TLS

2.4.3. 迁移集群和缓存配置

Red Hat Single Sign-On 7.6 为将服务器作为独立、独立集群和域集群运行提供了不同的操作模式。这些模式在启动脚本和配置文件中有所不同。Red Hat build of Keycloak 提供了带有单个启动脚本的简化解决方案： **kc.sh**。

要将服务器作为独立或独立集群运行，请使用 **kc.sh** 脚本：

红帽构建的 Keycloak	Red Hat Single Sign-On 7.6
./kc.sh start --cache=local	./standalone.sh
./kc.sh start [--cache=ispn]	./standalone.sh --server-config=standalone-ha.xml

--cache 参数的默认值为 start 模式：

- **local** - 当执行 start-dev 命令时
- **ISP** N- 执行 start 命令时

在 Red Hat Single Sign-On 7.6 中，集群和缓存配置是通过 Infinispan 子系统完成的，而在 Red Hat build of Keycloak 中，大多数配置都是通过单独的 Infinispan 配置文件完成的。例如，以前的 Infinispan 配置如下所示：

```
<subsystem xmlns="urn:jboss:domain:infinispan:13.0">
<cache-container name="keycloak" marshaller="JBOSS" modules="org.keycloak.keycloak-model-infinispan">
            <local-cache name="realms">
                <heap-memory size="10000"/>
            </local-cache>
            <local-cache name="users">
                <heap-memory size="10000"/>
            </local-cache>
            <local-cache name="sessions"/>
            <local-cache name="authenticationSessions"/>
```

```
            <local-cache name="offlineSessions"/>
    ...
</cache-container>
</subsystem>
```

在 Red Hat build of Keycloak 中，默认的 Infinispan 配置文件位于 **conf/cache-ispn.xml** 文件中。您可以提供自己的 Infinispan 配置文件，并使用 CLI 参数指定，如下所示：

```
kc.sh start --cache-config-file my-cache-file.xml
```

注意

红帽构建的 Keycloak 不支持域集群模式。

传输**堆**栈

红帽构建的 Keycloak 默认和自定义 JGroups 传输堆栈不需要迁移。

唯一的改进是通过提供 CLI 选项 cache-stack 来覆盖缓存配置文件中定义的堆栈，其具有优先权。考虑上面指定的 Infinispan 配置文件 **my-cache-file.xml** 的一部分，使用自定义 JGroups 传输堆栈，如下所示：

您可以注意 keycloak 缓存容器的传输堆栈设置为 tcp，但可使用 CLI 选项覆盖，如下所示：

```
<jgroups>
 <stack name="my-encrypt-udp" extends="udp">
  …
 </stack>
</jgroups>

<cache-container name="keycloak">
 <transport stack="tcp"/>
 …
</cache-container>
```

```
kc.sh start
   --cache-config-file my-cache-file.xml
   --cache-stack my-encrypt-udp
```

执行上述命令后，将使用 **my-encrypt-udp** 传输堆栈。

其他资源

- 配置分布式缓存

2.4.4. 迁移主机名和代理配置

在 Red Hat build of Keycloak 中，您现在被义务来配置 Hostname SPI，以便在重定向用户或与其客户端通信时，服务器将如何创建前端和后端 URL。

例如，如果您有一个类似于 Red Hat Single Sign-On 7.6 安装中的配置：

```
<spi name="hostname">
    <default-provider>default</default-provider>
```

```
    <provider name="default" enabled="true">
      <properties>
        <property name="frontendUrl" value="myFrontendUrl"/>
        <property name="forceBackendUrlToFrontendUrl" value="true"/>
      </properties>
    </provider>
</spi>
```

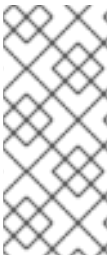
您可以在红帽构建的 Keycloak 中将其转换为以下配置选项：

```
kc.sh start
  --hostname-url myFrontendUrl
  --hostname-strict-backchannel true
```

hostname-url 配置选项允许您设置集群从集群前面的入口层公开到公共的基本 URL。您还可以通过设置 **hostname-admin-url** 配置选项来为管理资源设置 URL。

红帽构建的 Keycloak 允许您配置应反映哪些反向代理标头。您可以使用 **Forwarded** 标头或 **X-ForwardedDebug** 标头集合。例如：

```
kc.sh start --proxy-headers xforwarded
```



注意

主机名和代理配置用于确定资源 URL (redirect URI、CSS 和 JavaScript 链接、OIDC 已知的端点等)，而不是主动阻止传入的请求。没有 hostname/proxy 选项更改红帽构建的 Keycloak 服务器侦听的实际绑定地址或端口 - 这负责 HTTP/TLS 选项。在 Red Hat Single Sign-On 7.6 中，强烈建议设置主机名，但不强制设置。在 Red Hat build of Keycloak 中，现在需要使用 start 命令时，除非使用 **--hostname-strict=false** 选项明确禁用。

其他资源

- 使用反向代理
- 配置主机名

2.4.5. 迁移信任存储配置

truststore 用于外部 TLS 通信，如 HTTPS 请求和 LDAP 服务器。要使用信任存储，您可以将远程服务器或 CA 的证书导入到信任者中。然后，您可以启动红帽构建的 Keycloak 服务器，指定系统信任存储。

例如，以前的配置如下所示：

```
<spi name="truststore">
  <provider name="file" enabled="true">
    <properties>
      <property name="file" value="path/to/myTrustStore.jks"/>
      <property name="password" value="password"/>
      <property name="hostname-verification-policy" value="WILDCARD"/>
    </properties>
  </provider>
</spi>
```

红帽构建的 Keycloak 支持采用以下格式的信任存储： PEM 文件，或带有扩展 .p12 和 .pfx 的 PKCS12 文件。对于 PKCS12 文件，证书必须加密，这意味着不需要密码。需要转换 JKS 信任存储。例如：

```
keytool -importkeystore -srckeystore path/to/myTrustStore.jks \
  -destkeystore path/to/myTrustStore.p12 \
  -srcstoretype jks \
  -deststoretype pkcs12
  -srcstorepass password
  -deststorepass “”
```

在本例中，生成的 CLI 参数如下：

```
kc.sh start
   --truststore-paths path/to/myTrustStore.p12
   --tls-hostname-verifier WILDCARD
```

其他资源

- 为传出请求配置可信证书

2.4.6. 迁移 vault 配置

Keystore Vault 是 Vault SPI 的实现，对于将 secret 存储在裸机安装中非常有用。此密码库是 Red Hat Single Sign-On 7.6 中的 Elytron Credential Store 的替代。

```
<spi name="vault">
    <provider name="elytron-cs-keystore" enabled="true">
      <properties>
          <property name="location" value="path/to/keystore.p12"/>
           <property name="secret" value="password"/>
    </properties>
  </provider>
</spi>
```

在 Red Hat build of Keycloak 中，使用 CLI 参数的等效配置将是：

```
kc.sh start
  --vault keystore
  --vault-file /path/to/keystore.p12
  --vault-pass password
```

然后，可以在管理控制台中的多个位置访问存储在密码库中的 secret。当涉及从现有 Elytron vault 迁移到基于 Java KeyStore 的新密码库时，不需要更改 realm 配置。如果新创建的 Java 密钥存储包含相同的机密，您的现有域配置应该可以正常工作。

假设您使用默认的 **REALM_UNDERSCORE_KEY** 键解析器，该机密可以被 **${vault.realm-name_alias}**（例如，在 LDAP 用户联邦配置中）访问。

其他资源

- 使用密码库.

