Red Hat build of Keycloak 24.0

服务器指南

Last Updated: 2024-06-28

法律通告

摘要

本指南包含管理员配置红帽构建的 Keycloak 24.0 的信息。

目录

第 1 章 配置红帽构建的 KEYCLOAK

本章介绍了红帽构建的 Keycloak 配置方法，以及如何启动并应用首选配置。它包括优化红帽构建的 Keycloak 的配置指南，以便更快地启动和较少的内存占用。

1.1. 为红帽构建的 KEYCLOAK 配置源

红帽构建的 Keycloak 从四个源加载配置，这些配置按应用程序顺序列出。

1. 命令行参数
2. 环境变量
3. 在 **conf/keycloak.conf** 文件中定义的选项，或在用户创建的配置文件中定义。
4. 用户创建的 Java KeyStore 文件中定义的敏感选项。

在多个源中设置某个选项时，列表中的第一个选项决定了该选项的值。例如，命令行参数设置的选项值的优先级高于同一选项的环境变量。

1.1.1. 示例：配置 db-url-host 参数

以下示例演示了如何在四个配置源中设置 **db-url** 值：

Source	格式
命令行参数	**--db-url=cliValue**
环境变量	**KC_DB_URL=envVarValue**
配置文件	**db-url=confFileValue**
Java KeyStore 文件	**kc.db-url=keystoreValue**

根据应用程序的优先级，启动时使用的值是 **cliValue**，因为命令行是最高优先级。

如果没有使用 **--db-url=cliValue**，应用的值将是 **KC_DB_URL=envVarValue**。如果该值未被命令行或环境变量应用，则将使用 **db-url=confFileValue**。如果没有应用以上值，则将使用 **kc.db-url=confFileValue** 值，因为可用配置源中的最低优先级。

1.2. 配置格式

配置使用 *统一源* 格式，简化了从一个配置源到另一个配置源的键/值对转换。请注意，这些格式也适用于 spi 选项。

命令行参数格式

命令行的值使用 **-- *<key-with-dashes> = <value>*** 格式。对于某些值，也存在一个 **- *<abbreviation>= <value&*** gt; 简写。

环境变量格式

环境变量的值使用大写的 **KC_ *<key_with_underscores> = <value>*** 格式。

配置文件格式

进入配置文件的值使用 < ***key-with-dashes> = <value>*** 格式。

密钥存储配置文件格式

进入 KeyStore 配置文件的值使用 **kc. <*key-with-dashes*>** 格式。然后， ***<value*** > 是存储在 KeyStore 中的密码。

在每个配置章节的末尾，查看 *相关选项*标题，用于定义适用的配置格式。有关所有配置选项，请查看 所有配置。选择适用于您的用例的配置源和格式。

1.2.1. 示例 - 基于配置源的替代格式

以下示例显示了三个配置源的 **db-url-host** 的配置格式：

命令行参数

```
bin/kc.[sh|bat] start --db-url-host=mykeycloakdb
```

环境变量

```
export KC_DB_URL_HOST=mykeycloakdb
```

conf/keycloak.conf

```
db-url-host=mykeycloakdb
```

1.2.2. 命令行参数的格式

红帽构建的 Keycloak 使用许多命令行参数进行打包。要查看可用的配置格式，请输入以下命令：

```
bin/kc.[sh|bat] start --help
```

或者，查看所有服务器选项的所有配置。???

1.2.3. 环境变量的格式

您可以使用 **${ENV_VAR}** 语法从 **keycloak.conf** 文件中的环境变量解析环境特定值：

```
db-url-host=${MY_DB_HOST}
```

如果环境变量无法解析，您可以指定回退值。在 **mydb** 之前使用 **:** (colon)：

```
db-url-host=${MY_DB_HOST:mydb}
```

1.2.4. 包含特定配置文件的格式

默认情况下，服务器始终从 **conf/keycloak.conf** 文件中获取配置选项。对于新安装，此文件仅包含注释的设置，作为在生产环境中运行时要设置的想法。

您还可以输入以下命令使用 **[-cf|--config-file]** 选项指定显式配置文件位置：

```
bin/kc.[sh|bat] --config-file=/path/to/myconfig.conf start
```

设置该选项可让红帽从指定的文件而不是 **conf/keycloak.conf** 读取配置。

1.2.5. 使用 Java KeyStore 文件设置敏感选项

由于 Keystore Configuration Source，您可以使用 **[--config-keystore]** 和 **[--config-keystore-password]** 选项直接从 Java KeyStore 加载属性。另外，您可以使用 **[--config-keystore-type]** 选项指定 KeyStore 类型。默认情况下，KeyStore 类型是 **PKCS12**。

KeyStore 中的 secret 需要使用 **PBE** （基于密码的加密）密钥算法存储，其中密钥从 KeyStore 密码衍生而来。您可以使用以下 **keytool** 命令生成这种 KeyStore：

```
keytool -importpass -alias kc.db-password -keystore keystore.p12 -storepass keystorepass -
storetype PKCS12 -v
```

执行该命令后，系统将提示您输入要 **存储的密码，这表示** 上面的 **kc.db-password** 属性的值。

创建 KeyStore 时，您可以使用以下参数启动服务器：

```
bin/kc.[sh|bat] start --config-keystore=/path/to/keystore.p12 --config-keystore-password=storepass --
config-keystore-type=PKCS12
```

1.2.6. raw Quarkus 属性的格式

在大多数情况下，可用的配置选项应该可以满足配置服务器。但是，对于红帽构建的 Keycloak 配置中缺少的特定行为或功能，您可以使用底层 Quarkus 框架中的属性。

如果可能，请直接使用 Quarkus 的属性，因为它们不受红帽构建的 Keycloak 不支持。如果需要，请考虑首先打开 增强请求。这个方法有助于改进红帽构建的 Keycloak 配置以满足您的需要。

如果无法进行增强请求，您可以使用原始 Quarkus 属性配置服务器：

1. 在 **conf** 目录中创建 **quarkus.properties** 文件。
2. 在该文件中定义必要属性。
   您只能使用 Quarkus 文档 中定义的 Quarkus 扩展 子集。另外，请注意 Quarkus 属性的这些区别：
   - Quarkus 文档中的 Quarkus 属性的锁定图标表示构建时间属性。您可以运行 **build** 命令来应用此属性。有关 build 命令的详情，请参阅有关优化红帽构建的 Keycloak 的后续部分。
   - Quarkus 指南中的属性没有锁定图标表示 Quarkus 和 Red Hat build of Keycloak 的运行时属性。
3. 使用 **[-cf|--config-file]** 命令行参数来包括该文件。

同样，您还可以将 Quarkus 属性存储在 Java KeyStore 中。

请注意，一些 Quarkus 属性已在红帽构建的 Keycloak 配置中映射，如 **quarkus.http.port** 和类似基本属性。如果红帽构建的 Keycloak 使用了属性，在 **quarkus.properties** 中定义该属性键无效。红帽构建的 Keycloak 配置值优先于 Quarkus 属性值。

1.2.7. 在值中使用特殊字符

红帽构建的 Keycloak 依赖于 Quarkus 和 MicroProfile 来处理配置值。请注意，支持值表达式。例如，**${some_key}** 评估为 **some_key** 的值。

要禁用表达式评估，\ 字符充当转义字符。特别是，它必须用于在出现定义表达式或重复时转义 **$** 字符的使用。例如，如果您希望配置值 **my$password**，请改为使用 **my\$\$password**。请注意，在使用大多数 unix shell 或其出现在属性文件中时，\ 字符需要额外的转义或引用。例如，bash 单引号保留单个反斜杠 **--db-password='my\$\$password'**。另外，使用 bash 双引号时，您需要额外的反斜杠 **--db-password="my\\$\\$password"**。在属性文件中，还必须转义反斜杠字符：kc **.db-password=my\\$\\$password**

1.3. 启动红帽构建的 KEYCLOAK

您可以使用 **开发模式 或生产模式** 启动红帽 Keycloak 的构建。每个模式为预期的环境提供不同的默认值。

1.3.1. 以开发模式启动红帽构建的 Keycloak

使用开发模式第一次尝试红帽构建的 Keycloak，以快速启动并运行。此模式为开发人员提供方便的默认值，例如用于开发新的 Keycloak 主题构建。

要在开发模式下启动，请输入以下命令：

```
bin/kc.[sh|bat] start-dev
```

默认值

开发模式设置以下默认配置：

- HTTP 已启用
- 禁用严格的主机名解析
- 缓存设置为 local （不用于高可用性的分布式缓存机制）
- 禁用了主题缓存和模板缓存

1.3.2. 以生产环境模式启动红帽构建的 Keycloak

将 production 模式用于在生产环境中部署红帽构建的 Keycloak。这个模式 *默认是安全的*。

要在生产环境模式下启动，请输入以下命令：

```
bin/kc.[sh|bat] start
```

如果没有进一步配置，此命令将不会启动红帽构建的 Keycloak，并向您显示错误。此响应的目的是进行，因为红帽构建的 Keycloak *默认遵循一个安全* 原则。Production 模式需要设置主机名，并且启动时可以使用 HTTPS/TLS 设置。

默认值

Production 模式设置以下默认值：

- HTTP 被禁用，因为传输层安全(HTTPS)至关重要
- 主机名配置是预期的

- 预期为 HTTPS/TLS 配置

在生产环境中部署红帽构建的 Keycloak 之前，请确保按照为生产环境 配置红帽构建的 Keycloak 中所述的步骤进行。

默认情况下，生产模式的配置选项示例在默认的 **conf/keycloak.conf** 文件中被注释掉。这些选项可让您了解在生产环境中运行红帽构建的 Keycloak 时需要考虑的主要配置。

1.4. 创建初始 ADMIN 用户

您可以使用 Web frontend 创建初始 admin 用户，您可以使用本地连接(localhost)访问该前端。您可以使用环境变量创建此用户。为初始 admin ***用户名设置 KEYCLOAK_ADMIN= &*** lt;username>，为初始 admin ***密码设置 KEYCLOAK_ADMIN_PASSWORD= &*** lt;password>。

红帽构建的 Keycloak 会在第一次启动时解析这些值，以创建具有管理权限的初始用户。存在具有管理权限的第一个用户后，您可以使用 Admin Console 或命令行工具 **kcadm.[sh|bat]** 来创建其他用户。

如果初始管理员已存在并且环境变量在启动时仍然存在，日志中会显示一条错误消息，指出初始管理员创建失败。Red Hat build of Keycloak 会忽略值并正确启动。

1.5. 优化红帽构建的 KEYCLOAK 启动

我们建议优化红帽构建的 Keycloak，以便在生产环境中部署红帽构建的 Keycloak 前提供更快的启动和更好的内存消耗。本节论述了如何应用红帽构建的 Keycloak 优化，以获得最佳性能和运行时行为。

1.5.1. 创建经过优化的红帽构建的 Keycloak 构建

默认情况下，当使用 **start** 或 **start-dev** 命令时，Red Hat build of Keycloak 会在覆盖的原因下运行一个 **build** 命令。

此 **build** 命令为启动和运行时行为执行一组优化。构建过程可能需要几秒钟时间。特别是在容器化环境中运行 Keycloak 时，如 Kubernetes 或 OpenShift，启动时间非常重要。为避免丢失这一时间，请在启动前明确 运行构建，如 CI/CD 管道中的单独步骤。

1.5.1.1. 第一步：明确运行构建

要运行 **构建**，请输入以下命令：

```
bin/kc.[sh|bat] build <build-options>
```

此命令显示 **您输入的构建选项**。红帽构建的 Keycloak 可区分 **构建选项**，这些选项 可在运行构建命令时以及启动服务器时可用的配置选项。

对于红帽构建的 Keycloak 的非优化启动，这种区别无效。但是，如果您在启动前运行构建，则 build 命令只能使用一组选项。这个限制是因为构建选项被保留到优化的红帽构建的 Keycloak 镜像中。例如，出于安全原因，不能保留 db-password （这是配置选项）等凭据的配置。

警告

所有构建选项都以纯文本形式保留。不要将任何敏感数据存储为构建选项。这适用于所有可用的配置源，包括 KeyStore Config Source。因此，我们还不建议将任何构建选项存储在 Java 密钥存储中。另外，当涉及配置选项时，我们建议使用 KeyStore Config Source 来存储敏感数据。对于非敏感数据，您可以使用剩余的配置源。

构建选项在 All configuration 中标有工具图标。要查找可用的构建选项，请输入以下命令：

```
bin/kc.[sh|bat] build --help
```

示例： 在启动前 运行构建将数据库设置为 PostgreSQL

```
bin/kc.[sh|bat] build --db=postgres
```

1.5.1.2. 第二步： 使用 --optimized启动红帽构建的 Keycloak

构建成功后，您可以启动红帽构建的 Keycloak 并输入以下命令关闭默认的启动行为：

```
bin/kc.[sh|bat] start --optimized <configuration-options>
```

--optimized 参数告知红帽构建的 Keycloak 假设已使用已优化的红帽构建的 Keycloak 镜像。因此，红帽构建的 Keycloak 避免在启动时检查并运行构建，从而节省时间。

您可以在启动时输入所有配置选项；这些选项是所有未标记为工具图标的配置中的选项。???

- 如果在启动时找到构建选项，其值等于输入 构建 时使用的值，则该选项会在使用 --optimized 参数时静默忽略。
-

如果该选项的值与输入构建时使用的值不同，日志中会出现一个警告，并使用之前构建的值。要使这个值生效，请在启动前运行新构建。

创建优化的构建

以下示例显示了在启动红帽构建的 Keycloak 时创建优化的构建，然后使用 --optimized 参数。

1. 使用 build 命令为 PostgreSQL 数据库供应商设置构建选项

```
bin/kc.[sh|bat] build --db=postgres
```

2. 在 conf/keycloak.conf 文件中设置 postgres 的运行时配置选项。

```
db-url-host=keycloak-postgres
db-username=keycloak
db-password=change_me
hostname=mykeycloak.acme.com
https-certificate-file
```

3. 使用优化参数启动服务器

```
bin/kc.[sh|bat] start --optimized
```

您可以使用 build 命令，实现对启动和运行时行为的大多数优化。另外，通过使用 keycloak.conf 文件作为配置源，您可以避免在启动时需要命令行参数的一些步骤，如初始化 CLI 本身。因此，服务器可以更快地启动。

1.6. 底层概念

本节概述了红帽构建的 Keycloak 使用的底层概念，特别是在优化启动时。

红帽构建的 Keycloak 使用 Quarkus 框架，以及涵盖以下内容下的 re-augmentation/mutable-jar 方法。运行 build 命令时会启动此过程。

以下是 build 命令执行的一些优化：

- 创建一个新的有关已安装提供程序的关闭假设，这意味着不需要重新创建 registry，并在每次

红帽构建的 Keycloak 启动时初始化工厂。

- 配置文件是预先解析的，以在启动服务器时减少 I/O。
- 配置数据库特定资源，并准备好针对特定数据库供应商运行。
- 通过将构建选项持久化到服务器镜像中，服务器不会执行任何其他步骤来解释配置选项和（重新）配置自己。

您可以在特定的 Quarkus 指南中了解更多信息

第 2 章 为生产环境配置红帽构建的 KEYCLOAK

红帽构建的 Keycloak 生产环境为内部部署提供安全身份验证和授权，这些部署支持几千位用户来为数百万用户提供服务。

本章论述了生产环境就绪的 Keycloak 环境所需的配置的一般区域。这些信息侧重于常规概念，而不是具体实施，这取决于您的环境。本章涵盖的关键方面适用于所有环境，无论是容器化、内部部署、GitOps 或 Ansible。

2.1. TLS 用于安全通信

红帽构建的 Keycloak 持续交换敏感数据，这意味着所有与红帽构建的 Keycloak 通信都需要一个安全的通信频道。要防止一些攻击向量，您可以为该频道启用 HTTP over TLS 或 HTTPS。

要为红帽构建的 Keycloak 配置安全通信频道，请参阅配置 TLS 并配置 传出 HTTP 请求。

要保护红帽构建的 Keycloak 缓存通信，请参阅配置分布式缓存。

2.2. 红帽构建的 KEYCLOAK 的主机名

在生产环境中，红帽构建的 Keycloak 实例通常在专用网络中运行，但红帽构建的 Keycloak 需要公开某些面向公共的端点以便与要保护的应用程序通信。

有关端点类别的详情，以及如何为其配置公共主机名，请参阅 配置主机名。

2.3. 分布式环境中的反向代理

除了 配置主机名外，生产环境通常包括反向代理/负载均衡器组件。它分离并统一访问公司或组织所使用的网络。对于红帽构建的 Keycloak 生产环境，建议此组件。

有关在红帽构建的 Keycloak 中配置代理通信模式的详情，请参阅使用反向代理。本章还建议，在公共访问中隐藏哪些路径，以及应公开哪些路径，以便红帽构建的 Keycloak 可以保护您的应用程序。