2.4.7. 迁移 JVM 设置

Red Hat build of Keycloak 中的 JVM 设置方法类似于 Red Hat Single Sign-On 7.6 方法。您仍然需要设置特定的环境变量，但 **/bin** 文件夹不包含配置文件，如 **standalone.conf**。

红帽 Keycloak 的构建提供了各种默认 JVM 参数，该参数适用于大多数部署，因为它在内存分配和 CPU 开销方面提供了良好的吞吐量和效率。另外，其他默认 JVM 参数可确保平稳运行红帽构建的 Keycloak 实例，因此当您更改用例的参数时请小心。

要更改 JVM 参数或 GC 设置，您可以设置特定环境变量，这些变量指定为 Java 选项。对于这些设置的完整覆盖，请指定 **JAVA_OPTS** 环境变量。

当只需要附加特定 Java 属性时，您可以指定 **JAVA_OPTS_APPEND** 环境变量。如果没有指定 **JAVA_OPTS** 环境变量，则会使用默认的 Java 属性，并可在 **./kc.sh** 脚本中找到。

例如，您可以指定特定的 Java 选项，如下所示：

```
export JAVA_OPTS_APPEND=-XX:+HeapDumpOnOutOfMemoryError
kc.sh start
```

2.4.8. 迁移 SPI 供应商配置

通过新的配置系统提供了 SPI 提供程序的配置。这是旧格式：

```
<spi name="<spi-id>">
    <provider name="<provider-id>" enabled="true">
        <properties>
            <property name="<property>" value="<value>"/>
        </properties>
    </provider>
</spi>
```

这是新格式：

```
spi-<spi-id>-<provider-id>-<property>=<value>
```

Source	格式
CLI	./kc.sh start --spi-connections-http-client-default-connection-pool-size 10
环境变量	KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_POOL_SIZE=10
配置文件	spi-connections-http-client-default-connection-pool-size=10
Java 密钥存储文件	kc.spi-connections-http-client-default-connection-pool-size=10

其他资源

- 所有提供程序配置.

2.4.9. 对配置进行故障排除

使用这些命令进行故障排除：

- **kc.sh show-config** - 显示从中加载特定属性的配置源及其值。您可以检查属性及其值是否已正确传播。
- 当出现错误时，kc **.sh --verbose start** - 会输出整个错误堆栈 trace。

2.5. 迁移数据库

Red Hat build of Keycloak 可以自动迁移数据库模式，或者您可以选择手动执行它。默认情况下，当您首次启动新安装时，数据库会被自动迁移。

2.5.1. 自动关系数据库迁移

要执行自动迁移，请启动连接到所需数据库的服务器。如果服务器的新版本更改了数据库架构，则会迁移它。

2.5.2. 手动关系数据库迁移

要启用数据库模式的手动升级，请将默认 connection-jpa 供应商的 **migration-strategy** 属性值设置为 *manual* ：

```
kc.sh start --spi-connections-jpa-legacy-migration-strategy manual
```

使用此配置启动服务器时，它会检查是否需要迁移数据库。所需的更改被写入 **bin/keycloak-database-update.sql** SQL 文件，您可以检查并手动针对数据库运行。

要更改导出的 SQL 文件的路径和名称，请设置默认 connection- **jpa 供应商的 migration- export** 属性：

```
kc.sh start
  --spi-connections-jpa-legacy-migration-export <path>/<file.sql>
```

有关如何将此文件应用到数据库的详情，请查看您的关系数据库文档。将更改写入到文件后，服务器将退出。

2.6. 启动 RED HAT BUILD OF KEYCLOAK SERVER

启动 Red Hat Single Sign-On 7.6 和 Red Hat build of Keycloak 的发布与执行的脚本不同。这些脚本位于服务器分发的 **/bin** 文件夹中。

2.6.1. 在开发模式下启动服务器

要在不担心提供任何属性的情况下尝试红帽构建的 Keycloak，您可以在开发模式中启动发布，如下表所述。但请注意，这个模式严格用于开发，不应在生产环境中使用。

红帽构建的 Keycloak	Red Hat Single Sign-On 7.6
./kc.sh start-dev	./standalone.sh

警告

开发模式不应在生产环境中使用。

2.6.2. 以生产环境模式启动服务器

Red Hat build of Keycloak 为生产环境有一个专用的启动模式： **./kc.sh start**。使用 **start-dev** 运行的不同是不同的默认配置值。它自动使用严格和默认安全配置设置。在 production 模式中，禁用 HTTP，且需要显式 TLS 和主机名配置。

其他资源

- 为生产环境配置 Keycloak
- 优化 Keycloak 启动

第 3 章 在 OPENSHIFT 上迁移 OPERATOR 部署

要适应改进的服务器配置，红帽构建的 Keycloak Operator 被完全重新创建。Operator 提供了与红帽构建的 Keycloak 完全集成，但它与 Red Hat Single Sign-On 7.6 Operator 不兼容。使用新的 Operator 需要创建新的红帽构建的 Keycloak 部署。如需了解更多详细信息，请参阅 Operator 指南。

3.1. 先决条件

- 之前的 Red Hat Single Sign-On 7.6 实例被关闭，使其不使用红帽构建的 Keycloak 使用的数据库实例。
- 如果使用不受支持的嵌入式数据库（由 Red Hat Single Sign-On 7.6 Operator 管理），则它已转换为用户置备的外部数据库。
- 数据库备份已创建。
- 您已查看了 发行注记。

3.2. 迁移过程

1. 将红帽构建的 Keycloak Operator 安装到命名空间中。
2. 创建新 CR 和相关的 Secret。手动将 Red Hat Single Sign-On 7.6 配置迁移到新的 Keycloak CR。
3. 如果使用自定义供应商，请迁移它们并创建自定义红帽构建的 Keycloak 容器镜像，使其包含它们。
4. 如果使用自定义主题，请迁移它们并创建自定义的 Keycloak 容器镜像构建来包括它们。

3.3. 迁移 KEYCLOAK CR

Keycloak CR 现在支持所有服务器配置选项。在 CR 的 spec 下，所有相关选项都作为第一类的会面字段提供。CR 中的所有选项遵循相同的命名约定，这与在裸机和 Operator 部署间体验的服务器选项相同。

另外，您可以在 **additionalOptions** 字段中定义 CR 缺少的任何选项，如 SPI 供应商配置。另一个选择是使用 **podTemplate** （一个技术预览字段）来修改原始 Kubernetes 部署 pod 模板，如果支持的替代方案并不存在作为 CR 中的第一个类 该文件字段。

下面显示了一个通过 Operator 部署红帽构建的 Keycloak CR 示例：

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  instances: 1
  db:
    vendor: postgres
    host: postgres-db
    usernameSecret:
      name: keycloak-db-secret
      key: username
    passwordSecret:
```

```
      name: keycloak-db-secret
      key: password
  http:
    tlsSecret: example-tls-secret
  hostname:
    hostname: test.keycloak.org
  additionalOptions:
    - name: spi-connections-http-client-default-connection-pool-size
      value: 20
```

请注意 CLI 配置的位置：

```
./kc.sh start --db=postgres --db-url-host=postgres-db --db-username=user --db-password=pass --
https-certificate-file=mycertfile --https-certificate-key-file=myprivatekey --hostname=test.keycloak.org
--spi-connections-http-client-default-connection-pool-size=20
```

其他资源

- 基本 Keycloak 部署
- 高级配置

3.3.1. 迁移数据库配置

Red Hat build of Keycloak 可以使用与之前 Red Hat Single Sign-On 7.6 使用相同的数据库实例。当红帽首次构建 Keycloak 时，数据库架构将自动迁移。

警告

不支持迁移由 Red Hat Single Sign-On 7.6 Operator 管理的嵌入式数据库。

在 Red Hat Single Sign-On 7.6 Operator 中，外部数据库连接使用 Secret 配置，例如：

```
apiVersion: v1
kind: Secret
metadata:
  name: keycloak-db-secret
  namespace: keycloak
  labels:
    app: sso
stringData:
  POSTGRES_DATABASE:  kc-db-name
  POSTGRES_EXTERNAL_ADDRESS: my-postgres-hostname
  POSTGRES_EXTERNAL_PORT: 5432
  POSTGRES_USERNAME: user
  POSTGRES_PASSWORD: pass
type: Opaque
```

在 Red Hat build of Keycloak 中，数据库直接在 Keycloak CR 中配置，其凭证被引用为 Secret，例如：

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  db:
    vendor: postgres
    host: my-postgres-hostname
    port: 5432
    usernameSecret:
      name: keycloak-db-secret
      key: username
    passwordSecret:
      name: keycloak-db-secret
      key: password
  ...

apiVersion: v1
kind: Secret
metadata:
  name: keycloak-db-secret
stringData:
  username: "user"
  password: "pass"
type: Opaque
```

3.3.1.1. 支持的数据库供应商

Red Hat Single Sign-On 7.6 Operator 仅支持 PostgreSQL 数据库，但红帽构建的 Keycloak Operator 支持服务器支持的所有数据库供应商。

3.3.2. 迁移 TLS 配置

Red Hat Single Sign-On 7.6 Operator 默认将服务器配置为使用 OpenShift CA 生成的 TLS Secret。Red Hat build of Keycloak Operator 不对 TLS 进行任何假设来满足生产环境的最佳实践，并要求用户提供自己的 TLS 证书和密钥对，例如：

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  http:
    tlsSecret: example-tls-secret
  ...
```

tlsSecret 中引用的 secret 的预期格式应使用标准的 Kubernetes TLS Secret (**kubernetes.io/tls**)类型。

默认情况下，Red Hat Single Sign-On 7.6 Operator 在 Route 上默认使用重新加密 TLS 终止策略。默认情况下，Red Hat build of Keycloak Operator 使用 passthrough 策略。另外，Red Hat Single Sign-On 7.6 Operator 支持配置 TLS 终止。红帽构建的 Keycloak Operator 不支持当前版本中的 TLS 终止。

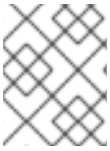
如果默认 Operator 管理的 Route 不满足所需的 TLS 配置，则需要用户创建自定义 Route，并禁用它：

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ingress:
    enabled: false
  ...
```

3.3.3. 使用自定义镜像进行扩展

为了反映最佳实践并支持不可变容器，红帽构建的 Keycloak Operator 不再支持在 Keycloak CR 中指定扩展。要部署扩展，必须构建优化的自定义镜像。Keycloak CR 现在包含指定红帽构建的 Keycloak 镜像的专用字段，例如：

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  image: quay.io/my-company/my-keycloak:latest
  ...
```



注意

在指定自定义镜像时，Operator 会假定它已被优化，且不会在每个服务器启动时执行昂贵的优化。

其他资源

- 在 Operator 中使用自定义 Keycloak 镜像
- 创建自定义和优化的容器镜像

3.3.4. 删除升级策略选项

Red Hat Single Sign-On 7.6 Operator 支持在执行服务器升级时重新创建和滚动策略。这种方法并不实际。只有 Red Hat Single Sign-On 7.6 Operator 应该在执行升级和数据库迁移前缩减部署，则最多可选择用户。在安全使用滚动策略时，用户不知道用户。

因此，红帽构建的 Keycloak Operator 中删除了这个选项，它会始终隐式执行 recreate 策略，这会在使用新服务器容器镜像创建 Pod 前缩减整个部署，以确保只有一个服务器版本访问数据库。

3.3.5. 默认公开的健康端点

Red Hat build of Keycloak 将服务器配置为默认公开一个简单的健康端点，供 OpenShift 探测使用。端点不会公开任何有关部署的安全敏感数据，但无需任何身份验证即可访问。作为替代方案，在自定义 Route 上可以阻止 < *your-server-context-root*> /**health** endpoint。

例如，

1. 创建为 TLS 边缘终止配置的 Keycloak。
   确保省略 **tlsSecret** 字段：

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  proxy:
   Headers:xforwarded
  hostname:
    hostname: example.com
  ...
```

2. 创建阻塞路由以禁止访问健康端点：

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: example-kc-block-health
  annotations:
    haproxy.router.openshift.io/rewrite-target: /404
spec:
  host: example.com
  path: /health
  to:
    kind: Service
    name: example-kc-service
  port:
    targetPort: http
  tls:
    termination: edge
```

注意

基于路径的路由需要为 edge 或 reencrypt 配置 TLS 终止。默认情况下，Operator 使用 passthrough。

3.3.6. 使用 Pod 模板迁移高级部署选项

Red Hat Single Sign-On 7.6 Operator 为部署配置公开多个低级别字段，如卷。Red Hat build of Keycloak Operator 更加建议，不会公开其中大多数字段。但是，仍然可以配置指定为 **podTemplate** 的任何所需的部署字段，例如：

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  unsupported:
    podTemplate:
      metadata:
        labels:
          foo: "bar"
      spec:
        containers:
          - volumeMounts:
```

```
            - name: test-volume
              mountPath: /mnt/test
      volumes:
        - name: test-volume
          secret:
            secretName: test-secret
...
```

注意

spec.unsupported.podTemplate 字段仅提供有限的支持，因为它会公开低级别配置，其中所有情况下尚未测试完整的功能。在可能的情况下，在 CR spec 的顶级使用完全支持的第一类保证字段。

例如，不使用 **spec.unsupported.podTemplate.spec.imagePullSecrets**，而是使用 **spec.imagePullSecrets**。

3.3.7. 不再支持连接到外部实例

Red Hat Single Sign-On 7.6 Operator 支持连接到 Red Hat Single Sign-On 7.6 的外部实例。例如，红帽构建的 Keycloak Operator 不再支持通过 Client CR 在现有域中创建客户端。

3.3.8. 迁移 Horizontal Pod Autoscaler 启用部署

要将 Horizontal Pod Autoscaler (HPA)与 Red Hat Single Sign-On 7.6 搭配使用，需要在 Keycloak CR 中设置 **disableReplicasSyncing: true** 字段并扩展 server StatefulSet。这不再需要，因为红帽构建的 Keycloak Operator 中的 Keycloak CR 可以直接由 HPA 扩展。

3.4. 迁移 KEYCLOAK 域 CR

Realm CR 被 Realm Import CR 替代，它提供类似的功能，并具有类似的模式。Realm Import CR 仅提供 Realm bootstrapping，因此不再支持 Realm deletion。它还不支持更新，类似于前面的 Realm CR。

完整的 Realm 表示现在包含在 Realm Import CR 中，与以前的 Realm CR 相比，它只提供了几个选择的字段。

Red Hat Single Sign-On 7.6 Realm CR 示例：

```
apiVersion: keycloak.org/v1alpha1
kind: KeycloakRealm
metadata:
  name: example-keycloakrealm
spec:
  instanceSelector:
    matchLabels:
      app: sso
  realm:
    id: "basic"
    realm: "basic"
    enabled: True
    displayName: "Basic Realm"
```

对应的红帽构建的 **Keycloak Realm Import CR 示例：**

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: KeycloakRealmImport
metadata:
  name: example-keycloakrealm
spec:
  keycloakCRName: example-kc
  realm:
    id: "basic"
    realm: "basic"
    enabled: True
    displayName: "Basic Realm"
```

其他资源

- realm Import

3.5. 删除的 CR

Client and User CR 从 Red Hat build of Keycloak Operator 中删除。缺少这些 CR 可能会被新的 Realm Import CR 部分缓解。为将来的红帽构建的 Keycloak 版本添加对 Client CR 的支持位于 road-map 中，而 User CR 当前并不是一个计划的功能。

第 4 章 在 OPENSHIFT 上迁移模板部署

OpenShift 模板已弃用，并从红帽构建的 Keycloak 容器镜像中删除。使用 Operator 是在 OpenShift 上部署红帽构建的 Keycloak 的替代选择。

注意

OpenShift 3.x 不再被支持。

通常，您需要创建一个引用外部管理数据库的 Keycloak CR （来自红帽构建的 Keycloak Operator）。带有此模板的 PostgreSQL 数据库由 DeploymentConfig 管理。您最初保留模板创建的 **application_name-postgresql** DeploymentConfig。DeploymentConfig 创建的 PostgreSQL 数据库实例将供红帽构建的 Keycloak Operator 使用。

本指南不包括从这个实例迁移到自我管理的数据库（由操作员或您的云供应商）迁移的指示。

Red Hat build of Keycloak Operator 不管理数据库，需要单独置备和管理数据库。

4.1. 使用内部 H2 数据库迁移部署

以下是受影响的模板：

- sso76-ocp3-https
- sso76-ocp4-https
- sso76-ocp3-x509-https
- sso76-ocp4-x509-https

这些模板依赖于 devel 数据库，且不支持在生产环境中使用。

4.2. 迁移带有临时 POSTGRESQL 数据库的部署

以下是受影响的模板：

- sso76-ocp3-postgresql
- sso76-ocp4-postgresql

此模板会创建一个没有持久性存储的 PostgreSQL 数据库，该数据库仅用于开发目的。

4.3. 迁移具有持久 POSTGRESQL 数据库的部署

以下是受影响的模板：

- sso76-ocp3-postgresql-persistent
- sso76-ocp4-postgresql-persistent
- sso76-ocp3-x509-postgresql-persistent
- sso76-ocp4-x509-postgresql-persistent

4.3.1. 先决条件

- 之前的 Red Hat Single Sign-On 7.6 实例被关闭，使其不使用红帽构建的 Keycloak 使用的数据库实例。
- 数据库备份已创建。
- 您已查看了 发行注记。

4.4. 迁移过程

1. 将红帽构建的 Keycloak Operator 安装到命名空间中。
2. 创建新 CR 和相关的 Secret。
   手动将基于 Red Hat Single Sign-On 7.6 配置的模板迁移到您的新 Red Hat build of Keycloak CR。有关 Template 参数和 Keycloak CR 字段之间的建议映射，请参阅以下示例。

以下示例将 Keycloak Operator CR 的 Red Hat build 与之前由 Red Hat Single Sign-On 7.6 模板创建的 DeploymentConfig 进行对比。

用于红帽构建的 Keycloak 的 Operator CR

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: rhbk
spec:
  instances: 1
  db:
    vendor: postgres
    host: postgres-db
    usernameSecret:
      name: keycloak-db-secret
      key: username
    passwordSecret:
      name: keycloak-db-secret
      key: password
  http:
    tlsSecret: sso-x509-https-secret
```