2.4. 限制已排队请求数

生产环境应该保护其自身不受过载情况的影响，以便它可以响应尽可能多的有效请求，并在情况再次返回到正常状态后继续常规操作。执行此操作的一种方法是在达到特定阈值后拒绝其他请求。

负载均衡器应该在所有级别上实现，包括环境中的负载均衡器。另外，红帽构建的 Keycloak 中有一个功能来限制无法立即处理的请求数，需要排队。默认情况下，没有设置限制。设置选项 **http-max-queued-requests**，将排队的请求数量限制为与您的环境匹配的给定阈值。任何超过这个限制的请求都会返回即时 **503 Server not Available** 响应。

2.5. PRODUCTION GRADE 数据库

红帽构建的 Keycloak 使用的数据库对于红帽构建的 Keycloak 的整体性能、可用性、可靠性和完整性至关重要。有关如何配置受支持的数据库的详情，请参阅配置数据库。

2.6. 支持集群中红帽构建的 KEYCLOAK

为确保用户在红帽构建的 Keycloak 实例时可以继续登录，典型的生产环境包含两个或更多红帽构建的 Keycloak 实例。

红帽构建的 Keycloak 在 JGroups 和 Infinispan 上运行，后者为集群场景提供可靠的高可用性堆栈。当部署到集群时，应保护嵌入式 Infinispan 服务器通信。您可以通过启用身份验证和加密或隔离用于集群通信的网络来保护此通信。

要了解更多有关使用多个节点的信息，请参阅 配置分布式缓存。

2.7. 使用 IPV4 或 IPV6 配置红帽构建的 KEYCLOAK 服务器

系统属性 **java.net.preferIPv4Stack** 和 **java.net.preferIPv6Addresses** 用于配置 JVM 以用于 IPv4 或 IPv6 地址。

默认情况下，红帽构建的 Keycloak 可通过 IPv4 和 IPv6 地址同时访问。要只使用 IPv4 地址运行，您需要指定属性 **java.net.preferIPv4Stack=true**。后者可确保任何主机名到 IP 地址转换始终返回 IPv4 地址变体。

这些系统属性可以通过 **JAVA_OPTS_APPEND** 环境变量轻松设置。例如，要将 IP 堆栈首选项改为 IPv4，请按如下所示设置环境变量：

```
export JAVA_OPTS_APPEND="-Djava.net.preferIPv4Stack=true"
```

第 3 章 在容器中运行红帽 KEYCLOAK 构建

本章论述了如何优化并运行红帽构建的 Keycloak 容器镜像，以提供运行容器的最佳体验。

警告

本章仅适用于构建您在 OpenShift 环境中运行的镜像。只有 OpenShift 环境支持此镜像。如果您在其他 Kubernetes 发行版中运行它，则不支持它。

3.1. 创建自定义和优化的容器镜像

Keycloak 容器镜像的默认红帽构建已准备好配置和优化。

为了更好地启动红帽构建的 Keycloak 容器，请在容器构建过程中运行 构建步骤 构建镜像。此步骤将在容器镜像的每个后续开始阶段节省时间。

3.1.1. 编写您优化的红帽构建的 Keycloak Dockerfile

以下 Dockerfile 会创建预配置的 Keycloak 镜像构建，启用健康和指标端点，启用令牌交换功能，并使用 PostgreSQL 数据库。

Dockerfile：

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:24 as builder

# Enable health and metrics support
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true

# Configure a database vendor
ENV KC_DB=postgres

WORKDIR /opt/keycloak
# for demonstration purposes only, please make sure to use proper certificates in production instead
RUN keytool -genkeypair -storepass password -storetype PKCS12 -keyalg RSA -keysize 2048 -dname "CN=server" -alias server -ext "SAN:c=DNS:localhost,IP:127.0.0.1" -keystore
```

```
conf/server.keystore
RUN /opt/keycloak/bin/kc.sh build

FROM registry.redhat.io/rhbk/keycloak-rhel9:24
COPY --from=builder /opt/keycloak/ /opt/keycloak/

# change these values to point to a running postgres instance
ENV KC_DB=postgres
ENV KC_DB_URL=<DBURL>
ENV KC_DB_USERNAME=<DBUSERNAME>
ENV KC_DB_PASSWORD=<DBPASSWORD>
ENV KC_HOSTNAME=localhost
ENTRYPOINT ["/opt/keycloak/bin/kc.sh"]
```

构建过程包含多个阶段：

- 运行 build 命令，以设置服务器构建选项来创建优化的镜像。
- 构建阶段生成的文件复制到新镜像中。
- 在最终镜像中，设置了主机名和数据库的额外配置选项，以便在运行容器时不需要再次设置它们。
- 在入口点中，kc.sh 启用对所有分发子命令的访问。

要安装自定义提供程序，您只需要定义一个步骤，将 JAR 文件包含到 /opt/keycloak/providers 目录中。此步骤必须放在 RUNs build 命令的行之前，如下所示：

```
# A example build step that downloads a JAR file from a URL and adds it to the providers directory
FROM registry.redhat.io/rhbk/keycloak-rhel9:24 as builder

...

# Add the provider JAR file to the providers directory
ADD --chown=keycloak:keycloak --chmod=644 <MY_PROVIDER_JAR_URL> /opt/keycloak/providers/myprovider.jar

...
```

```
# Context: RUN the build command
RUN /opt/keycloak/bin/kc.sh build
```

3.1.2. 安装额外的 RPM 软件包

如果您尝试在一个阶段的 FROM registry.redhat.io/rhbk/keycloak-rhel9 中安装新软件，您会注意到 microdnf、dnf，甚至没有安装 rpm。另外，很少的软件包可用，仅适用于 bash shell，并运行红帽构建的 Keycloak 本身。这是因为安全强化措施，这降低了红帽构建的 Keycloak 容器的攻击面。

首先，请考虑您的用例是否可以以不同的方式实施，因此请避免将新 RPM 安装到最终容器中：

- Dockerfile 中的 RUN curl 指令可以替换为 ADD，因为该指令原生支持远程 URL。
- 一些常见的 CLI 工具可以被 Linux 文件系统递归使用替代。例如，ip addr show tap0 变为 cat /sys/class/net/tap0/address
- 需要 RPM 的任务可以移到镜像构建的前阶段，以及复制的结果。

下面是一个示例。在以前的构建阶段运行 update-ca-trust，然后复制结果：

```
FROM registry.access.redhat.com/ubi9 AS ubi-micro-build
COPY mycertificate.crt /etc/pki/ca-trust/source/anchors/mycertificate.crt
RUN update-ca-trust

FROM registry.redhat.io/rhbk/keycloak-rhel9
COPY --from=ubi-micro-build /etc/pki /etc/pki
```

如果需要，可以安装新的 RPM，遵循由 ubi-micro 建立的双阶段模式：

```
FROM registry.access.redhat.com/ubi9 AS ubi-micro-build
RUN mkdir -p /mnt/rootfs
RUN dnf install --installroot /mnt/rootfs <package names go here> --releasever 9 --setopt
install_weak_deps=false --nodocs -y && \
    dnf --installroot /mnt/rootfs clean all && \
    rpm --root /mnt/rootfs -e --nodeps setup

FROM registry.redhat.io/rhbk/keycloak-rhel9
COPY --from=ubi-micro-build /mnt/rootfs /
```

这种方法使用 chroot /mnt/rootfs，以便只安装您指定的软件包及其依赖项，因此无需猜测在没有猜测的情况下，可以轻松地复制到第二个阶段。

警告

有些软件包具有大量依赖项。通过安装新的 RPM，您可能会意外增加容器的攻击面。仔细检查安装的软件包列表。

3.1.3. 构建容器镜像

要构建实际容器镜像，请从包含 Dockerfile 的目录运行以下命令：

```
podman build . -t mykeycloak
```

3.1.4. 启动优化的红帽 Keycloak 容器镜像构建

要启动镜像，请运行：

```
podman run --name mykeycloak -p 8443:8443 \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
      mykeycloak \
      start --optimized
```

红帽构建的 Keycloak 以生产模式启动，仅使用安全 HTTPS 通信，并可在 https://localhost:8443 中提供。

健康检查端点位于 https://localhost:8443/health、https://localhost:8443/health/ready 和 https://localhost:8443/health/live。

打开 https://localhost:8443/metrics 会导致页面，包含您的监控解决方案可以使用的操作指标。

3.2. 将容器公开给不同的端口

默认情况下，服务器分别使用端口 8080 和 8443 侦听 http 和 https 请求。

如果要使用不同的端口公开容器，则需要相应地设置 hostname-port ：

1. 使用默认端口以外的端口公开容器

```
podman run --name mykeycloak -p 3000:8443 \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
      mykeycloak \
      start --optimized --hostname-port=3000
```

通过设置 hostname-port 选项，您现在可以访问位于 https://localhost:3000 的服务器。

3.3. 试用红帽以开发模式构建 KEYCLOAK

从容器试用红帽从容器构建的 Keycloak 的最简单方法是使用 Development 模式。您可以使用 start-dev 命令：

```
podman run --name mykeycloak -p 8080:8080 \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
      registry.redhat.io/rhbk/keycloak-rhel9:24 \
      start-dev
```

调用此命令会以开发模式启动红帽 Keycloak 服务器的构建。

在生产环境中应该严格避免此模式，因为它具有不安全的默认值。有关在生产环境中运行红帽构建的 Keycloak 的更多信息，请参阅 为生产环境配置 Keycloak。

3.4. 运行标准红帽构建的 KEYCLOAK 容器

在与不可变基础架构等概念保持同步时，需要定期重新置备容器。在这些环境中，您需要快速启动的容器，因此您需要创建一个优化的镜像，如上一节中所述。但是，如果您的环境有不同的要求，您可以通过运行 start 命令运行标准红帽构建的 Keycloak 镜像。例如：

```
podman run --name mykeycloak -p 8080:8080 \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
      registry.redhat.io/rhbk/keycloak-rhel9:24 \
      start \
      --db=postgres --features=token-exchange \
      --db-url=<JDBC-URL> --db-username=<DB-USER> --db-password=<DB-PASSWORD> \
      --https-key-store-file=<file> --https-key-store-password=<password>
```

运行此命令会启动红帽构建的 Keycloak 服务器，它会首先检测并应用构建选项。在示例中，行 --db=postgres --features=token-exchange 将数据库供应商设置为 PostgreSQL，并启用令牌交换功能。

然后，红帽构建的 Keycloak 会启动并应用特定环境的配置。这个方法会显著增加启动时间，并创建一个可变的镜像，这不是最佳实践。

3.5. 在容器中运行时提供初始 ADMIN 凭证

红帽构建的 Keycloak 只允许从本地网络连接创建初始 admin 用户。在容器中运行时并非如此，因此您必须在运行镜像时提供以下环境变量：

```
# setting the admin username
-e KEYCLOAK_ADMIN=<admin-user-name>

# setting the initial password
-e KEYCLOAK_ADMIN_PASSWORD=change_me
```

3.6. 在启动时导入域

红帽构建的 Keycloak 容器有一个目录 /opt/keycloak/data/import。如果您通过卷挂载或其他方法将一个或多个导入文件放在那个目录中，并添加启动参数 --import-realm，红帽构建的 Keycloak 容器将在启动时导入这些数据！这可能只在 Dev 模式中有意义。

```
podman run --name keycloak_unoptimized -p 8080:8080 \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
      -v /path/to/realm/data:/opt/keycloak/data/import \
      registry.redhat.io/rhbk/keycloak-rhel9:24 \
      start-dev --import-realm
```

您可以自由加入有关管理 bootstrap 过程的改进的开放 GitHub 讨论。

3.7. 指定不同的内存设置

红帽构建的 Keycloak 容器，而不是为初始和最大堆大小指定硬编码值，而是使用相对值到容器的总内存。JVM 选项 -XX:MaxRAMPercentage=70、-XX:InitialRAMPercentage=50 来实现此行为。

-XX:MaxRAMPercentage 选项表示容器内存总量的 70% 的最大堆大小。-XX:InitialRAMPercentage 选项代表容器内存总量的 50% 的初始堆大小。这些值根据红帽构建的 Keycloak 内存管理的更深入分析

来选择。

由于堆大小根据容器内存总量动态计算，您应该始终为容器设置内存限值。在以前的版本中，最大堆大小设置为 512 MB，为了接近类似值，您应该将内存限值设置为至少 750 MB。对于较小的生产就绪部署，推荐的内存限值为 2 GB。

通过设置环境变量 JAVA_OPTS_KC_HEAP 来覆盖与堆相关的 JVM 选项。您可以在 kc.sh 或 kc.bat 脚本的源代码中找到 JAVA_OPTS_KC_HEAP 的默认值。

例如，您可以指定环境变量和内存限值，如下所示：

```
podman run --name mykeycloak -p 8080:8080 -m 1g \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
      -e JAVA_OPTS_KC_HEAP="-XX:MaxHeapFreeRatio=30 -XX:MaxRAMPercentage=65" \
      registry.redhat.io/rhbk/keycloak-rhel9:24 \
      start-dev
```

警告

如果没有设置内存限制，则内存消耗会快速增加，因为堆大小可增长总容器内存的 70%。在 JVM 分配内存后，它会与当前红帽构建的 Keycloak GC 设置一起返回到操作系统。

3.8. 相关选项

	value
db 数据库供应商。 CLI: **--db** Env: **KC_DB**	**dev-file**（默认）、**dev-mem**、**mariadb**、**mssql**、**mysql**、**oracle**、**postgres**
db-password 数据库用户的密码。 CLI: **--db-password** Env: **KC_DB_PASSWORD**	

	value
db-url 完整的数据库 JDBC URL。 如果没有提供，会根据所选数据库厂商设置默认 URL。例如，如果使用 **postgres**，默认的 JDBC URL 将是 **jdbc:postgresql://localhost/keycloak**。 CLI: **--db-url** Env: **KC_DB_URL**	
db-username 数据库用户的用户名。 CLI: **--db-username** Env: **KC_DB_USERNAME**	
功能 wagon 启用一组一个或多个功能。 CLI: **--features** Env: **KC_FEATURES**	**account-api[:v1]**, **account2[:v1]**, **account3[:v1]**, **admin-api[:v1]**, **admin-fine-grained-authz[:v1]**, **admin2[:v1]**, **authorization[:v1]**, **ciba[:v1]**, **client-policies[:v1]**, **client-secret-rotation[:v1]**, **client-types[:v1]**, **declarative-ui[:v1]**, **device-flow[:v1]**, **docker[:v1]**, **dpop[:v1]**, **dynamic-scopes[:v1]**, **fips[:v1]**, **hostname[:v1]**, **impersonation[:v1]**, **js-adapter[:v1]**, **kerberos[:v1]**, **linkedin-oauth[:v1]**, **login2[:v1]**, **Multi-site[:v1]**, **offline-session-preloading[:v1]**, **oid4vc-vci[:v1]**, **par[:v1]**, **preview**,**recovery-codes[:v1]**, **scripts[:v1]**, **step-up-authentication[:v1]**, **token-exchange[:v1]**, **transient-users[:v1]**, **update-email[:v1]**, **web-authn[:v1]**

	value
health-enabled ▮ 如果服务器应公开健康检查端点。 如果启用，则健康检查位于 **/health**、**/health/ready** 和 **/health/live** 端点中。 CLI: **--health-enabled** Env: **KC_HEALTH_ENABLED**	**true**,**false** （默认）
hostname Keycloak 服务器的主机名。 CLI: **--hostname** Env: **KC_HOSTNAME**	

	value
https-key-store-file 保存证书信息的密钥存储，而不是指定单独的文件。 CLI: **--https-key-store-file** Env: **KC_HTTPS_KEY_STORE_FILE**	
https-key-store-password 密钥存储文件的密码。 CLI: **--https-key-store-password** Env: **KC_HTTPS_KEY_STORE_PASSWORD**	**密码** （默认）
metrics-enabled 如果服务器应该公开指标。 如果启用，指标位于 **/metrics** 端点。 CLI: **--metrics-enabled** Env: **KC_METRICS_ENABLED**	**true**,**false** （默认）

第 4 章 配置 TLS

传输层安全性（短：TLS）对于通过安全通道交换数据至关重要。对于生产环境，您不应该通过 HTTP 公开红帽构建的 Keycloak 端点，因为敏感数据是红帽与其他应用程序构建 Keycloak 交换的核心。在本章中，您将了解如何将红帽构建的 Keycloak 配置为使用 HTTPS/TLS。

4.1. 在红帽构建的 KEYCLOAK 中配置 TLS

红帽构建的 Keycloak 可以配置为使用 PEM 格式或 Java Keystore 中的文件加载所需的证书基础架构。当同时配置了两个替代方案时，PEM 文件优先于 Java 密钥存储。

4.1.1. 以 PEM 格式提供证书

当您以 PEM 格式使用一组匹配的证书和私钥文件时，您可以通过运行以下命令来将 Keycloak 的红帽构建配置为使用它们：

```
bin/kc.[sh|bat] start --https-certificate-file=/path/to/certfile.pem --https-certificate-key-file=/path/to/keyfile.pem
```

Red Hat build of Keycloak 会在内存中从这些文件创建一个密钥存储，并在之后使用此密钥存储。

4.1.2. 提供 Java 密钥存储

如果没有显式配置密钥存储文件，但 http-enabled 被设置为 false，Red Hat build of Keycloak 会查找 conf/server.keystore 文件。

另外，您可以通过运行以下命令来使用现有的密钥存储：

```
bin/kc.[sh|bat] start --https-key-store-file=/path/to/existing-keystore-file
```

4.1.2.1. 设置密钥存储密码

您可以使用 https-key-store-password 选项为密钥存储设置安全密码：

```
bin/kc.[sh|bat] start --https-key-store-password=<value>
```

如果没有设置密码，则使用默认 密码。

4.2. 配置 TLS 协议

默认情况下，红帽构建的 Keycloak 不会启用已弃用的 TLS 协议。如果您的客户端只支持已弃用的协议，请考虑升级客户端。但是，作为临时工作，您可以通过运行以下命令来启用已弃用的协议：

```
bin/kc.[sh|bat] start --https-protocols=<protocol>[,<protocol>]
```

要也允许 TLSv1.2，请使用以下命令： kc.sh start --https-protocols=TLSv1.3,TLSv1.2。

4.3. 切换 HTTPS 端口

红帽构建的 Keycloak 在端口 8443 上侦听 HTTPS 流量。要更改此端口，请使用以下命令：

```
bin/kc.[sh|bat] start --https-port=<port>
```

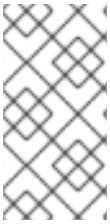
4.4. 使用信任存储

为了正确验证客户端证书并启用某些身份验证方法，如双向 TLS 或 mTLS，您可以使用服务器应信任的所有证书（和证书链）设置信任存储。依赖此信任存储的功能数量使用证书正确验证客户端，例如：

- mutual-TLS 客户端身份验证
- 最终用户 X.509 浏览器身份验证

您可以运行以下命令来配置此信任存储的位置：

```
bin/kc.[sh|bat] start --https-trust-store-file=/path/to/file
```



注意

此信任存储适用于验证红帽构建的 Keycloak 作为服务器的客户端。有关配置红帽构建的 Keycloak 作为一个通过 TLS 的客户端到外部服务的信任存储，请参阅配置可信证书。

4.4.1. 设置 truststore 密码

您可以使用 https-trust-store-password 选项为信任存储设置安全密码：

```
bin/kc.[sh|bat] start --https-trust-store-password=<value>
```

如果没有设置密码，则使用默认 密码。

4.5. 保护凭证

避免使用 CLI 以纯文本形式设置密码，或将其添加到 conf/keycloak.conf 文件中。反之，使用良好做法，比如使用 vault/ 挂载的 secret。如需了解更多详细信息，请参阅使用密码库 以及 为生产环境配置红帽 Keycloak。

4.6. 启用 MUTUAL TLS

默认禁用使用 mTLS 进行身份验证。当红帽构建的 Keycloak 是服务器时启用 mTLS 证书处理，需要验证来自红帽构建的 Keycloak 端点请求的证书，请将适当的证书放在红帽构建的 Keycloak 信任存储中，并使用以下命令启用 mTLS：

```
bin/kc.[sh|bat] start --https-client-auth=<none|request|required>
```

使用所需 值设置 红帽构建的 Keycloak 以始终询问证书，并在请求中没有提供证书时失败。通过将值设置为 请求，红帽构建的 Keycloak 也会在没有证书的情况下接受请求，且仅在证书存在时验证证书的正确性。

请注意，这是红帽构建的 Keycloak 作为服务器的 mTLS 用例的基本证书配置。当红帽构建的 Keycloak 作为客户端时，例如，当红帽构建的 Keycloak 尝试从代理身份提供程序的令牌端点获取令牌时，您需要设置 HttpClient 来为出站请求在密钥存储中提供正确的证书。要在这些场景中配置 mTLS，请参阅配置传出的 HTTP 请求。

4.7. 相关选项

	value
http-enabled 启用 HTTP 侦听器。 CLI: **--http-enabled** Env: **KC_HTTP_ENABLED**	**true**,**false** （默认）
https-certificate-file PEM 格式的服务器证书或证书链的文件路径。 CLI: **--https-certificate-file** Env: **KC_HTTPS_CERTIFICATE_FILE**	
https-certificate-key-file PEM 格式到私钥的文件路径。 CLI: **--https-certificate-key-file** Env: **KC_HTTPS_CERTIFICATE_KEY_FILE**	
https-cipher-suites 要使用的密码套件。 如果未指定，则会选择合理的默认值。 CLI: **--https-cipher-suites** Env: **KC_HTTPS_CIPHER_SUITES**	
https-client-auth 将服务器配置为 require/request 客户端身份验证。 CLI: **--https-client-auth** Env: **KC_HTTPS_CLIENT_AUTH**	**none** （默认）、请求、**必需**
https-key-store-file 保存证书信息的密钥存储，而不是指定单独的文件。 CLI: **--https-key-store-file** Env: **KC_HTTPS_KEY_STORE_FILE**	
https-key-store-password 密钥存储文件的密码。 CLI: **--https-key-store-password** Env: **KC_HTTPS_KEY_STORE_PASSWORD**	**密码** （默认）

	value
https-key-store-type 密钥存储文件的类型。 如果没有给定，会根据文件名自动检测类型。如果将 **fips-mode** 设置为 **strict** 且没有设置值，则默认为 **BCFKS**。 CLI: **--https-key-store-type** Env: **KC_HTTPS_KEY_STORE_TYPE**	
https-port 使用的 HTTPS 端口。 CLI: **--https-port** Env: **KC_HTTPS_PORT**	**8443**（默认）
https-protocols 要显式启用的协议列表。 CLI: **--https-protocols** Env: **KC_HTTPS_PROTOCOLS**	**[TLSv1.3,TLSv1.2]**（默认）
https-trust-store-file 包含要信任的证书信息的信任存储。 CLI: **--https-trust-store-file** Env: **KC_HTTPS_TRUST_STORE_FILE** **已弃用。**使用 System Truststore 替代，请参阅文档了解详情。	
https-trust-store-password 信任存储文件的密码。 CLI: **--https-trust-store-password** Env: **KC_HTTPS_TRUST_STORE_PASSWORD** **已弃用。**使用 System Truststore 替代，请参阅文档了解详情。	
https-trust-store-type 信任存储文件的类型。 如果没有给定，会根据文件名自动检测类型。如果将 **fips-mode** 设置为 **strict** 且没有设置值，则默认为 **BCFKS**。 CLI: **--https-trust-store-type** Env: **KC_HTTPS_TRUST_STORE_TYPE** **已弃用。**使用 System Truststore 替代，请参阅文档了解详情。	

第 5 章 配置主机名

5.1. 服务器端点

红帽构建的 Keycloak 会公开不同的端点，以便与应用程序进行通信，并允许访问管理控制台。这些端点可以归类为三个主要组：

- frontend
- 后端
- 管理控制台

每个组的基本 URL 对签发和验证令牌具有重要影响，关于如何为需要用户重定向到红帽构建的 Keycloak 的操作创建链接（例如，通过电子邮件链接重置密码）以及更重要的是，应用程序在从 realms/{realm-name}/.well-known/openid-configuration 中获取 OpenID Connect Discovery Document 操作时如何发现这些端点。

5.1.1. frontend

frontend 端点是指通过公共域访问，通常与通过前端通道进行的身份验证/授权流相关的。例如，当 SPA 想要验证其用户时，它会将他们重定向到 authorization_endpoint，以便用户可以通过前端通道使用浏览器进行身份验证。

默认情况下，如果没有设置主机名设置，这些端点的基本 URL 基于传入请求，因此 HTTP 方案、主机、端口和路径与请求相同。默认行为还直接影响到服务器如何发出令牌，因为签发者也基于设置为 frontend 端点的 URL。如果没有设置主机名设置，则令牌签发者也会基于传入请求，并在客户端使用不同的 URL 请求令牌时缺少一致性。

当部署到生产环境时，对于 frontend 端点和令牌签发者，您通常需要一个一致的 URL，无论请求是如何构建的。为了实现此一致性，您可以设置主机名或 hostname -url 选项。

大多数时候，应该足以设置 hostname 选项，以便仅更改前端 URL 的主机 ：

```
bin/kc.[sh|bat] start --hostname=<host>
```

当使用 hostname 选项时，服务器将自动解析 HTTP 方案、端口和路径，以便：

- 使用 HTTPS 方案，除非您设置了 hostname-strict- https =false
- 如果设置了 proxy-headers 选项，代理将使用默认端口（例如： 80 和 443）。如果代理使用不同的端口，则需要通过 hostname-url 配置选项指定它

但是，如果您不仅要设置主机，同时希望设置方案、端口和路径，您可以设置 hostname-url 选项：

```
bin/kc.[sh|bat] start --hostname-url=<scheme>://<host>:<port>/<path>
```

这个选项为您提供了更大的灵活性，因为您可以通过单个选项设置 URL 的不同部分。请注意，主机名和 hostname-url 是互斥的。

注意

通过 主机名和 proxy-headers 配置选项，您只能影响静态资源 URL、重定向 URI、OIDC 已知的端点等。若要更改服务器实际侦听的端口，您需要使用 http/tls 配置选项（如 http-host、https-port 等等）。如需了解更多详细信息，请参阅配置 TLS 和所有 配置。

5.1.2. 后端

后端端点是那些通过公共域或通过专用网络访问的端点。它们用于服务器和客户端之间的直接通信，而无需任何中间，但是普通 HTTP 请求。例如，在验证 SPA 后，希望通过向 token_endpoint 发送令牌请求来交换具有一组令牌的服务器发送的 代码。

默认情况下，后端端点的 URL 也基于传入的请求。要覆盖此行为，请输入以下命令设置 hostname-strict-backchannel 配置选项：

```
bin/kc.[sh|bat] start --hostname=<value> --hostname-strict-backchannel=true
```

通过设置 hostname-strict-backchannel 选项，后端端点的 URL 将与 frontend 端点完全相同。

当所有连接到红帽构建的 Keycloak 的应用程序都通过公共 URL 进行通信时，请将 hostname-strict-backchannel 设置为 true。否则，将此参数保留为 false，以允许通过专用网络进行客户端-服务器通

信。

5.1.3. 管理控制台

服务器使用特定 URL 公开管理控制台和静态资源。

默认情况下，管理控制台的 URL 也基于传入请求。但是，如果要使用特定 URL 限制对管理控制台的访问，您可以设置特定的主机或基本 URL。与设置 frontend URL 类似，您可以使用 hostname-admin 和 hostname-admin-url 选项来实现这一目的。请注意，如果启用了 HTTPS (http-enabled 配置选项被设置为 false，这是生产模式的默认设置)，红帽构建的 Keycloak 服务器会自动假设您要使用 HTTPS URL。然后，管理控制台会尝试通过 HTTPS 联系红帽构建的 Keycloak，HTTPS URL 也用于其配置的重定向/Web 原始 URL。不建议在生产环境中使用，但您可以使用 HTTP URL 作为 hostname-admin-url 来覆盖此行为。

大多数时候，设置 hostname-admin 选项应该只更改管理控制台 URL 的主机 ：

```
bin/kc.[sh|bat] start --hostname-admin=<host>
```

但是，如果您不仅要设置主机，同时希望设置方案、端口和路径，您可以设置 hostname-admin-url 选项：

```
bin/kc.[sh|bat] start --hostname-admin-url=<scheme>://<host>:<port>/<path>
```

请注意，hostname-admin 和 hostname-admin-url 是互斥的。

为减少受攻击面，红帽构建的 Keycloak 管理端点不应公开访问。因此，您可以使用反向代理来保护它们。有关使用反向代理公开的路径的更多信息，请参阅使用反向代理。

5.2. 使用示例

以下是更多示例场景以及设置主机名的对应命令。

请注意，start 命令需要设置 TLS。例如，不会显示对应的选项。如需了解更多详细信息，请参阅配置 TLS。

5.2.1. 在 TLS 终止代理后面公开服务器

在这个示例中，服务器在 TLS 终止代理后面运行，并从 https://mykeycloak 公开。

配置：

```
bin/kc.[sh|bat] start --hostname=mykeycloak --http-enabled=true --proxy-
headers=forwarded|xforwarded
```

5.2.2. 在没有代理的情况下公开服务器

在本例中，服务器在没有代理的情况下运行，并使用 HTTPS 使用 URL 公开。

红帽构建的 Keycloak 配置：

```
bin/kc.[sh|bat] start --hostname-url=https://mykeycloak
```

出于安全性和可用性的原因，强烈建议在服务器前面使用 TLS 终止代理。如需了解更多详细信息，请参阅使用反向代理。

5.2.3. 强制后端端点使用服务器公开的相同 URL

在本例中，后端端点使用服务器使用的同一 URL 来公开，以便无论请求的来源如何，客户端始终获取相同的 URL。

红帽构建的 Keycloak 配置：

```
bin/kc.[sh|bat] start --hostname=mykeycloak --hostname-strict-backchannel=true
```

5.2.4. 使用默认端口以外的端口公开服务器

在本例中，服务器可以通过默认端口以外的端口访问。

红帽构建的 Keycloak 配置：

```
bin/kc.[sh|bat] start --hostname-url=https://mykeycloak:8989
```

5.2.5. 使用不同端口在 TLS 重新加密代理后公开红帽构建的 Keycloak

在本例中，服务器在代理后面运行，服务器和代理都使用自己的证书，因此红帽构建的 Keycloak 和代理之间的通信是加密的。反向代理使用 Forwarded 标头，且不会设置 X-Forwardedjpeg 标头。我们需要记住代理配置选项（以及主机名配置选项）不会更改服务器实际侦听的端口（它只更改 JavaScript 和 CSS 链接、OIDC 已知端点、重定向 URI 等）。因此，我们需要使用 HTTP 配置选项将红帽构建的 Keycloak 服务器更改为内部侦听不同的端口，如 8543。代理将侦听端口 8443 （通过浏览器访问控制台时可见的端口）。示例主机名 my-keycloak.org 将用于服务器，类似管理控制台将可通过 admin.my-keycloak.org 子域访问。

红帽构建的 Keycloak 配置：

```
bin/kc.[sh|bat] start --proxy-headers=forwarded --https-port=8543 --hostname-url=https://my-keycloak.org:8443 --hostname-admin-url=https://admin.my-keycloak.org:8443
```

警告

使用 proxy-headers 选项分别依赖 Forwarded 和 X-Forwarded mysql 标头，该标头必须被反向代理设置和覆盖。错误配置可能会使红帽构建的 Keycloak 暴露给安全问题。如需了解更多详细信息，请参阅使用反向代理。

5.3. 故障排除

要排除主机名配置的问题，您可以使用专用的 debug 工具，它可以启用：

红帽构建的 Keycloak 配置：

```
bin/kc.[sh|bat] start --hostname=mykeycloak --hostname-debug=true
```

然后，在红帽构建的 Keycloak 正确启动后，打开浏览器并进入：

http://mykeycloak:8080/realms/<your-realm>/hostname-debug

5.4. 相关选项

表 5.1. 默认情况下，此端点被禁用(--hostname-debug=false)

	value
hostname Keycloak 服务器的主机名。 CLI: **--hostname** Env: **KC_HOSTNAME**	
hostname-admin 用于访问管理控制台的主机名。 如果您使用值设置为 hostname 选项以外的主机名公开管理控制台，请使用这个选项。 CLI: **--hostname-admin** Env: **KC_HOSTNAME_ADMIN**	
hostname-admin-url 设置用于访问管理控制台的基本 URL，包括方案、主机、端口和路径 CLI: **--hostname-admin-url** Env: **KC_HOSTNAME_ADMIN_URL**	

	value
hostname-debug 切换可通过 /realms/master/hostname-debug 访问的主机名调试页面 CLI: **--hostname-debug** Env: **KC_HOSTNAME_DEBUG**	**true**,**false** （默认）
hostname-path 如果代理为 Keycloak 使用不同的 context-path，则应设置此项。 CLI: **--hostname-path** Env: **KC_HOSTNAME_PATH**	
hostname-port 代理在公开主机名时使用的端口。 如果代理使用默认 HTTP 和 HTTPS 端口以外的端口，则设置这个选项。 CLI: **--hostname-port** Env: **KC_HOSTNAME_PORT**	**-1** （默认）
hostname-strict 禁用从请求标头动态解析主机名。 在生产环境中应始终设置为 true，除非代理验证 Host 标头。 CLI: **--hostname-strict** Env: **KC_HOSTNAME_STRICT**	**true** （默认）、**false**
hostname-strict-backchannel 默认情况下，后端通道 URL 从请求标头动态解析，以允许内部和外部应用程序。 如果所有应用都使用公共 URL，则应启用此选项。 CLI: **--hostname-strict-backchannel** Env: **KC_HOSTNAME_STRICT_BACKCHANNEL**	**true**,**false** （默认）
hostname-url 设置 frontend URL 的基本 URL，包括方案、主机、端口和路径。 CLI: **--hostname-url** Env: **KC_HOSTNAME_URL**	

	value
proxy 如果服务器位于反向代理后面，则代理地址转发模式。 CLI: **--proxy** Env: **KC_PROXY** **已弃用。**使用： **proxy-headers**。	**none**（默认）、边缘、**重新加密**、**透传**

第 6 章 使用反向代理

分布式环境通常需要使用反向代理。红帽构建的 Keycloak 提供多个选项来安全地与此类环境集成。

6.1. 配置反向代理标头

红帽构建的 Keycloak 将根据 proxy-headers 选项解析反向代理标头，该选项接受几个值：

- 默认情况下，如果没有指定选项，则不会解析反向代理标头。
- 转发 启用根据 RFC7239 解析 Forwarded 标头。
- X forwarded 启用解析非标准 X-Forwarded114 标头，如 X-Forwarded -For、X-Forwarded-Proto、X-Forwarded-Host、X-Forwarded-Port 和 X-Forwarded-Port。

例如：