Red Hat Single Sign-On 7.6 的 DeploymentConfig

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
metadata:
  name: rhsso
spec:
  replicas: 1
  template:
    spec:
      volumes:
        - name: sso-x509-https-volume
          secret:
            secretName: sso-x509-https-secret
            defaultMode: 420
```

```
containers:
  volumeMounts:
    - name: sso-x509-https-volume
      readOnly: true
  env:
    - name: DB_SERVICE_PREFIX_MAPPING
      value: postgres-db=DB
    - name: DB_USERNAME
      value: username
    - name: DB_PASSWORD
      value: password
```

下表通过 JSON 路径表示法引用 Keycloak CR 的字段。例如，**.spec** 是指 **spec** 字段。请注意，**spec.unsupported** 是一个技术预览字段。更表明，功能最终将由其他 CR 字段实现。标记为 **粗体** 的参数由 passthrough 和 reencrypt 模板支持。

4.4.1. 常规参数迁移

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
APPLICATION_NAME	.metadata.name
IMAGE_STREAM_NAMESPACE	N/A - 镜像由 Operator 控制，或者您主要使用 spec.image 指定自定义镜像
SSO_ADMIN_USERNAME	没有直接设置，默认为 admin
SSO_ADMIN_PASSWORD	N/A - 在初始协调过程中由 Operator 创建
MEMORY_LIMIT	**.spec.unsupported.podTemplate.spec.containers[0].resources.limits['memory']**
SSO_SERVICE_PASSWORD, SSO_SERVICE_USERNAME	不再使用。
SSO_TRUSTSTORE, SSO_TRUSTSTORE_PASSWORD, SSO_TRUSTSTORE_SECRET	**.spec.truststores** Notice，信任存储不能受密码保护。
SSO_REALM	如果您要重复使用现有的数据库，则不需要使用。另一种方法是 RealmImport CR。

4.4.2. 数据库部署参数迁移

POSTGRESQL_IMAGE_STREAM_TAG,POSTGRESQL_MAX_CONNECTIONS,VOLUME_CAPACITY 和 POSTGRESQL_SHARED_BUFFERS 将需要迁移到您选择的数据库部署的任何替换中。

4.4.3. 数据库连接参数迁移

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
DB_VENDOR	如果 PostgreSQL 仍在使用，则需要将 **.spec.db.vendor** - 设置为 PostgreSQL
DB_DATABASE	**.spec.db.database**
DB_MIN_POOL_SIZE	**.spec.db.poolMinSize**
DB_MAX_POOL_SIZE	**.spec.db.maxPoolSize**
DB_TX_ISOLATION	如果驱动程序支持或作为目标数据库的常规设置，则 **spec.db.url** 可以被 spec.db.url 设置
DB_USERNAME	**.spec.db.usernameSecret**
DB_PASSWORD	**.spec.db.passwordSecret**
DB_JNDI	不再适用

4.4.4. 网络参数迁移

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
HOSTNAME_HTTP	**.spec.hostname.hostname** - with **.spec.http.httpEnabled=true.**因为 Red Hat build of Keycloak operator 将只创建一个 Ingress/Route，因此为了创建 http 路由 **.spec.http.tlsSecret**，需要未指定
HOSTNAME_HTTPS	**.spec.hostname.hostname** - with **.spec.http.tlsSecret** specified.
SSO_HOSTNAME	**.spec.hostname.hostname**
HTTPS_SECRET	**.spec.http.tlsSecret** - 请查看下面的其他 HTTPS 参数
HTTPS_KEYSTORE HTTPS_KEYSTORE_TYPE HTTPS_NAME HTTPS_PASSWORD	不再适用。**.spec.http.tlsSecret** 引用的 secret 应该类型为 **kubernetes.io/tls**，带有 **tls.crt** 和 **tls.key** 条目
X509_CA_BUNDLE	**.spec.truststores**

请注意，红帽构建的 Keycloak Operator 目前不支持配置 TLS 终止的方法。默认情况下使用 passthrough 策略。因此，代理选项还没有作为第一类的看法选项字段公开，因为它是否使用了 passthrough 或 reencrypt 策略。但是，如果需要这个选项，可以替换默认的 Ingress Operator 证书并手动配置 Route，以信任红帽构建的 Keycloak 证书。

然后，可以通过以下方法覆盖红帽构建的 Keycloak Operator 的默认行为：

```
additionalOptions:
  name: proxy
  value: reencrypt
```

4.4.5. JGroups 参数迁移

JGROUPS_ENCRYPT_SECRET、JGROUPS_ENCRYPT_KEYSTORE、JGROUPS_ENCRYPT_NAME、JGROUPS_ENCRYPT_PASSWORD 和 **JGROUPS_CLUSTER_PASSWORD** 在 Keycloak CR 中没有第一类表示。仍可使用缓存配置文件保护缓存通信。

其他资源

- 配置分布式缓存

第 5 章 迁移由 RED HAT SINGLE SIGN-ON 7.6 保护的应用程序

Red Hat build of Keycloak 引入了对应用程序使用一些 Red Hat Single Sign-On 7.6 Client Adapters 的关键更改。

除了不再释放一些客户端适配器外，红帽构建的 Keycloak 还引入了修复和改进，影响客户端应用程序如何使用 OpenID Connect 和 SAML 协议。

在本章中，您将找到解决这些更改的说明，并迁移应用程序以与红帽构建的 Keycloak 集成。

5.1. 迁移 OPENID CONNECT 客户端

从此 Red Hat build of Keycloak 开始，以下 Java Client OpenID Connect Adapters 不再发布

- Red Hat JBoss Enterprise Application Platform 6.x
- Red Hat JBoss Enterprise Application Platform 7.x
- Spring Boot
- Red Hat Fuse

与首次发布这些适配器相比，OpenID Connect 现在可在 Java 生态系统中广泛使用。此外，使用技术堆栈中提供的功能（如应用服务器或框架）可以实现更好的互操作性和支持。

这些适配器已结束其生命周期，且仅在 Red Hat Single Sign-On 7.6 中提供。强烈建议您寻找替代方案，以便应用程序使用 OAuth2 和 OpenID 连接协议的最新更新进行更新。

5.1.1. OpenID Connect 协议和客户端设置中的密钥更改

5.1.1.1. 访问类**型客**户**端**选项**不再可用**

当您创建或更新 OpenID Connect 客户端时，**Access Type** 不再可用。但是，您可以使用其他方法来实现此功能。

- 要达到 **Bearer Only** 功能，请创建一个没有身份验证流的客户端。在客户端详情的 **Capability config** 部分中，确保没有选择流。客户端无法从 Keycloak 获取任何令牌，这等同于使用 **Bearer Only** 访问类型。
- 要实现 **公共** 功能，请确保为此客户端禁用客户端身份验证，并且至少启用了一个流。
- 要实现 **机密** 功能，请确保 为**客户端** 启用客户端身份验证，并且至少启用了一个流。

客户端 JSON 对象上仍然存在 **bearerOnly** 和 **publicClient**。它们可用于在管理员 REST API 创建或更新客户端时使用，或者在通过部分导入或域导入时导入此客户端时使用。但是，这些选项不能直接在管理控制台 v2 中提供。

5.1.1.2. 验证**有效重定向 URI 的方案的更改**

如果应用程序客户端使用非 http (s)自定义方案，验证现在需要有效的重定向模式明确允许该方案。允许自定义方案的示例模式为 custom:/test, custom:/test decisions 或 custom:。为了安全起见，常规模式（如 * ）不再涵盖它们。

5.1.1.3. 支持 OpenID Connect Logout 端点中的 client_id 参数

支持 **client_id** 参数，它基于 OIDC RP-Initiated Logout 1.0 规格。此功能可用于检测在 **id_token_hint** 参数无法使用哪些客户端时用于 Post Logout Redirect URI 验证。当只使用了不带参数 **id_token_hint** 的情况下使用 **client_id** 参数时，才会向用户显示注销确认屏幕，因此如果他们不希望注销确认屏幕为用户显示 logout 确认屏幕，则鼓励客户端使用 **id_token_hint** 参数。

5.1.2. 有效的 Post Logout Redirect URI

Valid Post Logout Redirect URIs 配置选项添加到 OIDC 客户端，并与 OIDC 规格保持一致。您可以在登录和注销后，使用一组不同的重定向 URI 进行重定向。用于 **Valid Post Logout Redirect URIs** 的值 **+** 表示注销使用与 **Valid Redirect URIs** 选项的相同集合。由于向后兼容性，这个更改还与从以前的版本迁移时的默认行为匹配。

5.1.2.1. userinfo 端点更改

5.1.2.1.1. 错误响应更改

UserInfo 端点现在返回与 RFC 6750 完全兼容的错误响应(OAuth 2.0 Authorization Framework: Bearer Token Usage)。错误代码和描述（如果可用）以 **WWW-Authenticate** challenge 属性而不是 JSON 对象字段提供。

根据错误条件，响应如下：

- 如果没有提供访问令牌：