```
bin/kc.[sh|bat] start --proxy-headers forwarded
```

警告

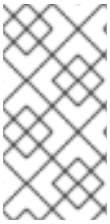
如果选择了 转发 或 xforwarded，请确保您的反向代理正确设置并分别覆盖 Forwarded 或 X-Forwarded noted 标头。要设置这些标头，请参阅您的反向代理文档。错误配置会将红帽构建的 Keycloak 暴露给安全漏洞。

采取额外的措施，以确保通过 Forwarded 或 X-Forwarded-For 标头的反向代理正确设置客户端地址。如果这个标头配置不正确，则 rogue 客户端可以设置此标头，并欺骗红帽构建的 Keycloak 认为客户端从与实际地址不同的 IP 地址连接。如果您执行任何拒绝或允许 IP 地址列表，则这种预防措施更为重要。

注意

使用 xforwarded 设置时，X-Forwarded-Port 优先于 X-Forwarded-Host 中包含的任何端口。

6.2. 代理模式



注意

对设置代理模式的支持已弃用，并将在以后的红帽构建的 Keycloak 发行版本中删除。考虑配置接受的反向代理标头，如上面的章节中所述。有关迁移说明，请参阅升级指南。

对于红帽构建的 Keycloak，您选择的代理模式取决于您的环境中的 TLS 终止。可用的代理模式如下：

edge

启用代理与红帽构建的 Keycloak 之间的 HTTP 通信。这个模式适用于具有高度安全内部网络的部署，其中反向代理在与使用 HTTP 的 Keycloak 进行通信时，将安全连接（通过 TLS 通过 TLS）与客户端保持安全连接（通过 TLS）。

reencrypt

需要通过 HTTPS 在代理和红帽构建的 Keycloak 间的通信。此模式适用于在反向代理和红帽 Keycloak 构建间内部通信的部署也应受到保护。在反向代理以及 Keycloak 的红帽构建中使用不同的密钥和证书。

passthrough

代理将 HTTPS 连接转发到红帽构建的 Keycloak，而不终止 TLS。服务器和客户端之间的安全连接基于红帽构建的 Keycloak 服务器使用的密钥和证书。

在 边缘 或重新加密 代理模式中，红帽构建的 Keycloak 将解析以下标头，并期望反向代理来设置它们：

- 根据 RFC7239进行转发
- 非标准 X-Forwarded114，如 X-Forwarded-For、X-Forwarded-Proto、X-Forwarded-Host 和 X-Forwarded-Port

6.2.1. 在 Red Hat build of Keycloak 中配置代理模式

要选择代理模式，请输入以下命令：

```
bin/kc.[sh|bat] start --proxy <mode>
```

6.3. 反向代理上的不同 CONTEXT-PATH

红帽构建的 Keycloak 假设它通过与红帽构建的 Keycloak 配置相同的上下文路径下公开。默认情况下，Red Hat build of Keycloak 通过 root (/)公开，这意味着它还需要通过 / 上的反向代理公开。在这些情况下，您可以使用 hostname-path 或 hostname-url，例如，如果在 /auth 上通过反向代理公开 Keycloak 的红帽构建，则可以使用 --hostname-path=/auth。

另外，您还可以使用 http-relative-path 选项更改红帽构建的 Keycloak 本身的上下文路径，以匹配反向代理的上下文路径，这将更改红帽构建的 Keycloak 本身的上下文路径，以匹配反向代理所使用的上下文路径。

6.4. 信任代理来设置主机名

默认情况下，红帽构建的 Keycloak 需要知道它将被调用的主机名。如果您的反向代理被配置为检查正确的主机名，您可以将红帽构建的 Keycloak 设置为接受任何主机名。

```
bin/kc.[sh|bat] start --proxy-headers=forwarded|xforwarded --hostname-strict=false
```

6.5. 启用粘性会话

典型的集群部署由负载均衡器（反向代理）和 2 个或更多红帽在专用网络上构建 Keycloak 服务器。出于性能的需要，如果负载均衡器将与特定浏览器会话相关的所有请求转发到同一红帽构建的 Keycloak 后端节点，这可能很有用。

其原因在于，红帽构建的 Keycloak 使用 Infinispan 分布式缓存，覆盖之下，以保存与当前身份验证会话和用户会话相关的数据。Infinispan 分布式缓存默认配置为两个所有者。这意味着，特定的会话主要存储在两个集群节点上，而其他节点需要远程查找会话（如果想要访问它）。

例如，如果 ID 为 123 的验证会话保存在 node1 上的 Infinispan 缓存中，则 node2 需要通过网络向 node1 发送请求，以返回特定的会话实体。

如果特定的会话实体始终在本地可用，这可以通过粘性会话的帮助来完成。集群环境中的工作流带有公

共前端负载均衡器，以及两个红帽构建的 Keycloak 节点，如下所示：

- 用户发送初始请求以查看红帽 Keycloak 登录屏幕的构建
- 此请求由 frontend 负载均衡器提供，该负载均衡器将其转发到一些随机节点（例如 node1）。严格说，节点不需要随机，但可以根据某些其他条件（客户端 IP 地址等）进行选择。它都取决于底层负载均衡器的实施和配置（反向代理）。
- 红帽构建的 Keycloak 使用随机 ID （如 123）创建身份验证会话，并将其保存到 Infinispan 缓存。
- Infinispan 分布式缓存根据会话 ID 的哈希值分配会话的主要所有者。有关此问题的更多详细信息，请参阅 Infinispan 文档。假设 Infinispan 分配 node2 是此会话的所有者。
- Red Hat build of Keycloak 创建 cookie AUTH_SESSION_ID，格式为 <session-id>.<owner-node-id>。在我们的示例中，它将为 123.node2。
- 使用红帽构建的 Keycloak 登录屏幕和浏览器中的 AUTH_SESSION_ID cookie 返回用户的响应

从此时，如果负载均衡器将所有下一个请求转发到 node2，因为这是 ID 为 123 的验证会话的所有者，因此 Infinispan 可以在本地查找此会话。身份验证完成后，身份验证会话将转换为用户会话，该会话也会保存在 node2 中，因为它具有相同的 ID 123。

集群设置的粘性会话不是强制的，但由于上述原因，最好是性能。您需要通过 AUTH_SESSION_ID cookie 将 loadbalancer 配置为粘性。具体操作取决于您的负载均衡器。

如果您的代理支持会话关联，而不处理来自后端节点的 Cookie，您应该将 spi-sticky-session-encoder-infinispan-should-attach-route 选项设为 false，以避免将节点附加到 Cookie，并只依赖于反向代理功能。

```
bin/kc.[sh|bat] start --spi-sticky-session-encoder-infinispan-should-attach-route=false
```

默认情况下，spi-sticky-session-encoder-infinispan-should-attach-route 选项值为 true，以便节点名称附加到 Cookie，以指示后续请求应发送到的反向代理。

6.5.1. 公开管理控制台

默认情况下，管理控制台 URL 仅基于请求来解析正确的方案、主机名和端口。例如，如果您使用 边缘代理模式，且代理配置不正确，则来自 TLS 终止代理的后端请求将使用普通 HTTP，并可能导致管理控制台被访问，因为将使用 http 方案创建 URL，代理不支持普通 HTTP。

要正确公开管理控制台，您应该确保代理在这里设置 X-Forwarded 114 标头，以便使用代理公开的方案、主机名和端口来创建 URL。

6.5.2. 公开的路径建议

当使用反向代理时，红帽构建的 Keycloak 只需要公开某些路径。下表显示了要公开的推荐路径。

红帽构建的 Keycloak 路径	反向代理路径	expose	原因
/	-	否	在公开所有路径时，管理路径会不必要地公开。
/admin/	-	否	公开的管理路径会导致不必要的攻击向量。
/js/	-	是（请参阅以下备注）	访问"内部"客户端所需的 keycloak.js，例如帐户控制台
/welcome/	-	否	不需要在初始安装后公开欢迎页面。
/realms/	/realms/	是	需要这个路径才能正常工作，例如对于 OIDC 端点。
/resources/	/resources/	是	需要此路径才能正确提供资产。它可以从 CDN 而不是红帽构建的 Keycloak 路径提供。
/robots.txt	/robots.txt	是	搜索引擎规则
/metrics	-	否	公开的指标会导致不必要的攻击向量。
/health	-	否	公开健康检查会导致不必要的攻击向量。

注意

由于像帐户控制台等内部客户端需要 js 路径，因此最好使用 JavaScript 软件包管理器中的 keycloak.js，如 npm 或 yarn 用于外部客户端。

我们假设您在反向代理/网关公共 API 的 root 路径 / 上运行红帽 Keycloak。如果没有，请使用您所需路径作为前缀。

6.5.3. 启用客户端证书查找

当代理配置为 TLS 终止代理时，客户端证书信息可以通过特定的 HTTP 请求标头转发到服务器，然后用于验证客户端。您可以根据您使用的代理，配置服务器如何检索客户端证书信息。

服务器支持一些最常见的 TLS 终止代理，例如：

Proxy	供应商
Apache HTTP 服务器	Apache
HAProxy	hapoxy
NGINX	nginx

要配置如何从您需要的请求检索客户端证书：

启用对应的代理供应商

```
bin/kc.[sh|bat] build --spi-x509cert-lookup-provider=<provider>
```

配置 HTTP 标头

```
bin/kc.[sh|bat] start --spi-x509cert-lookup-<provider>-ssl-client-cert=SSL_CLIENT_CERT --spi-x509cert-lookup-<provider>-ssl-cert-chain-prefix=CERT_CHAIN --spi-x509cert-lookup-<provider>-certificate-chain-length=10
```

在配置 HTTP 标头时，您需要确保您使用的值对应于代理使用客户端证书信息转发的标头的名称。

用于配置供应商的可用选项有：

选项	描述
ssl-client-cert	保存客户端证书的标头名称
ssl-cert-chain-prefix	标头在链中包含额外证书的前缀，用于相应地检索单个证书到链的长度。例如，值 **CERT_CHAIN** 将告知服务器从标头 **CERT_CHAIN_0** 加载到 **CERT_CHAIN_9**（如果 **certificate-chain-length** 设置为 **10**）。
certificate-chain-length	证书链的最大长度。
trust-proxy-verification	启用信任 NGINX 代理证书验证，而不是将证书转发到红帽构建的 Keycloak 并在红帽构建的 Keycloak 中进行验证。

6.5.3.1. 配置 NGINX 供应商

NGINX SSL/TLS 模块不会公开客户端证书链。红帽构建的 Keycloak 的 NGINX 证书查找供应商使用红帽构建的 Keycloak 信任存储重新构建它。

如果您使用此供应商，请参阅配置可信证书 以了解如何配置红帽构建的 Keycloak Truststore。

6.6. 相关选项

	value
hostname-path 如果代理为 Keycloak 使用不同的 context-path，则应设置此项。 CLI: **--hostname-path** Env: **KC_HOSTNAME_PATH**	
hostname-url 设置 frontend URL 的基本 URL，包括方案、主机、端口和路径。 CLI: **--hostname-url** Env: **KC_HOSTNAME_URL**	
http-relative-path 为服务资源设置相对于 / 的路径。 该路径必须以 / 开头。 CLI: **--http-relative-path** Env: **KC_HTTP_RELATIVE_PATH**	/ （默认）
proxy 如果服务器位于反向代理后面，则代理地址转发模式。 CLI: **--proxy** Env: **KC_PROXY** **已弃用。**使用： **proxy-headers**。	**none**（默认）、边缘、**重新加密**、**透传**
proxy-headers 服务器应接受的代理标头。 错误配置可能会使服务器暴露给安全漏洞。优先于已弃用的代理选项。 CLI: **--proxy-headers** Env: **KC_PROXY_HEADERS**	转发，**xforwarded**

第 7 章 配置数据库

本章介绍了如何配置红帽构建的 Keycloak 服务器，将数据存储在关系数据库中。

7.1. 支持的数据库

服务器具有对不同数据库的内置支持。您可以通过查看 db 配置选项的预期值来查询可用的数据库。下表列出了支持的数据库及其测试的版本。

数据库	选项值	测试的版本
MariaDB 服务器	**mariadb**	10.11
Microsoft SQL Server	**mssql**	2022
MySQL	**mysql**	8.0
Oracle 数据库	**oracle**	19.3
PostgreSQL	**postgres**	16
Amazon Aurora PostgreSQL	**postgres**	16.1

默认情况下，服务器使用 dev-file 数据库。这是服务器用来持久保留数据的默认数据库，仅存在于开发用例中。dev-file 数据库不适用于生产环境用例，必须在部署到生产之前替换。

7.2. 安装数据库驱动程序

数据库驱动程序作为红帽构建的 Keycloak 的一部分提供，但 Oracle Database 和 Micrsoft SQL Server 驱动程序需要单独安装。

如果要连接到其中一个数据库，请安装必要的驱动程序，如果您要连接到已经包含数据库驱动程序的不同数据库，请跳过此部分。

7.2.1. 安装 Oracle 数据库驱动程序

为红帽构建的 Keycloak 安装 Oracle Database 驱动程序：

1. 从以下源之一下载 ojdbc11 和 orai18n JAR 文件：

   a. 来自 Oracle 驱动程序下载页的 zip JDBC 驱动程序和Companion Jars 版本 23.3.0.23.09。

   b. 通过 ojdbc11 和 orai18n 的 Maven Central。

   c. 数据库厂商推荐的安装介质用于使用的特定数据库。

2. 运行解压缩的分发时：红帽构建的 Keycloak 供应商 文件夹中的 ojdbc11 和 orai18n JAR 文件

3. 运行容器： 构建自定义红帽构建的 Keycloak 镜像，并在 provider 文件夹中添加 JAR。为 Operator 构建自定义镜像时，这些镜像需要使用红帽构建的 Keycloak 集的所有构建时间选项优化镜像。

   用于构建可用于红帽构建的 Keycloak Operator 的最小 Dockerfile，并包括从 Maven Central 下载的 Oracle Database JDBC 驱动程序，如下所示：

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:24
ADD --chown=keycloak:keycloak --chmod=644
https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc11/23.3.0.23.09/ojdbc1
1-23.3.0.23.09.jar /opt/keycloak/providers/ojdbc11.jar
ADD --chown=keycloak:keycloak --chmod=644
https://repo1.maven.org/maven2/com/oracle/database/nls/orai18n/23.3.0.23.09/orai18n-
23.3.0.23.09.jar /opt/keycloak/providers/orai18n.jar
# Setting the build parameter for the database:
ENV KC_DB=oracle
# Add all other build parameters needed, for example enable health and metrics:
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true
# To be able to use the image with the Red Hat build of Keycloak Operator, it needs to
be optimized, which requires Red Hat build of Keycloak's build step:
RUN /opt/keycloak/bin/kc.sh build
```

   如需了解如何构建优化镜像的详细信息，请参阅 容器章节中的 Running Red Hat build of Keycloak。

然后，按照下一节中所述继续配置数据库。

7.2.2. 安装 Microsoft SQL Server 驱动程序

为红帽构建的 Keycloak 安装 Microsoft SQL Server 驱动程序：

1. 从以下源之一下载 mssql-jdbc JAR 文件：
   a. 从 Microsoft JDBC Driver for SQL Server 页面 下载版本。
   b. 通过 mssql-jdbc 的 Maven Central。
   c. 数据库厂商推荐的安装介质用于使用的特定数据库。
2. 运行解压缩的分发时：红帽构建的 Keycloak 供应商 文件夹中的 mssql-jdbc
3. 运行容器： 构建自定义红帽构建的 Keycloak 镜像，并在 provider 文件夹中添加 JAR。当为红帽构建的 Keycloak Operator 构建自定义镜像时，需要使用红帽构建的 Keycloak 集的所有构建时选项优化镜像。

   构建可用于红帽构建的 Keycloak Operator 的最小 Dockerfile，并包括从 Maven Central 下载的 Microsoft SQL Server JDBC 驱动程序，如下所示：

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:24
ADD --chown=keycloak:keycloak --chmod=644
https://repo1.maven.org/maven2/com/microsoft/sqlserver/mssql-
jdbc/12.4.2.jre11/mssql-jdbc-12.4.2.jre11.jar /opt/keycloak/providers/mssql-jdbc.jar
# Setting the build parameter for the database:
ENV KC_DB=mssql
# Add all other build parameters needed, for example enable health and metrics:
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true
# To be able to use the image with the Red Hat build of Keycloak Operator, it needs to
be optimized, which requires Red Hat build of Keycloak's build step:
RUN /opt/keycloak/bin/kc.sh build
```

   如需了解如何构建优化镜像的详细信息，请参阅 容器章节中的 Running Red Hat build of Keycloak。

然后，按照下一节中所述继续配置数据库。

7.3. 配置数据库

对于每个支持的数据库，服务器会提供一些建议的默认值来简化数据库配置。您可以通过提供一些关键设置（如数据库主机和凭证）来完成配置。

1. 启动服务器并设置基本选项来配置数据库

```
bin/kc.[sh|bat] start --db postgres --db-url-host mypostgres --db-username myuser --db-password change_me
```

此命令包含连接到数据库所需的最小设置。

默认模式是 keycloak，但您可以使用 db-schema 配置选项更改它。

警告

如果要使用特定的 DB (H2 除外)，请不要将 --optimized 标志用于 start 命令。在启动服务器实例之前，执行构建阶段。您可以通过在没有 --optimized 标志的情况下启动实例，或者在优化启动前执行 build 命令来实现它。如需更多信息，请参阅配置红帽构建的 Keycloak。

7.4. 覆盖默认连接设置

服务器使用 JDBC 作为底层技术与数据库通信。如果默认连接设置不足，您可以使用 db-url 配置选项指定 JDBC URL。

以下是 PostgreSQL 数据库的示例命令：

```
bin/kc.[sh|bat] start --db postgres --db-url jdbc:postgresql://mypostgres/mydatabase
```

请注意，在调用包含特殊 shell 字符的命令时，您需要转义字符 ；使用 CLI，因此您可能需要在配置文件中设置它。

7.5. 覆盖默认 JDBC 驱动程序

服务器会相应地使用默认 JDBC 驱动程序到您选择的数据库。

要设置不同的驱动程序，您可以使用 JDBC 驱动程序的完全限定类名称设置 db-driver ：

```
bin/kc.[sh|bat] start --db postgres --db-driver=my.Driver
```

无论您设置的驱动是什么，默认驱动程序始终在运行时可用。

仅当您真正需要时设置此属性。例如，在为特定云数据库服务利用 JDBC 驱动程序 Wrapper 的功能时。

7.6. 为数据库配置 UNICODE 支持

Unicode 支持所有字段取决于数据库是否允许 VARCHAR 和 CHAR 字段使用 Unicode 字符集。

- 如果可以设置这些字段，则 Unicode 可能可以正常工作，通常以字段长度的费用为代价。
- 如果数据库仅支持 NVARCHAR 和 NCHAR 字段中的 Unicode，则所有文本字段的 Unicode 支持不太可能正常工作，因为服务器架构使用 VARCHAR 和 CHAR 字段。

数据库架构仅对以下特殊字段提供对 Unicode 字符串的支持：

- 域 ：显示名称、HTML 显示名称、本地化文本（键和值）
- 联邦 供应商：显示名称
- 用户 ：用户名、指定名称、姓氏、属性名称和值

- 组 ：名称、属性名称和值
- roles: name
- 对象的描述

否则，字符仅限于数据库编码中包含的字符，通常为 8 位。但是，对于某些数据库系统，您可以启用 Unicode 字符的 UTF-8 编码，并使用所有文本字段中设置的完整 Unicode 字符。对于给定数据库，这个选择可能会导致最大字符串长度比 8 位编码支持的最大字符串长度短。

7.6.1. 为 Oracle 数据库配置 Unicode 支持

如果在 VARCHAR 和 CHAR 字段中使用 Unicode 支持创建数据库，则 Oracle 数据库中支持 Unicode 字符。例如，您可以将 AL32UTF8 配置为数据库字符集。在这种情况下，JDBC 驱动程序不需要特殊设置。

如果没有使用 Unicode 支持创建数据库，则需要配置 JDBC 驱动程序来支持特殊字段中的 Unicode 字符。您可以配置两个属性。请注意，您可以将这些属性配置为系统属性或连接属性。

1. 将 oracle.jdbc.defaultNChar 设置为 true。
2. （可选）将 oracle.jdbc.convertNcharLiterals 设置为 true。

注意

有关这些属性以及任何性能影响的详情，请查看 Oracle JDBC 驱动程序配置文档。

7.6.2. Unicode 对 Microsoft SQL Server 数据库的支持

Unicode 字符只支持 Microsoft SQL Server 数据库的特殊字段。数据库不需要特殊设置。

JDBC 驱动程序的 sendStringParametersAsUnicode 属性应设置为 false，以显著提高性能。如果没有这个参数，Microsoft SQL Server 可能无法使用索引。

7.6.3. 为 MySQL 数据库配置 Unicode 支持

如果使用 CREATE DATABASE 命令在 VARCHAR 和 CHAR 字段中使用 Unicode 支持创建数据库，则 MySQL 数据库中支持 Unicode 字符。

请注意，因为 utf8mb4 字符集的不同存储要求，不支持 utf8 字符集。详情请查看 MySQL 文档。在这种情况下，非特殊字段的长度限制不适用，因为创建了列来容纳字符数，而不是字节。如果数据库默认字符集不允许 Unicode 存储，则只有特殊字段允许存储 Unicode 值。

1. 启动 MySQL 服务器。
2. 在 JDBC 驱动程序设置下，找到 JDBC 连接设置。
3. 添加此连接属性： characterEncoding=UTF-8

7.6.4. 为 PostgreSQL 数据库配置 Unicode 支持

当数据库字符设置为 UTF8 时，PostgreSQL 数据库支持 Unicode。Unicode 字符可以在没有减少字段长度的任何字段中用于非特殊字段。JDBC 驱动程序不需要特殊设置。字符集在创建 PostgreSQL 数据库时确定。

1. 输入以下 SQL 命令检查 PostgreSQL 集群的默认字符集。

```
show server_encoding;
```

2. 如果默认字符集不是 UTF 8，请使用 UTF8 作为默认字符集创建数据库，例如：

```
create database keycloak with encoding 'UTF8';
```

7.7. 准备 AMAZON AURORA POSTGRESQL

使用 Amazon Aurora PostgreSQL 时， Amazon Web Services JDBC 驱动程序在 Multi-AZ 设置中更改时提供数据库连接的其他功能，如传输数据库连接。此驱动程序不是发行版的一部分，需要先安装它，然后才能使用。

要安装这个驱动程序，请应用以下步骤：

1. 运行解压缩分发时：从 Amazon Web Services JDBC Driver 发行页面下载 JAR 文件，并将其放在红帽构建的 Keycloak 提供程序 文件夹中。

2. 运行容器： 构建自定义红帽构建的 Keycloak 镜像，并在 provider 文件夹中添加 JAR。

   用于构建可用于红帽构建的 Keycloak Operator 的镜像的最小 Dockerfile 类似如下：

   ```
   FROM registry.redhat.io/rhbk/keycloak-rhel9:24
   ADD --chmod=0666 https://github.com/awslabs/aws-advanced-jdbc-wrapper/releases/download/2.3.1/aws-advanced-jdbc-wrapper-2.3.1.jar /opt/keycloak/providers/aws-advanced-jdbc-wrapper.jar
   ```

   如需了解如何 使用红帽构建的 Keycloak 运行优化和未优化 的镜像，请参阅在容器中运行 Keycloak 的红帽构建的 Keycloak。

3. 将红帽构建的 Keycloak 配置为使用以下参数运行：

   db-url

   将 aws-wrapper 插入到常规 PostgreSQL JDBC URL 中，生成 jdbc:aws-wrapper:postgresql://.. 等 URL。

   db-driver

   设置为 software.amazon.jdbc.Driver，以使用 AWS JDBC 包装器。

   transaction-xa-enabled

   设置为 false，因为 Amazon Web Services JDBC 驱动程序不支持 XA 事务。

7.8. 准备 MYSQL 服务器

从 MySQL 8.0.30 开始，MySQL 支持为任何没有显式主密钥（详情请参阅 ）创建的任何 InnoDB 表生成不可见的主密钥。https://dev.mysql.com/doc/refman/8.0/en/create-table-gipks.html如果启用了这个功能，数据库架构初始化和迁移将失败，并显示出错信息 多个主键定义(1068)。然后，在安装或升级 Red Hat build of Keycloak 前，您需要将 MySQL 服务器配置中的 sql_generate_invisible_primary_key 参数设置为 OFF 来禁用它。

7.9. 在集群配置中更改数据库锁定超时

由于集群节点可以同时引导，所以它们需要额外的时间进行数据库操作。例如，引导服务器实例可以执行一些数据库迁移、导入或首次初始化。数据库锁定可防止在集群节点同时引导时相互冲突启动操作。

这个锁定的最大超时时间为 900 秒。如果节点等待这个锁定超过超时时间，则引导会失败。需要更改默认值不太可能，但您可以通过输入以下命令进行更改：

```
bin/kc.[sh|bat] start --spi-dblock-jpa-lock-wait-timeout 900
```

7.10. 使用没有 XA 事务支持的云供应商

红帽构建的 Keycloak 默认使用 XA 事务和适当的数据库驱动程序。某些供应商（如 Azure SQL 和 MariaDB Galera）不支持或依赖 XA 事务机制。要使用没有 XA 事务支持的 Keycloak 的红帽构建，请使用适当的 JDBC 驱动程序，请输入以下命令：

```
bin/kc.[sh|bat] build --db=<vendor> --transaction-xa-enabled=false
```

红帽构建的 Keycloak 会自动为您的供应商选择适当的 JDBC 驱动程序。

7.11. 为 MIGRATIONSTRATEGY 设置 JPA 供应商配置选项

要设置 JPA migrationStrategy (manual/update/validate)，您应该设置 JPA 供应商，如下所示：

为 connections-jpa SPI 的 quarkus 提供者设置 migration-strategy

```
bin/kc.[sh|bat] start --spi-connections-jpa-quarkus-migration-strategy=manual
```

另外，要获取 DB 初始化的 SQL 文件，您必须添加此额外的 SPI initializeEmpty (true/false)：

为 connections-jpa SPI 的 quarkus 供应商设置 initialize-empty

```
bin/kc.[sh|bat] start --spi-connections-jpa-quarkus-initialize-empty=false
```

与 migrationExport 相同，以指向特定文件和位置：

为 connections-jpa SPI 的 quarkus 供应商设置 migration-export

```
bin/kc.[sh|bat] start --spi-connections-jpa-quarkus-migration-export=<path>/<file.sql>
```

7.12. 相关选项

	value
db 数据库供应商。 CLI: **--db** Env: **KC_DB**	**dev-file**（默认）、**dev-mem**、**mariadb**、**mssql**、**mysql**、**oracle**、**postgres**
db-driver JDBC 驱动程序的完全限定类名称。 如果没有设置，则会将默认驱动程序相应地设置为所选数据库。 CLI: **--db-driver** Env: **KC_DB_DRIVER**	
db-password 数据库用户的密码。 CLI: **--db-password** Env: **KC_DB_PASSWORD**	
db-pool-initial-size 连接池的初始大小。 CLI: **--db-pool-initial-size** Env: **KC_DB_POOL_INITIAL_SIZE**	

	value
db-pool-max-size 连接池的最大大小。 CLI: **--db-pool-max-size** Env: **KC_DB_POOL_MAX_SIZE**	**100** （默认）
db-pool-min-size 连接池的最小大小。 CLI: **--db-pool-min-size** Env: **KC_DB_POOL_MIN_SIZE**	
db-schema 要使用的数据库架构。 CLI: **--db-schema** Env: **KC_DB_SCHEMA**	
db-url 完整的数据库 JDBC URL。 如果没有提供，会根据所选数据库厂商设置默认 URL。例如，如果使用 **postgres**，默认的 JDBC URL 将是 **jdbc:postgresql://localhost/keycloak**。 CLI: **--db-url** Env: **KC_DB_URL**	
db-url-database 设置所选供应商的默认 JDBC URL 的数据库名称。 如果设置了 **db-url** 选项，则忽略这个选项。 CLI: **--db-url-database** Env: **KC_DB_URL_DATABASE**	
db-url-host 设置所选供应商的默认 JDBC URL 的主机名。 如果设置了 **db-url** 选项，则忽略这个选项。 CLI: **--db-url-host** Env: **KC_DB_URL_HOST**	

	value
db-url-port 设置所选供应商的默认 JDBC URL 的端口。 如果设置了 **db-url** 选项，则忽略这个选项。 CLI: **--db-url-port** Env: **KC_DB_URL_PORT**	
db-url-properties 设置所选供应商的默认 JDBC URL 的属性。 确保将属性相应地设置为数据库供应商期望的格式，并在此属性值的开头附加正确的字符。如果设置了 **db-url** 选项，则忽略这个选项。 CLI: **--db-url-properties** Env: **KC_DB_URL_PROPERTIES**	
db-username 数据库用户的用户名。 CLI: **--db-username** Env: **KC_DB_USERNAME**	
transaction-xa-enabled 如果设置为 false，则 Keycloak 在数据库不支持 XA 事务时使用非 XA 数据源。 CLI: **--transaction-xa-enabled** Env: **KC_TRANSACTION_XA_ENABLED**	**true**（默认）、**false**

第 8 章 配置分布式缓存

红帽构建的 Keycloak 专为高可用性和多节点集群设置而设计。当前的分布式缓存实施基于 Infinispan，它是一个高性能、可分布式内存数据网格。

8.1. 启用分布式缓存

当您以 production 模式启动红帽 Keycloak 时，会使用 start 命令启用缓存，并发现网络中的所有 Keycloak 节点构建。

默认情况下，缓存使用 UDP 传输堆栈，以便节点可以使用基于 UDP 的 IP 多播传输来发现节点。对于大多数生产环境，UDP 有更好的发现替代方案。红帽构建的 Keycloak 允许您从一组预定义的默认传输堆栈中选择，或者定义自己的自定义堆栈，因为本章稍后将看到。

要显式启用分布式 infinispan 缓存，请输入以下命令：