```
401 Unauthorized
WWW-Authenticate: Bearer realm="myrealm"
```

- 如果同时使用多种方法来提供访问令牌（如 Authorization 标头 + POST access_token 参数），或 POST 参数会被重复：

```
400 Bad Request
WWW-Authenticate: Bearer realm="myrealm", error="invalid_request", error_description="..."
```

- 如果访问令牌缺少 **openid** 范围：

```
403 Forbidden
WWW-Authenticate: Bearer realm="myrealm", error="insufficient_scope",
error_description="Missing openid scope"
```

- 如果无法解析 UserInfo 响应签名/加密的加密密钥：

```
500 Internal Server Error
```

- 如果是令牌验证错误，则返回 **401 Unauthorized** 与 **invalid_token** 错误代码。这个错误包括用户和客户端相关的检查，并实际捕获所有剩余的错误情况：

```
401 Unauthorized
WWW-Authenticate: Bearer realm="myrealm", error="invalid_token", error_description="..."
```

5.1.2.1.2. 对 UserInfo 端点的其他更改

现在，访问令牌需要具有 **openid** 范围，它由 UserInfo 做为特定于 OpenID Connect 的功能，而不是 OAuth 2.0。如果令牌中没有 **openid** 范围，则请求将被拒绝，为 **403 Forbidden**。请参阅上一节中。

userinfo 现在检查用户状态，并在用户被禁用时返回 **invalid_token** 响应。

5.1.2.1.3. 更改 Service Account 客户端的默认客户端 ID 映射器。

Service Account Client 的默认客户端 ID 映射程序已被更改。**令牌 Claim Name** 字段值已从 **clientId** 改为 **client_id**。**client_id** 声明符合 OAuth2 规格：

- OAuth 2.0 访问令牌的 JSON Web 令牌(JWT)配置文件
- OAuth 2.0 令牌内省
- OAuth 2.0 令牌交换

clientId userSession 备注仍然存在。

5.1.2.1.4. 向 OAuth 2.0/OpenID Connect Authentication Response 中添加的 iss 参数

RFC 9207 OAuth 2.0 Authorization Server Issuer Identification 规格在 OAuth 2.0/OpenID Connect Authentication Response 中添加了参数，以实现安全授权响应。

在以前的版本中，我们没有这个参数，但红帽构建的 Keycloak 默认会添加这个参数，如规格的要求。但是，一些 OpenID Connect / OAuth2 适配器（特别是旧的红帽构建的 Keycloak 适配器）可能会遇到这个新参数的问题。例如，在客户端应用成功身份验证后，参数始终存在于浏览器 URL 中。

在这些情况下，将 iss 参数添加到身份验证响应中可能会很有用。这可以在 Admin Console 中的客户端详情中针对带有 OpenID Connect 兼容性模式 的特定客户端完成。您可以启用 Exclude Issuer From Authentication Response，以防止将 iss 参数添加到身份验证响应中。

5.2. 迁移 RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 应用程序

5.2.1. Red Hat JBoss Enterprise Application Platform 8.x

您的应用程序不再需要任何其他依赖项来与红帽构建 Keycloak 或其他 OpenID 供应商集成。

相反，您可以利用 JBoss EAP 原生 OpenID Connect 客户端中的 OpenID Connect 支持。如需更多信息，请参阅 JBoss EAP 中的 OpenID Connect。

JBoss EAP 原生适配器依赖于与红帽构建的 Keycloak Adapter JSON 配置类似的配置模式。例如，使用 keycloak.json 配置文件的部署可以映射到 JBoss EAP 中的以下配置：

```
{
  "realm": "quickstart",
  "auth-server-url": "http://localhost:8180",
  "ssl-required": "external",
```

```
  "resource": "jakarta-servlet-authz-client",
  "credentials": {
    "secret": "secret"
  }
}
```

有关使用 JBoss EAP 原生适配器与红帽构建的 Keycloak 集成的示例，请参阅红帽构建的 Keycloak Quickstart 仓库示例：

- JAX-RS 资源服务器
- Servlet 应用程序

强烈建议迁移到 JBoss EAP 原生 OpenID Connect 客户端，因为它是部署到 JBoss EAP 8 及更新版本的 Jakarta 应用程序的最佳候选者。

5.2.2. Red Hat JBoss Enterprise Application Platform 7.x

因为 Red Hat JBoss Enterprise Application Platform 7.x 接近完全支持，因此红帽构建的 Keycloak 不会为其提供支持。对于部署到带有维护支持的 Red Hat JBoss Enterprise Application Platform 7.x 适配器的现有应用程序，可通过 Red Hat Single Sign-On 7.6 获得。

Red Hat Single Sign-On 7.6 适配器支持与 Red Hat build of Keycloak 24.0 服务器结合使用。

5.2.3. Red Hat JBoss Enterprise Application Platform 6.x

因为 Red Hat JBoss Enterprise Application PlatformJBoss EAP 6.x 已结束维护支持，所以 Red Hat Single Sign-On 7.6 或 Red Hat build of Keycloak 将为其提供支持。

5.3. 迁移 SPRING BOOT 应用程序

Spring Framework 生态系统正在快速演进，您应该通过利用 OpenID Connect 支持已在那里获得更好的体验。

您的应用程序不再需要任何其他依赖项来与红帽构建 Keycloak 或其他 OpenID 供应商集成，但依赖 Spring Security 中的全面的 OAuth2/OpenID Connect 支持。如需更多信息，请参阅 Spring Security 中的 OAuth2/OpenID Connect 支持。

就功能而言，它提供了一个基于标准的 OpenID Connect 客户端实现。可能要查看的功能示例（如果尚未使用标准协议）是 Logout。红帽构建的 Keycloak 提供对 OpenID Connect 生态系统中基于标准的退出协议的完全支持。

有关如何将 Spring Security 应用程序与红帽构建的 Keycloak 集成的示例，请参阅 Keycloak Quickstart Repository。

如果从 Spring Boot 的 Red Hat build of Keycloak Client Adapter 进行迁移，您仍可从 Red Hat Single Sign-On 7.6 访问适配器，它现在只支持维护。

Red Hat Single Sign-On 7.6 适配器支持与 Red Hat build of Keycloak 24.0 服务器结合使用。

5.4. 迁移 RED HAT FUSE 应用程序

因为 Red Hat Fuse 已结束完全支持，红帽构建的 Keycloak 24.0 不会为其提供任何支持。Red Hat Fuse 适配器仍可通过 Red Hat Single Sign-On 7.6 提供维护支持。

Red Hat Single Sign-On 7.6 适配器支持与 Red Hat build of Keycloak 24.0 服务器结合使用。

5.5. 使用授权服务策略强制迁移应用程序

为了支持与红帽构建的 Keycloak 授权服务集成，策略强制执行器与 Java 客户端适配器分开提供。

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-policy-enforcer</artifactId>
    <version>${Red Hat build of Keycloak .version}</version>
</dependency>
```