```
bin/kc.[sh|bat] build --cache=ispn
```

当您以开发模式启动 Keycloak 的红帽构建时，Red Hat build of Keycloak 仅使用本地缓存和分布式缓存，通过隐式设置 --cache=local 选项会完全禁用。本地 缓存模式仅用于开发和测试目的。

8.2. 配置缓存

Red Hat build of Keycloak 提供了一个缓存配置文件，其默认设置位于 conf/cache-ispn.xml。

缓存配置是常规 Infinispan 配置文件。

下表介绍了红帽对 Keycloak 使用的特定缓存。您可以在 conf/cache-ispn.xml 中配置这些缓存：

缓存名称	缓存类型	描述
realms	Local	缓存持久的域数据
users	Local	缓存持久的用户数据
授权	Local	缓存持久的授权数据

缓存名称	缓存类型	描述
keys	Local	缓存外部公钥
work	复制	在节点间传播无效消息
authenticationSessions	分布式	缓存身份验证会话，在身份验证过程中创建/销毁/过期
会话	分布式	缓存用户会话，在退出、令牌撤销或过期时成功进行身份验证并销毁
clientSessions	分布式	缓存客户端会话，在成功验证特定客户端并在注销、令牌撤销或到期时销毁
offlineSessions	分布式	缓存离线用户会话，在成功身份验证并销毁后创建，令牌撤销或因为过期
offlineClientSessions	分布式	缓存客户端会话，在成功验证特定客户端并在注销、令牌撤销或到期时销毁
loginFailures	分布式	跟踪失败的登录，fraud 检测
actionTokens	分布式	缓存操作令牌

8.2.1. 缓存类型和默认值

本地缓存

红帽构建的 Keycloak 会在本地缓存持久性数据，以避免不必要的往返到数据库。

以下数据使用本地缓存保存在集群中的每个节点上：

- 域和 相关数据，如客户端、角色和组。
- 用户和 相关数据，如授予角色和组成员身份。
- 授权和 相关数据，如资源、权限和策略。

- keys

域、用户和授权的本地缓存被配置为每个默认包含 10,000 个条目。默认情况下，本地密钥缓存可以保存最多 1,000 个条目，并且默认为每一小时的过期。因此，密钥会被强制从外部客户端或身份提供程序定期下载。

为了获得最佳运行时并避免额外的往返数据库，您应该考虑查看每个缓存的配置，以确保您的数据库大小一致。您可以缓存更多条目，但服务器通常需要从数据库获取数据。您应该评估内存使用率和性能之间的利弊。

本地缓存无效

本地缓存提高了性能，但在多节点设置中添加了一个挑战。

当一个红帽构建的 Keycloak 节点更新共享数据库中的数据时，所有其他节点都需要了解它，因此它们会使其缓存中的数据无效。

work 缓存是一个复制缓存，用于发送这些无效消息。此缓存中的条目/消息非常短，您不应预期这个缓存随时间增长。

身份验证会话

当用户试图身份验证时，都会创建身份验证会话。身份验证过程完成或达到其过期时间后，它们会自动销毁。

authenticationSessions 分布式缓存用于存储身份验证会话以及在身份验证过程中与之关联的任何其他数据。

通过依赖可分布式缓存，身份验证会话可供集群中的任何节点使用，以便用户能够重定向到任何节点，而不会丢失其身份验证状态。但是，生产就绪部署应始终考虑会话关联，并优先将用户重定向到最初创建会话的节点。通过这样做，您将避免在节点间不必要的状态传输，并提高 CPU、内存和网络利用率。

用户会话

验证用户后，会创建一个用户会话。用户会话会跟踪您的活跃用户及其状态，以便他们可以无缝地向任何应用程序进行身份验证，而无需再次要求提供其凭证。对于每个应用程序，用户也会创建使用客户端会话进行身份验证，以便服务器可以跟踪用户按应用程序进行身份验证的应用程序及其状态。

当用户执行注销时，用户和客户端会话会自动销毁，客户端会执行令牌撤销，或者因为达到其过期时间。

以下缓存用于存储用户和客户端会话：

- 会话
- clientSessions

通过依赖可分布式缓存，用户和客户端会话可供集群中的任何节点使用，以便用户能够重定向到任何节点，而无需丢失其状态。但是，生产就绪部署应始终考虑会话关联，并优先将用户重定向到最初创建会话的节点。通过这样做，您将避免在节点间不必要的状态传输，并提高 CPU、内存和网络利用率。

作为 OpenID Connect 提供程序，服务器还可以对用户进行身份验证并发出离线令牌。与常规用户和客户端会话类似，当服务器成功发布离线令牌时，服务器还会创建离线用户会话和离线客户端会话。但是，由于离线令牌的性质，离线会话会以不同的方式处理，因为它们是长期的，且应在完整的集群关闭后保留。因此，它们也会保留在数据库中。

以下缓存用于存储离线会话：

- offlineSessions
- offlineClientSessions

在集群重启后，离线会话会从数据库完全加载，并使用上述两个缓存保存在共享缓存中。

密码 brute 强制检测

loginFailures 分布式缓存用于跟踪登录尝试失败的数据。这个缓存是 Brute Force Protection 功能需要在多节点 Red Hat build of Keycloak 设置中工作。

操作令牌

当用户需要异步确认操作时，可以使用操作令牌，例如在忘记密码流发送的电子邮件

中。actionTokens 分布式缓存用于跟踪操作令牌的元数据。

8.2.2. 配置缓存以实现可用性

分布式缓存在集群中的节点子集上复制缓存条目，并将条目分配给固定所有者节点。

每个分布式缓存都默认有两个所有者，这意味着两个节点具有特定缓存条目的副本。非所有者节点查询特定缓存的所有者来获取数据。当两个所有者节点都离线时，所有数据都会丢失。这种情况通常会导致用户在下次请求中注销，并必须再次登录。

默认所有者数量足以在至少三个节点的群集设置中保留 1 个节点（所有者）故障。您可以自由地相应地更改所有者数量，以更好地满足您的可用性要求。要更改所有者数量，请打开 conf/cache-ispn.xml，并将分布式缓存的 owners=<value > 的值改为您所需的值。

8.2.3. 指定您自己的缓存配置文件

要指定您自己的缓存配置文件，请输入以下命令：

```
bin/kc.[sh|bat] build --cache-config-file=my-cache-file.xml
```

配置文件相对于 conf/ 目录。

8.2.4. 远程服务器的 CLI 选项

对于红帽构建的 Keycloak 服务器配置，用于高可用性和多节点集群设置，以下 CLI 选项 cache-remote-host、cache-remote-port、cache-remote-username 和 cache-remote-password 简化 XML 文件中的配置。出现任何声明的 CLI 参数后，XML 文件中应该没有与远程存储相关的配置。

8.3. 传输堆栈

传输堆栈可确保集群中的分布式缓存节点以可靠的方式进行通信。红帽构建的 Keycloak 支持广泛的传输堆栈：

- tcp

- udp
- kubernetes
- ec2
- azure
- google

要应用特定的缓存堆栈，请输入以下命令：

```
bin/kc.[sh|bat] build --cache-stack=<stack>
```

启用分布式缓存时，默认堆栈设置为 udp。

8.3.1. 可用的传输堆栈

下表显示了比使用 --cache-stack 构建选项的情况下可用的传输堆栈，而无需进一步配置：

堆栈名称	传输协议	Discovery（发现）
tcp	TCP	MPING （使用 UDP 多播）。
udp	UDP	UDP 多播

下表显示了使用 --cache-stack 构建选项和最小配置可用的传输堆栈：

堆栈名称	传输协议	Discovery（发现）

堆栈名称	传输协议	Discovery（发现）
kubernetes	TCP	DNS_PING （需要 **-Djgroups.dns.query=<headless-service-FQDN >**）添加到 JAVA_OPTS 或 JAVA_OPTS_APPEND 环境变量中。

8.3.2. 其他传输堆栈

下表显示了红帽构建的 Keycloak 支持的传输堆栈，但需要一些额外的步骤才能工作。请注意，这些堆栈都不是 Kubernetes / OpenShift 堆栈，因此如果您希望在 Google Kubernetes 引擎之上运行红帽构建的 Keycloak，则不需要启用 google 堆栈。*在这种情况下，使用 kubernetes 堆栈。相反，当您在 AWS EC2 实例上运行分布式缓存设置时，您需要将堆栈设置为 ec2，因为 ec2 不支持默认的发现机制，如 UDP。*

堆栈名称	传输协议	Discovery（发现）
ec2	TCP	NATIVE_S3_PING
google	TCP	GOOGLE_PING2
azure	TCP	AZURE_PING

特定于云供应商的堆栈具有红帽构建的 Keycloak 的额外依赖项。有关这些依赖项的存储库的更多信息和链接，请参阅 Infinispan 文档。

要为红帽构建的 Keycloak 提供依赖项，请在 供应商 目录中放置相应的 JAR，并通过输入以下命令构建红帽 Keycloak 构建：

```
bin/kc.[sh|bat] build --cache-stack=<ec2|google|azure>
```

8.3.3. 自定义传输堆栈

如果部署没有可用的传输堆栈，您可以更改您的缓存配置文件并定义您自己的传输堆栈。

如需了解更多详细信息，请参阅 使用内联 JGroups 堆栈。

定义自定义传输堆栈

```
<jgroups>
    <stack name="my-encrypt-udp" extends="udp">
    <SSL_KEY_EXCHANGE keystore_name="server.jks"
        keystore_password="password"
        stack.combine="INSERT_AFTER"
        stack.position="VERIFY_SUSPECT2"/>
        <ASYM_ENCRYPT asym_keylength="2048"
        asym_algorithm="RSA"
        change_key_on_coord_leave = "false"
        change_key_on_leave = "false"
        use_external_key_exchange = "true"
        stack.combine="INSERT_BEFORE"
        stack.position="pbcast.NAKACK2"/>
    </stack>
</jgroups>

<cache-container name="keycloak">
    <transport lock-timeout="60000" stack="my-encrypt-udp"/>
    ...
</cache-container>
```

默认情况下，设置为 **cache-stack** 选项的值优先于您在缓存配置文件中定义的传输堆栈。如果您要定义自定义堆栈，请确保 **cache-stack** 选项不用于自定义更改生效。

8.4. 保护缓存通信

当前的 **Infinispan** 缓存实施应该通过各种安全措施（如 **RBAC**、**ACL** 和传输堆栈加密）进行保护。

JGroups 处理红帽构建的 **Keycloak** 服务器之间的所有通信，并支持用于 **TCP** 通信的 **Java SSL** 套接字。红帽构建的 **Keycloak** 使用 **CLI** 选项配置 **TLS** 通信，而无需创建自定义 **JGroups** 堆栈或修改缓存 **XML** 文件。

要启用 **TLS**，**cache-embedded-mtls-enabled** 必须设为 **true**。它需要一个带有证书的密钥存储才能使用： **cache-embedded-mtls-key-store-file** 设置密钥存储的路径，**cache-embedded-mtls-key-store-password** 设置密码来解密它。**truststore** 包含接受连接的有效证书，它可以使用 **cache-embedded-mtls-trust-store-file** （到信任存储的路径）和 **cache-embedded-mtls-trust-store-password** （解密它）进行配置。要限制未授权访问，请为每个红帽构建的 **Keycloak** 部署使用一个自签名证书。

有关使用 UDP 或 TCP_NIO2 的 JGroups 堆栈，请参阅 JGroups 加密文档，了解如何设置协议堆栈。

有关保护缓存通信的更多信息，请参阅 Infinispan 安全指南。

8.5. 从缓存公开指标

默认情况下，启用指标时，缓存的指标不会自动公开。有关如何启用指标的详情，请参阅启用红帽构建的 Keycloak 指标。

要为 cache-container 内的所有缓存启用全局指标，您需要更改缓存配置文件（例如： conf/cache-ispn.xml）以便在 cache-container 级别启用 统计信息，如下所示：

为所有缓存启用指标

```
<cache-container name="keycloak" statistics="true">
    ...
</cache-container>
```

同样，您可以通过启用 统计信息 来为每个缓存单独启用指标，如下所示：

为特定缓存启用指标

```
<local-cache name="realms" statistics="true">
    ...
</local-cache>
```

8.6. 相关选项

	value
cache ▮ 定义高可用性的缓存机制。 默认情况下，在生产模式中，使用 **ispn** 缓存在多个服务器节点之间创建集群。在开发模式中，**本地缓存** 会禁用集群，并用于开发和测试目的。 CLI: **--cache** Env: **KC_CACHE**	**ISPN** （默认）、**local**
cache-config-file ▮ 定义应从中加载缓存配置的文件。 配置文件相对于 **conf/** 目录。 CLI: **--cache-config-file** Env: **KC_CACHE_CONFIG_FILE**	
cache-embedded-mtls-enabled 加密 Keycloak 服务器之间的网络通信。 CLI: **--cache-embedded-mtls-enabled** Env: **KC_CACHE_EMBEDDED_MTLS_ENABLED**	**true**,**false** （默认）
cache-embedded-mtls-key-store-file Keystore 文件路径。 Keystore 必须包含由 TLS 协议使用的证书。默认情况下，它会在 conf/ 目录下查找 **cache-mtls-keystore.p12**。 CLI: **--cache-embedded-mtls-key-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_FILE**	
cache-embedded-mtls-key-store-password 用于访问密钥存储的密码。 CLI: **--cache-embedded-mtls-key-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_PASSWORD**	
cache-embedded-mtls-trust-store-file Truststore 文件路径。 它应包含签署证书的可信证书或证书颁发机构。默认情况下，它会在 conf/ 目录下查找 **cache-mtls-truststore.p12**。 CLI: **--cache-embedded-mtls-trust-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_FILE**	

	value
cache-embedded-mtls-trust-store-password 访问 Truststore 的密码。 CLI: **--cache-embedded-mtls-trust-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_PASSWORD**	
cache-remote-host 远程存储配置的远程服务器的主机名。 它替换通过 XML 文件指定的配置的**remote-server** 标签的**host** 属性（请参阅**cache-config-file** 选项）。如果指定了 选项，则需要**cache-remote-username** 和 **cache-remote-password**，并且 XML 文件中的相关配置不应存在。 CLI: **--cache-remote-host** Env: **KC_CACHE_REMOTE_HOST**	
cache-remote-password 远程服务器对远程存储进行身份验证的密码。 它替换通过 XML 文件指定的配置的**digest** 标签的**password** 属性（请参阅**cache-config-file** 选项）。如果指定了 选项，则需要**cache-remote-host** 和 **cache-remote-username**，并且 XML 文件中的相关配置不应存在。 CLI: **--cache-remote-password** Env: **KC_CACHE_REMOTE_PASSWORD**	
cache-remote-port 远程服务器用于远程存储配置的端口。 它替换通过 XML 文件指定的配置的**remote-server** 标签的**port** 属性（请参阅**cache-config-file** 选项）。 CLI: **--cache-remote-port** Env: **KC_CACHE_REMOTE_PORT**	**11222** （默认）
cache-remote-username 远程存储的远程服务器身份验证的用户名。 它替换通过 XML 文件指定的配置的**digest** 标签的**username** 属性（请参阅**cache-config-file** 选项）。如果指定了 选项，则需要**cache-remote-host** 和 **cache-remote-password**，并且 XML 文件中的相关配置不应存在。 CLI: **--cache-remote-username** Env: **KC_CACHE_REMOTE_USERNAME**	

	value
cache-stack ▮ 定义用于集群通信和节点发现的默认堆栈。 只有在 **缓存** 设置为 **ispn** 时，此选项才会生效。默认：udp。 CLI: **--cache-stack** Env: **KC_CACHE_STACK**	**tcp**, **udp**, **kubernetes**, **ec2**, **azure**, **google**

第 9 章 配置传出 HTTP 请求

红帽构建的 Keycloak 通常需要向安全的应用程序和服务发出请求。红帽构建的 Keycloak 使用 HTTP 客户端管理这些传出连接。本章介绍了如何配置客户端、连接池、代理设置、超时等。

9.1. 客户端配置命令

红帽构建的 Keycloak 用于传出通信的 HTTP 客户端高度可配置。要配置红帽构建的 Keycloak 传出 HTTP 客户端，请输入以下命令：

```
bin/kc.[sh|bat] start --spi-connections-http-client-default-<configurationoption>=<value>
```

以下是命令选项：

establish-connection-timeout-millis

建立连接超时前的最长时间（毫秒）。默认 ：未设置。

socket-timeout-millis

两个数据数据包之间的不活跃时间，直到套接字连接超时（以毫秒为单位）。默认： 5000ms

connection-pool-size

用于出站连接的连接池的大小。默认：128。

max-pooled-per-route

每个主机可以共用多少个连接。默认：64。

connection-ttl-millis

以毫秒为单位进行的最大连接时间。默认 ：未设置。

max-connection-idle-time-millis

连接在连接池中保持的最大时间，以毫秒为单位。闲置连接将由后台干净的线程从池中删除。将这个选项设置为 -1 以禁用这个检查。默认：900000。

disable-cookies

启用或禁用 Cookie 的缓存。默认： true。

client-keystore

Java 密钥存储文件的文件路径。此密钥存储包含双向 SSL 的客户端证书。

client-keystore-password

客户端密钥存储的密码。REQUIRED，当设置 client-keystore 时。

client-key-password

客户端私钥的密码。REQUIRED，当设置 client-keystore 时。

proxy-mappings

为传出 HTTP 请求指定代理配置。如需了解更多详细信息，请参阅 第 9.2 节 “传出 HTTP 请求的代理映射”。

disable-trust-manager

如果传出请求需要 HTTPS，且此配置选项被设置为 true，则不必指定信任存储。此设置应只在开发期间和从生产中使用， 因为它将禁用 SSL 证书的验证。默认：false。

9.2. 传出 HTTP 请求的代理映射

要将传出请求配置为使用代理，您可以使用以下标准代理环境变量来配置代理映射：HTTP_PROXY、HTTPS_PROXY 和 NO_PROXY。

- HTTP_PROXY 和 HTTPS_PROXY 变量代表用于传出 HTTP 请求的代理服务器。红帽构建的 Keycloak 不会区分这两个变量。如果您定义了这两个变量，HTTPS_PROXY 优先于代理服务器使用的实际方案。
- NO_PROXY 变量定义不应使用代理的以逗号分隔的主机名列表。对于您指定的每个主机名，所有子域也会从使用代理中排除。

环境变量可以是小写或大写。小写优先。例如，如果您同时定义了 HTTP_PROXY 和 http_proxy，则使用http_proxy。

代理映射和环境变量示例

```
HTTPS_PROXY=https://www-proxy.acme.com:8080
NO_PROXY=google.com,login.facebook.com
```

在这个示例中，会出现以下结果：

- 所有传出请求都使用代理 https://www-proxy.acme.com:8080，除了对 google.com 或 google.com 的任何子域（如 auth.google.com）的请求除外。
- login.facebook.com 及其所有子域都不使用定义的代理，而 groups.facebook.com 使用代理，因为它不是 login.facebook.com 的子域。

9.3. 使用正则表达式的代理映射

使用代理映射的环境变量的替代方法是为红帽构建的 Keycloak 发送的传出请求配置以逗号分隔的 proxy-mappings 列表。proxy-mapping 由基于 regex 的主机名模式和 proxy-uri 组成，使用格式 hostname-pattern;proxy-uri。

例如，请考虑以下正则表达式：

```
.*\.(google|googleapis)\.com
```

输入以下命令应用基于 regex 的主机名模式：

```
bin/kc.[sh|bat] start --spi-connections-http-client-default-proxy-mappings='''*\\\.(google|googleapis)\\\.com;http://www-proxy.acme.com:8080'''
```

要确定传出 HTTP 请求的代理，会出现以下情况：

- 目标主机名与所有配置的主机名模式匹配。
- 使用第一个匹配模式的 proxy-uri。

- 如果没有配置模式匹配主机名，则不会使用代理。

当代理服务器需要身份验证时，以 username:password@ 格式包括代理用户的凭证。例如：

```
.*\.(google|googleapis)\.com;http://proxyuser:password@www-proxy.acme.com:8080
```

proxy-mapping 的正则表达式示例：

```
# All requests to Google APIs use http://www-proxy.acme.com:8080 as proxy
.*\.(google|googleapis)\.com;http://www-proxy.acme.com:8080

# All requests to internal systems use no proxy
.*\.acme\.com;NO_PROXY

# All other requests use http://fallback:8080 as proxy
.*;http://fallback:8080
```

在本例中，会出现以下情况：

- 使用 proxy-uri 的特殊值 NO_PROXY，这意味着没有代理用于与关联的主机名模式匹配的主机。
- catch-all 模式结束 proxy-mappings，为所有传出请求提供默认代理。

9.4. 为 TLS 连接配置可信证书

如需了解如何配置红帽构建的 Keycloak Truststore，以便红帽构建的 Keycloak 能够使用 TLS 执行传出请求，请参阅配置可信证书。

第 10 章 配置可信证书

当红帽构建的 Keycloak 与外部服务通信或具有通过 TLS 的传入连接时，必须验证远程证书，以确保它连接到可信服务器。这是为了防止中间人攻击所必需的。

这些客户端或服务器的证书或签署这些证书的 CA 必须放在信任存储中。然后，将这个 truststore 配置为由红帽构建的 Keycloak 使用。

10.1. 配置 SYSTEM TRUSTSTORE

现有的 Java 默认信任存储证书将始终被信任。如果您需要额外的证书，如果有自签名或内部证书颁发机构没有被 JRE 识别，则它们可以包含在 conf/truststores 目录或子目录中。certs 可能位于 PEM 文件中，也可以是名为 .p12 或 .pfx 的 PKCS12 文件。在 PKCS12 中，证书必须为未加密的 - 这代表没有密码。

如果您需要替代路径，请使用 --truststore-paths 选项指定 PEM 或 PKCS12 文件所在的其他文件或目录。路径相对于您启动红帽构建的 Keycloak 的位置，因此推荐使用绝对路径。如果指定了目录，它将递归扫描为 truststore 文件。

包含了所有适用的 certs，信任存储将通过 javax.net.ssl 属性用作系统默认信任存储，并作为红帽构建的 Keycloak 中内部使用的默认信任存储。

例如：

```
bin/kc.[sh|bat] start --truststore-paths=/opt/truststore/myTrustStore.pfx,/opt/other-truststore/myOtherTrustStore.pem
```

仍可直接设置您自己的 javax.net.ssl truststore 系统属性，但建议使用 --truststore-paths。

10.2. 主机名验证策略

您可以使用 tls-hostname-verifier 属性优化 TLS 连接验证主机名的方式。

- WILDCARD （默认）允许子域名称中的通配符，如 If*foo.com。

- *ANY 表示主机名不会被验证。*

- *使用 STRICT 时，通用名称(CN)必须与主机名完全匹配。*

  请注意，此设置不适用于 LDAP 安全连接，这需要严格的主机名检查。

10.3. 相关选项

	value
tls-hostname-verifier 传出 HTTPS 和 SMTP 请求的 TLS 主机名验证策略。 CLI: **--tls-hostname-verifier** Env: **KC_TLS_HOSTNAME_VERIFIER**	**ANY**,**WILDCARD** (default), **STRICT**
truststore-paths pkcs12 (p12 或 pfx 文件扩展)、PEM 文件或目录包含要用作系统信任存储的文件的目录。 CLI: **--truststore-paths** Env: **KC_TRUSTSTORE_PATHS**	

第 11 章 启用和禁用功能

红帽构建的 Keycloak 具有一些功能，包括一些禁用的功能，如技术预览和已弃用的功能。其他功能会被默认启用，但如果它们不适用于使用红帽构建的 Keycloak，则可以禁用它们。

11.1. 启用功能

一些支持的功能以及所有预览功能都默认禁用。要启用功能，请输入以下命令：

```
bin/kc.[sh|bat] build --features="<name>[,<name>]"
```

例如，要启用 docker 和 token-exchange，请输入以下命令：

```
bin/kc.[sh|bat] build --features="docker,token-exchange"
```

要启用所有预览功能，请输入以下命令：

```
bin/kc.[sh|bat] build --features="preview"
```

启用的功能可能被版本化或未指定版本。如果您使用版本化功能名称，如 feature:v1，则确切的功能版本将会启用，只要它在运行时中仍然存在。如果您使用未指定版本的名称，例如，选择特定支持的功能版本可能会根据以下优先级从发行版本改为发行版本：

1. 最高默认支持版本
2. 最高非默认支持版本
3. 最高已弃用的版本
4. 最高预览版本
5. 最高实验性版本

11.2. 禁用功能

要禁用默认启用的功能，请输入以下命令：

```
bin/kc.[sh|bat] build --features-disabled="<name>[,<name>]"
```

例如，要禁用 模拟，请输入以下命令：

```
bin/kc.[sh|bat] build --features-disabled="impersonation"
```

不允许在 features-disabled 列表中和 features 列表中都有一个功能。

当禁用了这个功能时，这个功能的所有版本都会被禁用。

11.3. 支持的功能

以下列表包含默认启用的支持功能，如果不需要，则可以禁用。

account-api

帐户管理 REST API

account3

帐户控制台版本 3

admin-api

Admin API

admin2

新管理控制台

授权

授权服务

ciba

OpenID Connect Client Initiated Backchannel Authentication (CIBA)

client-policies

客户端配置策略

device-flow

OAuth 2.0 设备授权

hostname-v1

主机名选项 V1

模拟

管理员能够模拟用户

js-adapter

通过 Keycloak 服务器托管 keycloak.js 和 keycloak-authz.js

kerberos

Kerberos

PAR

OAuth 2.0 推送授权请求(PAR)

step-up-authentication

步骤身份验证

web-authn

W3C Web 身份验证(WebAuthn)

11.3.1. 默认禁用

以下列表包含默认禁用的支持功能，并在需要时启用。

docker

Docker Registry 协议

fips

FIPS 140-2 模式

多站点

多站点支持

11.4. 技术预览功能

预览功能默认是禁用的，不建议在生产环境中使用。这些功能可能会更改，或在以后的版本中被删除。

admin-fine-grained-authz

精细的管理权限

client-secret-rotation

客户端 Secret 轮转

dpop

应用程序层上的 OAuth 2.0 演示概念验证

recovery-codes

恢复代码

脚本

使用 JavaScript 编写自定义验证器

token-exchange

令牌交换服务

update-email

更新电子邮件操作

11.5. 已弃用的功能

以下列表包含将在以后的版本中删除的已弃用的功能。这些功能默认为禁用。

account2

帐户控制台版本 2

linkedin-oauth

基于 OAuth 的 LinkedIn 社交身份提供程序

offline-session-preloading

离线会话预加载

11.6. 相关选项

	value

	value
功能 wagon 启用一组一个或多个功能。 CLI: **--features** Env: **KC_FEATURES**	**account-api[:v1]**, **account2[:v1]**, **account3[:v1]**, **admin-api[:v1]**, **admin-fine-grained-authz[:v1]**, **admin2[:v1]**, **authorization[:v1]**, **ciba[:v1]**, **client-policies[:v1]**, **client-secret-rotation[:v1]**, **client-types[:v1]**, **declarative-ui[:v1]**, **device-flow[:v1]**, **docker[:v1]**, **dpop[:v1]**, **dynamic-scopes[:v1]**, **fips[:v1]**, **hostname[:v1]**, **impersonation[:v1]**, **js-adapter[:v1]**, **kerberos[:v1]**, **linkedin-oauth[:v1]**, **login2[:v1]**, **Multi-site[:v1]**, **offline-session-preloading[:v1]**, **oid4vc-vci[:v1]**, **par[:v1]**, **preview**,**recovery-codes[:v1]**, **scripts[:v1]**, **step-up-authentication[:v1]**, **token-exchange[:v1]**, **transient-users[:v1]**, **update-email[:v1]**, **web-authn[:v1]**

	value
features-disabled ▮ 禁用一组一个或多个功能。 CLI: **--features-disabled** Env: **KC_FEATURES_DISABLED**	**account-api**,**account2**,**account3**,**admin-api**,**admin-fine-grained-authz**,**admin2**,**authorization**,**ciba**,**client-policies**,**client-secret-rotation**,**client-types**,**declarative-ui**,**device-flow**,**docker**,**dpop**, dpop ,**dynamic-scopes**,**fips**,**impersonation**,**js-adapter**,**kerberos**,**linkedin-oauth**,**login2**,**multi-site**,**offline-session-preloading**, **oid4vc-vci**,**par**,**preview**,**recovery-codes**,**script**,**step-up-authentication**,**token-exchange**,**transient-users**,**update-email**,**web-authn**

第 12 章 配置供应商

服务器以可扩展性为基础，对于它提供多个服务提供商接口或 SPI，每个服务器都负责为服务器提供特定功能。在本章中，您将了解配置 SPI 的核心概念及其相应的提供程序。

阅读完本章后，您应能够使用概念以及下面解释的步骤来安装、卸载、启用、禁用和配置任何提供程序，包括用于扩展服务器功能以更好地满足您的要求。

12.1. 配置选项格式

可以使用特定的配置格式配置提供程序。格式包括：

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

& lt;spi-id > 是您要配置的 SPI 的名称。

& lt;provider-id > 是您要配置的供应商的 id。这是设置为对应的供应商工厂实现的 id。

<property > 是您要为给定供应商设置的属性的实际名称。

所有这些名称(spi、provider 和 property)都应为小写，如果名称位于 camel-case 中，如 myKeycloakProvider，它应该包括大写字母前的横线(-)，如下所示： my-keycloak-provider。

将 HttpClientSpi SPI 作为示例，SPI 的名称是 connection HttpClient，可用的其中一个提供程序实施名为 default。要设置 connectionPoolSize 属性，您可以使用配置选项，如下所示：

```
spi-connections-http-client-default-connection-pool-size=10
```

12.2. 设置供应商配置选项

在启动服务器时提供提供程序配置选项。有关配置 红帽构建的 Keycloak 中的选项的所有支持配置源和格式。例如，通过命令行选项：

为 connections-http-client SPI 的默认 提供程序设置 connection-pool-size

```
bin/kc.[sh|bat] start --spi-connections-http-client-default-connection-pool-size=10
```

12.3. 配置默认供应商

根据 SPI，多个供应商实现可以共存，但在运行时只能使用其中一个。对于这些 SPI，默认供应商是将激活并在运行时使用的主要实现。

要将供应商配置为默认值，您应该运行 build 命令，如下所示：

将 mycustomprovider 供应商标记为 email-template SPI 的默认提供程序

```
bin/kc.[sh|bat] build --spi-email-template-provider=mycustomprovider
```

在上例中，我们使用 provider 属性来设置要标记为默认值的供应商的 id。

12.4. 启用和禁用供应商

要启用或禁用供应商，您应该运行 build 命令，如下所示：

启用供应商

```
bin/kc.[sh|bat] build --spi-email-template-mycustomprovider-enabled=true
```

要禁用提供程序，可使用同样的命令，并将 enabled 属性设置为 false。

12.5. 安装和卸载供应商

自定义提供程序应打包在 Java 存档(JAR)文件中，并复制到发行版的 提供程序 目录中。之后，您必须运行 build 命令，以便使用 JAR 文件中的实现来更新服务器的提供程序注册表。

为了优化服务器运行时，需要这一步，以便在启动服务器或运行时都知道所有供应商，而不是提前发现。

要卸载提供程序，您应该从 providers 目录中删除 JAR 文件，然后再次运行 build 命令。

12.6. 使用第三方依赖项

在实施提供程序时，您可能需要使用服务器分发中没有的一些第三方依赖项。

在这种情况下，您应该将任何其他依赖项复制到 provider 目录中，并运行 build 命令。完成后，服务器将在运行时为依赖它们的任何供应商提供这些额外的依赖项。

12.7. 参考

- 配置红帽构建的 Keycloak
- 服务器开发人员文档

第 13 章 配置日志记录

红帽构建的 Keycloak 使用 JBoss Logging 框架。以下是可用日志处理程序的高级概述：

- root
  - 控制台(默认)
  - file

13.1. 日志记录配置

日志记录是在红帽构建的 Keycloak 中按类别进行的。您可以为 root 日志级别或更具体的类别配置日志记录，如 org.hibernate 或 org.keycloak。本章论述了如何配置日志记录。

13.1.1. 日志级别

下表定义了可用的日志级别。

级别	描述
FATAL	因无法满足任何类型的请求而造成关键故障。
ERROR	造成无法处理请求的重大错误或问题。
WARN	可能需要立即修正的非关键错误或问题。
INFO	红帽构建的 Keycloak 生命周期事件或重要信息。低频率。
DEBUG	用于调试目的的更多详细信息，如数据库日志。更高的频率。
TRACE	最详细的调试信息。非常高的频率。
ALL	所有日志消息的特殊级别。
OFF	用于完全关闭日志记录的特殊级别（不推荐）。

13.1.2. 配置 root 日志级别

如果没有用于更具体的类别日志记录器的日志级别配置，则改为使用分隔类别。如果没有分隔类别，则使用根日志记录器级别。

要设置 root 日志级别，请输入以下命令：

```
bin/kc.[sh|bat] start --log-level=<root-level>
```

为这个命令使用这些指南：

- 对于 <root-level >，请提供上表中定义的级别。
- 日志级别区分大小写。例如，您可以使用 DEBUG 或 debug。
- 如果您要意外设置日志级别两次，则列表中最后一次出现的日志级别将变为日志级别。例如，如果您包含语法 --log-level="info,...,DEBUG,..."，根日志记录器为 DEBUG。

13.1.3. 配置特定于类别的日志级别

您可以在红帽构建的 Keycloak 中为特定区域设置不同的日志级别。使用这个命令提供需要不同日志级别的以逗号分隔的类别列表：

```
bin/kc.[sh|bat] start --log-level="<root-level>,<org.category1>:<org.category1-level>"
```

应用到某个类别的配置也适用于其子类，除非您包含更为具体的匹配子类别。

Example

```
bin/kc.[sh|bat] start --log-level="INFO,org.hibernate:debug,org.hibernate.hql.internal.ast:info"
```

这个示例设定以下日志级别：

- 所有日志记录器的根日志级别设置为 INFO。
- hibernate 日志级别一般设置为 debug。
- 为了防止 SQL 抽象语法树，创建详细日志输出时，特定的子类别 org.hibernate.hql.internal.ast 设置为 info。因此，SQL 抽象语法树会被省略，而不是出现在 debug 级别。

13.2. 启用日志处理程序

要启用日志处理程序，请输入以下命令：

```
bin/kc.[sh|bat] start --log="<handler1>,<handler2>"
```

可用的处理程序是 控制台和 文件。下面提到的更具体的处理程序配置仅在处理程序添加到此逗号分隔列表中时生效。

13.3. 控制台日志处理程序

控制台日志处理程序默认为启用，为控制台提供无结构日志消息。

13.3.1. 配置控制台日志格式

Red Hat build of Keycloak 使用基于模式的日志格式器，它默认生成人类可读的文本日志。

这些行的日志记录格式模板可以在 root 级别应用。默认格式模板为：

- %d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c](%t)%s%e%n

格式字符串支持下表中的符号：

符号	概述	描述
%%	%	呈现简单的%字符。
%c	类别	呈现日志类别名称。
%d{xxx}	Date	使用由 **java.text.SimpleDateFormat** 定义的给定日期格式 string.String 语法呈现日期
%e	例外	呈现出的异常。
%h	主机名	呈现简单的主机名。
%H	合格主机名	根据操作系统配置，呈现的完全限定主机名可能与简单主机名相同。
%i	进程 ID	呈现当前进程 PID。
%m	完整消息	呈现日志消息和异常（如果抛出）。
%n	newline	呈现特定于平台的行分隔符字符串。
%N	进程名称	呈现当前进程的名称。
%p	级别	呈现消息的日志级别。
%r	相对时间	以毫秒为单位呈现应用程序日志开始的时间。
%s	简单消息	仅呈现没有异常追踪的日志消息。
%t	线程名称	呈现线程名称。
%T{id}	线程 ID	呈现线程 ID。
%z{<zone name>}	timezone	将日志输出的时区设置为 <zone name>。
%L	行号	呈现日志消息的行号。

13.3.2. 设置日志记录格式

要为日志记录行设置日志记录格式，请执行以下步骤：

1. 使用上表构建您需要的格式模板。

2. 输入以下命令：

```
bin/kc.[sh|bat] start --log-console-format='"<format>"'
```

请注意，在调用包含特殊 shell 字符的命令时，您需要使用 CLI 来转义字符 ；因此，请考虑在配置文件中设置它。

示例：缩写完全限定类别名称