通过将其与 Java 客户端适配器分离，现在可以将红帽构建的 Keycloak 集成到为 OAuth2 或 OpenID Connect 提供内置支持的任何 Java 技术中。Red Hat build of Keycloak Policy Enforcer 为以下类型的应用程序提供内置支持：

- 使用细粒度授权的 servlet 应用

- 使用红帽构建的 Keycloak 授权服务的 Spring Boot REST Service Protected

对于红帽构建的 Keycloak Policy Enforcer 与不同类型的应用程序集成，请考虑以下示例：

- 使用细粒度授权的 servlet 应用
- 使用 Keycloak 授权服务进行 Spring Boot REST 服务保护

如果从您正在使用的 Red Hat Single Sign-On 7.6 Java Adapter 进行迁移，您仍可从 Red Hat Single Sign-On 7.6 访问适配器，它现在处于维护支持。

Red Hat Single Sign-On 7.6 适配器支持与 Red Hat build of Keycloak 24.0 服务器结合使用。

其他资源

- 策略强制器

5.6. 使用红帽构建的 KEYCLOAK JS ADAPTER 迁移单一页面应用程序(SPA)

要将使用 Red Hat Single Sign-On 7.6 适配器保护的应用程序迁移到红帽构建的 Keycloak 24.0，它提供最新版本的适配器。根据使用情况的方式，需要进行一些次要更改，如下所述。

5.6.1. 旧的 Promise API 被删除

在这个版本中，红帽构建的 Keycloak JS 适配器的传统 Promise API 方法已被删除。这意味着，不再可以在适配器返回的承诺中调用 .success （） 和 .error （）。

5.6.2. 需要使用新 Operator 实例化

在以前的版本中，当在没有新 Operator 的情况下构建 Red Hat build of Keycloak JS 适配器时，会记录弃用警告。从这个版本开始，这样做会抛出异常。这个更改是与 JavaScript 类的 预期行为保持一致，这将允许以后进一步重构适配器。

要将使用 Red Hat Single Sign-On 7.6 适配器保护的应用程序迁移到红帽构建的 Keycloak 24.0，它

提供最新版本的适配器。

5.7. 迁移 SAML 应用程序

5.7.1. 迁移 Red Hat JBoss Enterprise Application Platform 应用程序

5.7.1.1. Red Hat JBoss Enterprise Application Platform 8.x

红帽构建的 Keycloak 24.0 包括 Red Hat JBoss Enterprise Application Platform 8.x 的客户端适配器，包括支持 Jakarta EE。

5.7.1.2. Red Hat JBoss Enterprise Application Platform 7.x

因为 Red Hat JBoss Enterprise Application Platform 7.x 接近完全支持，因此红帽构建的 Keycloak 不会为其提供支持。对于部署到带有维护支持的 Red Hat JBoss Enterprise Application Platform 7.x 适配器的现有应用程序，可通过 Red Hat Single Sign-On 7.6 获得。

Red Hat Single Sign-On 7.6 适配器支持与 Red Hat build of Keycloak 24.0 服务器结合使用。

5.7.1.3. Red Hat JBoss Enterprise Application Platform 6.x

因为 Red Hat JBoss Enterprise Application PlatformJBoss EAP 6.x 已结束维护支持，所以 Red Hat Single Sign-On 7.6 或 Red Hat build of Keycloak 将为其提供支持。

5.7.2. SAML 协议和客户端设置中的主要变化

5.7.2.1. SAML SP 元数据更改

在此版本之前，SAML SP 元数据包含用于签名和加密使用的相同密钥。从 Keycloak 的这个版本开始，我们仅包含在 SP 元数据中使用的加密目标域密钥。对于每个加密密钥描述符，我们还指定应与之使用的算法。下表显示了带有红帽构建的 Keycloak 域键的受支持的 XML-Enc 算法。

xml-Enc 算法	realm 键算法
rsa-oaep-mgf1p	RSA-OAEP
rsa-1_5	RSA1_5

其他资源

- Keycloak 升级指南

5.7.2.2. 弃用了 SAML 的 RSA_SHA1 和 DSA_SHA1 算法

算法 RSA_SHA1 和 DSA_SHA1，它可以配置为 SAML 适配器、客户端和身份提供程序上的签名算法。我们建议使用基于 SHA256 或 SHA512 的安全替代方案。另外，使用这些算法在已签名的 SAML 文档或断言上验证签名无法在 Java 17 或更高版本上工作。如果您使用此算法，并且消耗 SAML 文档的其他方在 Java 17 或更高版本上运行，请验证签名将无法正常工作。

可能的解决方法是删除算法，如下所示：

- 列表中的 http://www.w3.org/2000/09/xmldsig#rsa-sha1 或 http://www.w3.org/2000/09/xmldsig#dsa-sha1

- "disallowed 算法"在文件 $JAVA_HOME/conf/security/java.security中配置了 jdk.xml.dsig.secureValidationPolicy

第 6 章 迁移自定义供应商

与 Red Hat Single Sign-On 7.6 类似，通过把自定义供应商复制到部署目录中，自定义供应商会部署到红帽构建的 Keycloak 中。在 Red Hat build of Keycloak 中，将供应商复制到 供应商 目录中，而不是独立/部署，这不再存在。另外，其他依赖项也应复制到 providers 目录中。

红帽构建的 Keycloak 不会将单独的类路径用于自定义供应商，因此您可能需要更小心地使用您包含的额外依赖项。此外，不再支持 EAR 和 WAR 打包格式和 jboss-deployment-structure.xml 文件。

虽然 Red Hat Single Sign-On 7.6 会自动发现自定义供应商，但在 Keycloak 运行时热部署自定义供应商的功能，但它不再被支持。另外，在更改 providers 目录中的供应商或依赖项后，您必须使用自动构建功能执行构建或重启服务器。

根据您的提供程序使用的 API，您可能需要对提供程序进行一些更改。详情请查看以下部分。

6.1. 从 JAVA EE 转换到 JAKARTA EE

Keycloak 将其代码库从 Java EE （企业版）迁移到 Jakarta EE，这会带来各种更改。我们已升级了所有 Jakarta EE 规格，以支持 Jakarta EE 10，例如：

- Jakarta Persistence 3.1
- Jakarta RESTful Web Services 3.1
- Jakarta Mail API 2.1
- Jakarta Servlet 6.0
- Jakarta Activation 2.1

Jakarta EE 10 提供了现代化、简化的轻量级方法来构建云原生 Java 应用程序。此计划中提供的主要更改是将命名空间从 javax configured 改为 jakarta ruby。这个更改不适用于 JDK 中直接提供的 javax targeted 软件包，如 javax.security、xx.net、xx.crypto 等。

另外，不再支持 Jakarta EE API （如 session/stateless Bean）。

6.2. 删除了第三方依赖项

红帽构建的 Keycloak 中删除了一些依赖项，包括

- openshift-rest-client
- okio-jvm
- okhttp
- commons-lang
- commons-compress
- jboss-dmr
- kotlin-stdlib

另外，因为红帽构建的 Keycloak 不再基于 EAP，因此大多数 EAP 依赖项已被删除。这个变化意味着，如果您使用这些库作为您自己的供应商的依赖项，则您可能还需要将这些 JAR 文件明确复制到 Keycloak 发布 供应商 目录中。

6.3. JAX-RS 资源不再启用上下文和依赖项注入

为了提供更好的运行时并尽可能利用底层堆栈，使用 javax.ws.rs.core.Context 注解的所有上下文数据注入点都会被移除。预期的性能改进涉及在请求生命周期内多次创建代理实例，并可在运行时减少反映代码的数量。

如果您需要访问当前请求和响应对象，您现在可以直接从 KeycloakSession 获取其实例：

```
@Context
org.jboss.resteasy.spi.HttpRequest request;
@Context
org.jboss.resteasy.spi.HttpResponse response;
```

被替换：

```
KeycloakSession session = // obtain the session, which is usually available when creating a custom provider from a factory
KeycloakContext context = session.getContext();

HttpRequest request = context.getHttpRequest();
HttpResponse response = context.getHttpResponse();
```

可以通过 KeycloakContext 实例从运行时获取其他上下文数据：

```
KeycloakSession session = // obtain the session
KeycloakContext context = session.getContext();
MyContextualObject myContextualObject =
context.getContextObject(MyContextualObject.class);
```

6.4. 弃用了来自数据供应商和模型的方法

现在，一些已弃用的方法已在 Red Hat build of Keycloak 中删除：

- RealmModel SerialsearchForGroupByNameStream (String, Integer, Integer)
- UserProvider"getUsersStream (RealmModel, boolean)
- UserSessionPersisterProvider SerialloadUserSessions (int, int, boolean, int, String)
- 为流化添加的接口，如 RoleMapperModel.Streams 和类似
- KeycloakModelUtils#getClientScopeMappings
- 弃用了 KeycloakSession的方法

- UserQueryProviderRHACMgetUsersStream 方法

另外，还会进行这些其他更改：

- 来自 UserSessionProvider 的一些方法被移到 UserLoginFailureProvider 中。
- 联合存储供应商类中的流接口已弃用。
- Streamification - 接口现在只包含基于流的方法。

  例如在 GroupProvider 接口中

  ```
  @Deprecated
  List<GroupModel> getGroups(RealmModel realm);
  ```

  被替换

  ```
  Stream<GroupModel> getGroupsStream(RealmModel realm);
  ```

- 一致参数排序 - 方法现在具有严格的参数排序，其中 RealmModel 始终是第一个参数。

  例如，在 UserLookupProvider 接口：

  ```
  @Deprecated
  UserModel getUserById(String id, RealmModel realm);
  ```

  被替换

  ```
  UserModel getUserById(RealmModel realm, String id)
  ```

6.4.1. 更改的接口列表

(p.k. 代表 org.keycloak. 软件包)

- **server-spi 模块**
  - **o.k.credential.CredentialInputUpdater**
  - **o.k.credential.UserCredentialStore**
  - **o.k.models.ClientProvider**
  - **o.k.models.ClientSessionContext**
  - **o.k.models.GroupModel**
  - **o.k.models.GroupProvider**
  - **o.k.models.KeyManager**
  - **o.k.models.KeycloakSessionFactory**
  - **o.k.models.ProtocolMapperContainerModel**
  - **o.k.models.RealmModel**
  - **o.k.models.RealmProvider**
  - **o.k.models.RoleContainerModel**
  - **o.k.models.RoleMapperModel**
  -

- o.k.models.RoleModel
- o.k.models.RoleProvider
- o.k.models.ScopeContainerModel
- o.k.models.UserCredentialManager
- o.k.models.UserModel
- o.k.models.UserProvider
- o.k.models.UserSessionProvider
- o.k.models.utils.RoleUtils
- o.k.sessions.AuthenticationSessionProvider
- o.k.storage.client.ClientLookupProvider
- o.k.storage.group.GroupLookupProvider
- o.k.storage.user.UserLookupProvider
- o.k.storage.user.UserQueryProvider

- server-spi-private module

- o.k.events.EventQuery
- o.k.events.admin.AdminEventQuery
- o.k.keys.KeyProvider

6.4.2. 在存储层中重新考虑

Red Hat build of Keycloak 进行大型重构，以简化 API 的使用，这会影响现有代码。其中一些更改需要更新现有代码。以下部分详细介绍了。

6.4.2.1. 模块结构的更改

围绕 KeycloakSession 中的存储功能的几个公共 API 已合并，一些已被移动、弃用或删除。引入了三个新模块，并且从 server-spi、 server-spi -private 和 services 模块中面向数据的代码已移动：

org.keycloak:keycloak-model-legacy

包含来自旧存储的所有面向公共的 API，如 User Storage API。

org.keycloak:keycloak-model-legacy-private

包含与用户存储管理相关的私有实现，如 storage *Manager 类。

org.keycloak:keycloak-model-legacy-services

包含所有直接在传统存储上操作的 REST 端点。

如果您在自定义用户存储供应商中使用示例，则实施已移至新模块的类，您需要更新依赖项使其包含上面列出的新模块。

6.4.2.2. KeycloakSession中的更改

KeycloakSession 已简化。KeycloakSession 中删除了多种方法。

KeycloakSession 会话包含多种获取特定对象类型的提供程序的方法，如 UserProvider，有 usersProvider、userLocalStorage （） 、userCache （）、userCache

（）、userStorageManager （） 和 userFederatedStorage （）。对于必须了解每种方法完全相同的含义的开发人员，这种情形可能会造成混淆。

因此，只有 users （） 方法保存在 KeycloakSession 中，并且应替换上面列出的所有其他调用。删除了其余方法。相同的模式适用于其他对象区域的方法，如 clients （） 或 groups （）。以 *StorageManager （） 和 *LocalStorage （） 结尾的所有方法均已被删除。下面的部分论述了如何将这些调用迁移到新 API，或使用旧的 API。

6.4.3. 迁移现有供应商

如果没有调用删除的方法，则现有供应商不需要迁移，这应该是大多数供应商的情况。

如果提供商使用删除的方法，但不依赖于本地存储与非本地存储，则将调用从现在删除的 userLocalStorage （） 更改为方法 users （） 是最佳选择。请注意，如果本地设置中已启用，则语义更改在此处，因为新方法涉及缓存。

在迁移前：访问删除的 API 不会编译

```
session.userLocalStorage();
```

迁移后： 当调用者不依赖于旧的存储 API 时访问新的 API

```
session.users();
```

在个别情况下，当自定义供应商需要区分特定提供程序的模式时，使用 LegacyStoreManagers 数据存储提供程序提供对已弃用对象的访问。如果提供商直接访问本地存储或希望跳过缓存，则可能会出现这种情况。只有在旧模块是部署的一部分时，此选项才可用。

在迁移前：访问已删除的 API

```
session.userLocalStorage();
```

迁移后：通过 LegacyStoreManagers API 访问新功能

```
((LegacyDatastoreProvider)
session.getProvider(DatastoreProvider.class)).userLocalStorage();
```

为方便起见，一些用户存储相关的 API 已被嵌套在 org.keycloak.storage.UserStorageUtil 中。

6.4.4. 更改为 RealmModel

方法 getUserStorageProviders,getUserStorageProvidersStream,getClientStorageProviders,getClientStorageProvidersStream,getRoleStorageProviders Streams 和 getRoleStorageProvidersStream 已被删除。依赖于这些方法的代码应广播实例，如下所示：

迁移前：因为更改的 API，代码不会编译

```
realm.getClientStorageProvidersStream()...;
```

迁移后：将实例转换为旧接口

```
((LegacyRealmModel) realm).getClientStorageProvidersStream()...;
```

同样，用于实现接口 RealmModel 并希望提供这些方法的代码应该实施新的接口

LegacyRealmModel。这个接口是 RealmModel 的子接口，包括旧方法：

迁移前：代码实现旧接口

```
public class MyClass extends RealmModel {
    /* might not compile due to @Override annotations for methods no longer present
       in the interface RealmModel. / / ... */
}
```

迁移后：代码实现新接口

```
public class MyClass extends LegacyRealmModel {
    /* ... */
}
```

6.4.5. Interface UserCache 移到旧的模块

由于对象的缓存状态对服务是透明的，因此接口 UserCache 已移到模块 keycloak-model-legacy 中。

依赖于传统实施的代码应直接访问 UserCache。

在迁移前：代码不会编译[source,java,subs="+ quote"]

```
session**.userCache()**.evict(realm, user);
```

迁移后：直接使用 API

```
UserStorageUitl.userCache(session);
```

要触发域的无效，而不是使用 UserCache API，请考虑触发事件：

在迁移前：代码使用 cache API[source,java,subs="+ quotes"]

```
UserCache cache = session.getProvider(UserCache.class);
if (cache != null) cache.evict(realm)();
```

迁移后：使用无效 API

```
session.invalidate(InvalidationHandler.ObjectType.REALM, realm.getId());
```

6.4.6. 用户的凭证管理

用户凭证之前使用 session.userCredentialManager （）.*方法*(realm、user、...) 进行管理。新方法是利用 user.credentialManager （）.*方法*(...)。这种形式使凭证功能更接近用户 API，不依赖于用户凭据对域和存储相关的位置。

旧的 API 已被移除。

在迁移前：访问已删除的 API

```
session.userCredentialManager().createCredential(realm, user, credentialModel)
```

迁移后：访问新的 API

```
user.credentialManager().createStoredCredential(credentialModel)
```

对于自定义 UserStorageProvider，有一个新的方法 credentialManager （），需要在返回 UserModel 时实施。它们必须返回 LegacyUserCredentialManager 实例：

迁移前：因为 UserModel需要的新方法 credentialManager （），代码不会被编译

```
public class MyUserStorageProvider implements UserLookupProvider, ... {
   /* ... */
   protected UserModel createAdapter(RealmModel realm, String username) {
      return new AbstractUserAdapter(session, realm, model) {
         @Override
         public String getUsername() {
            return username;
         }
      };
   }
}
```

迁移后：为传统存储实施 API UserModel.credentialManager （）。

```
public class MyUserStorageProvider implements UserLookupProvider, ... {
   /* ... */
   protected UserModel createAdapter(RealmModel realm, String username) {
      return new AbstractUserAdapter(session, realm, model) {
         @Override
         public String getUsername() {
            return username;
         }

         @Override
         public SubjectCredentialManager credentialManager() {
            return new LegacyUserCredentialManager(session, realm, this);
         }
```

```
        };
    }
}
```

第 7 章 迁移自定义主题

7.1. 新的管理控制台

新的 Admin Console (keycloak.v2)使用 React 构建。旧的 Admin Console (keycloak)使用 AngularJS 1.x 构建，它在前期到期了生命周期结束。因此，没有从旧控制台或扩展它的任何主题的迁移路径。由于相同原因，也不支持基础主题管理控制台。

7.2. 新帐户控制台

新帐户控制台(keycloak.v2)使用 React 构建，从而提供更好的用户体验。旧帐户控制台(keycloak)使用基本的服务器端模板构建。因此，没有从旧控制台或扩展它的任何主题的迁移路径。

7.3. 迁移登录主题

主题用于配置登录页面和帐户控制台的外观和感觉。

在创建或更新自定义主题时，特别是在覆盖模板时，使用内置模板作为参考可能很有用。这些模板位于 ${KC_HOME}/lib/lib/main/org.keycloak.keycloak-themes-${KC_VERSION}.jar 中，可以使用任何标准 ZIP 存档工具打开。

在使用 start-dev 在开发模式下运行服务器时，它们不会被缓存，以便您可以轻松处理它们，而无需在进行更改时重新启动服务器。

要安装自定义主题，您可以选择将主题文件打包为 JAR 文件，并将其部署到 ${KC_HOME}/providers 目录，或者将文件复制到 ${KC_HOME}/themes 目录。在这两种情况下，请参阅 服务器预期的文件和目录结构的更多详情，请参阅服务器开发人员指南。

第 8 章 将上游 KEYCLOAK 迁移到红帽构建的 KEYCLOAK 24.0

从版本 22 开始，红帽构建的 Keycloak 和上游 Keycloak 之间存在最小差异。存在以下区别：

- 对于上游 Keycloak，分发工件位于 keycloak.org 上；对于红帽构建的 Keycloak，分发工件位于 红帽客户门户网站中。
- Oracle 和 MSSQL 数据库驱动程序与上游 Keycloak 捆绑，但不与红帽构建的 Keycloak 捆绑。有关如何安装这些驱动程序的详细信息，请参阅配置 数据库。
- Red Hat build of Keycloak 不提供 GELF 日志处理程序。

迁移过程取决于要迁移的 Keycloak 版本以及 Keycloak 安装的类型。详情请查看以下部分。

8.1. 匹配 KEYCLOAK 版本

迁移过程取决于要迁移的 Keycloak 版本。

- 如果您的 Keycloak 项目版本与红帽构建的 Keycloak 版本匹配，请使用红帽客户门户网站上的红帽构建 Keycloak 工件来迁移 Keycloak。https://access.redhat.com/products/red-hat-build-of-keycloak
- 如果您的 Keycloak 项目版本是旧版本，请使用 Keycloak 升级指南 来升级 Keycloak，以匹配红帽构建的 Keycloak 版本。然后，使用 红帽客户门户网站中的工件迁移 Keycloak。
- 如果您的 Keycloak 项目版本大于红帽构建的 Keycloak 版本，则无法迁移到红帽构建的 Keycloak。相反，创建一个红帽构建的 Keycloak 的新部署，或等待将来红帽构建的 Keycloak 版本。

8.2. 根据 KEYCLOAK 安装类型进行迁移

匹配的 Keycloak 版本后，根据安装类型迁移 Keycloak。

- 如果您从 ZIP 发行版安装 Keycloak，请使用红帽客户门户网站中的工件迁移

Keycloak。https://access.redhat.com/products/red-hat-build-of-keycloak

- 如果部署了 Keycloak Operator，请卸载它并使用 Operator 指南 安装 Keycloak Operator 的 Red Hat build。CR 在上游 Keycloak 和 Red Hat build of Keycloak 之间兼容。

- 如果您创建了自定义服务器容器镜像，请使用红帽构建的 Keycloak 镜像重建它。请参阅 在容器中运行 Keycloak。

第 9 章 其他显著变化

9.1. CLASSPATH 上默认提供的 JAVASCRIPT 引擎

在以前的版本中，当 Keycloak 用于带有 Javascript 供应商的 Java 17 时(Script authenticator, Javascript 授权策略或 OIDC 和 SAML 客户端的脚本协议映射程序)，需要将 javascript 引擎复制到分发中。默认情况下，Red Hat build of Keycloak 服务器提供了 Nashorn javascript 引擎，所以不再需要它。当您部署脚本供应商时，建议不要将 Nashorn 的脚本引擎及其依赖项复制到红帽构建的 Keycloak 分发中。

9.2. 重命名 KEYCLOAK ADMIN 客户端工件

升级到 Jakarta EE 后，Keycloak Admin 客户端的工件被重命名为更描述性的名称，并考虑长期可维护。但是，仍然存在两个单独的 Keycloak Admin 客户端：一个带有 Jakarta EE，另一个支持 Java EE。

org.keycloak:keycloak-admin-client-jakarta 工件不再被释放。带有 Jakarta EE 支持的 Keycloak Admin 客户端的默认值为 org.keycloak:keycloak-admin-client （自 24.0.0 开始）。

Java EE 支持的新工件是 org.keycloak:keycloak-admin-client-jee。

9.2.1. Jakarta EE 支持

Java EE 支持的新工件是 org.keycloak:keycloak-admin-client-jee。Jakarta EE 支持

迁移前：

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client-jakarta</artifactId>
    <version>18.0.0.redhat-00001</version>
</dependency>
```

迁移后：

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client</artifactId>
    <version>22.0.0.redhat-00001</version>
</dependency>
```

9.2.2. Java EE 支持

迁移前：

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client</artifactId>
    <version>18.0.0.redhat-00001</version>
</dependency>
```

迁移后：

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client-jee</artifactId>
    <version>22.0.0.redhat-00001</version>
</dependency>
```

9.3. 从客户端高级设置组合中删除过期选项

现在，选项 Never expires 已从 Advanced Settings 客户端选项卡的所有组合中删除。这个选项有误导，因为不同的生命周期或空闲超时永远不会是无限的，但受一般的用户会话或域值的限制。因此，这个选项被移除为另外两个剩余的选项： 来自 realm 设置中的 Inherits （客户端使用常规域超时）和 Expires （客户端的值会被覆盖）。Never 在内部过期 由 -1 表示。现在，该值显示在 Admin Console 中，且无法由管理员直接设置。

9.4. 新的电子邮件规则和限制验证

Red Hat build of Keycloak 在电子邮件创建时具有新规则，允许在电子邮件创建过程中允许 ASCII 字符。此外，本地电子邮件部分(@ 之前)上也存在 64 个字符的新限制。因此，添加了一个新的 parameter -spi-user-profile-declarative-user-profile-max-email-local-part-length，以设置最大电子邮件本地部分长度，以便向后兼容。默认值为 64。

```
kc.sh start --spi-user-profile-declarative-user-profile-max-email-local-part-length=100
```