```
bin/kc.[sh|bat] start --log-console-format='"%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n"'
```

本例通过将类别名称缩写为三个字符，方法是在模板中设置 [%c{3.}]，而不是默认的 [%c]。

13.3.3. 配置 JSON 或普通控制台日志记录

默认情况下，控制台日志处理程序会将普通非结构化数据记录到控制台。要使用结构化 JSON 日志输出，请输入以下命令：

```
bin/kc.[sh|bat] start --log-console-output=json
```

日志消息示例

```
{"timestamp":"2022-02-25T10:31:32.452+01:00","sequence":8442,"loggerClassName":"org.jboss.logging.Logger","loggerName":"io.quarkus","level":"INFO","message":"Keycloak 18.0.0-SNAPSHOT on JVM (powered by Quarkus 2.7.2.Final) started in 3.253s. Listening on: http://0.0.0.0:8080","threadName":"main","threadId":1,"mdc":{},"ndc":"","hostName":"host-name","processName":"QuarkusEntryPoint","processId":36946}
```

使用 JSON 输出时，会禁用颜色，由 --log-console-format 设置的格式设置将不适用。

要使用非结构化日志记录，请输入以下命令：

```
bin/kc.[sh|bat] start --log-console-output=default
```

日志消息示例：

```
2022-03-02 10:36:50,603 INFO  [io.quarkus] (main) Keycloak 18.0.0-SNAPSHOT on JVM
(powered by Quarkus 2.7.2.Final) started in 3.615s. Listening on: http://0.0.0.0:8080
```

13.3.4. colors

默认禁用无结构日志的带颜色控制台日志输出。颜色可能会提高可读性，但可能会在日志发送到外部日志聚合系统时导致问题。要启用或禁用颜色编码的控制台日志输出，请输入以下命令：

```
bin/kc.[sh|bat] start --log-console-color=<false|true>
```

13.4. 文件日志记录

作为登录到控制台的替代选择，您可以使用非结构化日志记录到文件中。

13.4.1. 启用文件日志记录

默认禁用记录到文件。要启用它，请输入以下命令：

```
bin/kc.[sh|bat] start --log="console,file"
```

在红帽构建的 Keycloak 安装的 data/log 目录中创建了名为 keycloak.log 的日志文件。

13.4.2. 配置日志文件的位置和名称

要更改创建日志文件的位置和文件名，请执行以下步骤：

1. 创建用于存储日志文件的可写目录。

   如果该目录不可写入，则红帽构建的 Keycloak 将正确启动，但会发出警告，且不会创建日志文件。

2. 输入这个命令：

```
bin/kc.[sh|bat] start --log="console,file" --log-file=<path-to>/<your-file.log>
```

13.4.3. 配置文件处理程序格式

要为文件日志处理器配置不同的日志格式，请输入以下命令：

```
bin/kc.[sh|bat] start --log-file-format="<pattern>"
```

有关可用模式配置的更多信息和表，请参阅 第 13.3.1 节 “配置控制台日志格式”。

13.5. 相关选项

	value
log 在逗号分隔列表中启用一个或多个日志处理程序。 CLI: **--log** Env: **KC_LOG**	**控制台、文件**
log-console-color 登录到控制台时启用或禁用颜色。 CLI: **--log-console-color** Env: **KC_LOG_CONSOLE_COLOR**	**true**,**false** （默认）

	value
log-console-format 非结构化控制台日志条目的格式。 如果格式有空格，请使用 "<format>" 转义值。 CLI: **--log-console-format** Env: **KC_LOG_CONSOLE_FORMAT**	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c](%t)%s%e%n** (default)
log-console-output 将日志输出设置为 JSON 或 default (plain)无结构的日志。 CLI: **--log-console-output** Env: **KC_LOG_CONSOLE_OUTPUT**	**Default** (default), **json**
log-file 设置日志文件路径和文件名。 CLI: **--log-file** Env: **KC_LOG_FILE**	**data/log/keycloak.log** （默认）
log-file-format 设置特定于文件日志条目的格式。 CLI: **--log-file-format** Env: **KC_LOG_FILE_FORMAT**	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c](%t)%s%e%n** (default)
log-file-output 将日志输出设置为 JSON 或 default (plain)无结构的日志。 CLI: **--log-file-output** Env: **KC_LOG_FILE_OUTPUT**	**Default** (default), **json**
log-level root 类别的日志级别，或者以逗号分隔的单个类别列表及其级别。 对于 root 类别，您不需要指定类别。 CLI: **--log-level** Env: **KC_LOG_LEVEL**	**[info]** （默认）

第 14 章 FIPS 140-2 支持

Federal Information Processing Standard Publication 140-2 (FIPS 140-2)是一个用来批准加密模块的美国政府计算机安全标准。红帽构建的 Keycloak 支持在 FIPS 140-2 兼容模式下运行。在这种情况下，红帽构建的 Keycloak 只会将 FIPS 批准的加密算法用于其功能。

要在 FIPS 140-2 中运行，红帽构建的 Keycloak 应该在启用了 FIPS 140-2 的系统中运行。这个要求通常假定在安装过程中启用了 FIPS 的 RHEL 或 Fedora。详情请查看 RHEL 文档。当系统处于 FIPS 模式时，它会确保底层 OpenJDK 处于 FIPS 模式，且只使用 启用了 FIPS 的安全供应商。

要检查系统是否处于 FIPS 模式，您可以从命令行使用以下命令检查它：

```
fips-mode-setup --check
```

如果系统没有处于 FIPS 模式，您可以使用以下命令启用它，但建议系统处于 FIPS 模式，因为安装而不是随后启用它：

```
fips-mode-setup --enable
```

14.1. BOUNCYCASTLE LIBRARY

红帽构建的 Keycloak 在内部将 BouncyCastle 库用于很多加密工具。请注意，红帽构建的 Keycloak 附带的 BouncyCastle 库的默认版本不兼容 FIPS，但 BouncyCastle 还提供了其库的 FIPS 验证版本。由于许可证限制，红帽构建的 Keycloak 无法附带 FIPS 验证的 BouncyCastle 库，红帽构建的 Keycloak 无法提供对它的官方支持。因此，若要以 FIPS 兼容模式运行，您需要下载 BouncyCastle-FIPS 位，并将它们添加到 Keycloak 发行版本的红帽构建中。当红帽构建的 Keycloak 在 fips 模式下执行时，它将使用 BCFIPS 位而不是默认的 BouncyCastle 位，它实现了 FIPS 合规性。

14.1.1. BouncyCastle FIPS 位

BouncyCastle FIPS 可以从 BouncyCastle 官方页面 下载。然后，您可以将其添加到发行版的 KEYCLOAK_HOME/providers 目录中。确保使用与 BouncyCastle 红帽构建的 Keycloak 依赖项兼容的正确版本。所需的 BCFIPS 位是：

- bc-fips-1.0.2.3.jar
- bctls-fips-1.0.18.jar

- bcpkix-fips-1.0.7.jar

14.2. 生成密钥存储

您可以创建 pkcs12 或 bcfks 密钥存储，以用于红帽构建的 Keycloak 服务器 SSL。

14.2.1. PKCS12 密钥存储

p12 （或 pkcs12）密钥存储（和/或信任存储）在 BCFIPS 非批准模式下可以正常工作。

PKCS12 密钥存储可以以标准的方式在 RHEL 9 上使用 OpenJDK 17 Java 生成。例如，以下命令可用于生成密钥存储：

```
keytool -genkeypair -sigalg SHA512withRSA -keyalg RSA -storepass passwordpassword \
  -keystore $KEYCLOAK_HOME/conf/server.keystore \
  -alias localhost \
  -dname CN=localhost -keypass passwordpassword
```

当系统处于 FIPS 模式时，默认 java.security 文件会被修改以使用启用了 FIPS 的安全供应商，因此不需要额外的配置。另外，在 PKCS12 密钥存储中，您只能使用 keytool 命令存储 PBE （基于密码的加密）密钥，这使其成为将红帽构建的 Keycloak KeyStore Vault 和/或将配置属性存储在 KeyStore Config Source 中。如需了解更多详细信息，请参阅配置红帽构建的 Keycloak 和 使用密码库。

14.2.2. BCFKS 密钥存储

BCFKS 密钥存储生成需要使用 BouncyCastle FIPS 库和自定义安全文件。

您可以先创建一个帮助程序文件，如 /tmp/kc.keystore-create.java.security。文件的内容只需要具有以下属性：

```
securerandom.strongAlgorithms=PKCS11:SunPKCS11-NSS-FIPS
```

接下来，输入如下命令来生成密钥存储：

```
keytool -keystore $KEYCLOAK_HOME/conf/server.keystore \
  -storetype bcfks \
  -providername BCFIPS \
```

```
-providerclass org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider \
-provider org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider \
-providerpath $KEYCLOAK_HOME/providers/bc-fips-*.jar \
-alias localhost \
-genkeypair -sigalg SHA512withRSA -keyalg RSA -storepass passwordpassword \
-dname CN=localhost -keypass passwordpassword \
-J-Djava.security.properties=/tmp/kc.keystore-create.java.security
```

警告

使用自签名证书仅用于演示目的，因此当您迁移到生产环境时，将这些证书替换为正确的证书。

当您使用 bcfks 类型的密钥存储/truststore 进行任何其他操作时需要类似的选项。

14.3. 运行服务器。

要在非批准模式下使用 BCFIPS 运行服务器，请输入以下命令

```
bin/kc.[sh|bat] start --features=fips --hostname=localhost --https-key-store-password=passwordpassword --log-level=INFO,org.keycloak.common.crypto:TRACE,org.keycloak.crypto:TRACE
```

注意

在非批准模式下，默认的密钥存储类型（以及默认信任存储类型）是 PKCS12。因此，如果您按照上述步骤生成 BCFKS 密钥存储，则还需要使用 --https-key-store-type=bcfks 命令。如果要使用 truststore，可能需要类似的命令。

注意

如果一切按预期工作，您可以禁用生产环境中的日志记录。

14.4. STRICT 模式

有 fips-mode 选项，该选项会在启用 fips 功能时自动设置为 non-strict。这意味着，在"非批准的模式下运行 BCFIPS"。更安全的替代方案是使用 --features=fips --fips-mode=strict，在这种情况

下，BouncyCastle FIPS 将使用"approved 模式"。使用该选项会对加密和安全算法产生更严格的安全要求。

注意

在 strict 模式中，默认的密钥存储类型（以及默认的信任存储类型）是 BCFKS。如果要使用不同的密钥存储类型，则需要在适当的类型中使用 --https-key-store-type 选项。如果要使用 truststore，可能需要类似的命令。

在启动服务器时，您可以检查启动日志是否包含 KC 供应商，请注意 批准模式，如下所示：

```
KC(BCFIPS version 1.000203 Approved Mode, FIPS-JVM: enabled) version 1.0 - class
org.keycloak.crypto.fips.KeycloakFipsSecurityProvider,
```

14.4.1. 严格模式中的加密限制

- 如上一节中所述，strict 模式可能无法用于 pkcs12 密钥存储。需要使用前面提到的另一个密钥存储（如 bcfks）。在使用 strict 模式时，红帽构建的 Keycloak 不支持 jks 和 pkcs12 密钥存储。有些示例是在管理控制台或 realm 键中的 java-keystore 供应商导入或生成 OIDC 或 SAML 客户端的密钥存储。

- 用户密码必须是 14 个字符或更长时间。红帽构建的 Keycloak 默认使用基于 PBKDF2 的密码编码。BCFIPS 批准模式要求密码至少使用 PBKDF2 算法使用 112 位（有效 14 个字符）。如果要允许较短的密码，将 SPI password -hashing 的供应商 pbkdf2-sha256 的供应商 pbkdf2-sha256 的属性 max-padding 设置为值 14 以在验证此算法创建的哈希时提供额外的 padding。此设置还向后兼容之前存储的密码。例如，如果用户的数据库位于非FIPS 环境中，并且您有较短的密码，并且您希望在批准模式中使用 BCFIPS 的红帽构建 Keycloak，则密码应该可以正常工作。因此，您可以在启动服务器时使用如下选项：

```
--spi-password-hashing-pbkdf2-sha256-max-padding-length=14
```

注意

使用上述选项不会破坏 FIPS 合规性。但请注意，不再有密码是很好的做法。例如，现代浏览器自动生成的密码与此要求匹配，因为它们超过 14 个字符。

- RSA 密钥 1024 位无法正常工作(2048 是最小的)。这适用于红帽构建的 Keycloak 域本身使用的密钥（管理控制台中 Keys 选项卡的Realm 密钥），但也适用于客户端密钥和 IDP 密钥

-

HMAC SHA-XXX 密钥必须至少为 112 位（或 14 个字符长）。例如，如果您使用带有客户端身份验证的 OIDC 客户端，带有 Client Secret （或 OIDC 标记中的 client-secret-jwt ），则您的客户端 secret 应该至少为 14 个字符。请注意，为了获得良好的安全性，建议使用红帽构建的 Keycloak 服务器生成的客户端 secret，这始终不满意。

14.5. 其他限制

要让 SAML 正常工作，请确保您的安全供应商中提供了 XMLDSig 安全提供程序。要使 Kerberos 正常工作，请确保有 SunJGSS 安全供应商可用。在 OpenJDK 17.0.6 中启用了 FIPS 的 RHEL 9 中，这些安全供应商不在 java.security 中，这意味着它们有效地无法正常工作。

要使 SAML 正常工作，您可以手动将提供程序添加到 JAVA_HOME/conf/security/java.security 中，到列表 fips 供应商中。例如，添加如下行：

```
fips.provider.7=XMLDSig
```

添加此安全供应商应该可以正常工作。实际上，它兼容 FIPS，可能在以后的 OpenJDK 17 微版本中默认添加。详情包括在 bugzilla 中。

注意

建议查看 JAVA_HOME/conf/security/java.security，并在此处检查所有配置的供应商，并确保数字匹配。换句话说，fips.provider.7 假设已经有 6 个提供商配置了此文件中的 fips.provider.N 等前缀。

如果您不希望在 java 本身内编辑 java.security 文件，您可以创建自定义 java 安全文件（例如，名为 kc.java.security），并将上面的单个属性添加到该文件中。然后，使用附加此属性文件启动您的红帽 Keycloak 服务器构建：

```
-Djava.security.properties=/location/to/your/file/kc.java.security
```

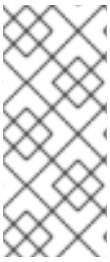
对于 Kerberos/SPNEGO，安全供应商 SunJGSS 尚未完全兼容 FIPS。因此，如果要兼容 FIPS，不建议将其添加到安全供应商列表中。当在 FIPS 平台上执行时，红帽构建的 Keycloak 中默认禁用 KERBEROS 功能，当安全供应商不可用时。详情包括在 bugzilla 中。

14.6. 在 FIPS 主机上运行 CLI

如果要运行 Client Registration CLI (kcreg.sh|bat 脚本)或 Admin CLI (kcadm.sh|bat 脚本)，CLI 还必须使用 BouncyCastle FIPS 依赖项而不是纯 BouncyCastle 依赖项。要达到此目的，您可以将 jars 复

制到 CLI 库文件夹，且足够了。当检测到相应的 BCFIPS jars 存在时，CLI 工具将自动使用 BCFIPS 依赖项而不是普通 BCFIPS jars （请参阅上面的用于使用的版本）。例如，在运行 CLI 前使用类似以下内容的命令：

```
cp $KEYCLOAK_HOME/providers/bc-fips-*.jar $KEYCLOAK_HOME/bin/client/lib/
cp $KEYCLOAK_HOME/providers/bctls-fips-*.jar $KEYCLOAK_HOME/bin/client/lib/
```



注意

当试图通过 CLI 使用 BCFKS truststore/keystore 时，您可能会看到问题，因为此信任存储不是默认的 java 密钥存储类型。在 java 安全属性中，最好将其指定为默认值。例如，在使用 kcadm|kcreg 客户端执行任何操作前，在 unix 的系统上运行此命令：

```
echo "keystore.type=bcfks
fips.keystore.type=bcfks" > /tmp/kcadm.java.security
export KC_OPTS="-Djava.security.properties=/tmp/kcadm.java.security"
```

14.7. 红帽在容器中以 FIPS 模式构建 KEYCLOAK 服务器

如果您希望红帽以 FIPS 模式在 FIPS 模式下构建 Keycloak 时，您的"主机"必须使用 FIPS 模式。容器随后将从父主机"inherit" FIPS 模式。详情请查看 RHEL 文档中的这个部分。https://access.redhat.com/documentation/zh-cn/red_hat_enterprise_linux/9/html/security_hardening/using-the-system-wide-cryptographic-policies_security-hardening#enabling-fips-mode-in-a-container_using-the-system-wide-cryptographic-policies

当从 FIPS 模式的主机执行时，红帽构建的 Keycloak 容器镜像将自动处于 fips 模式。但是，请确保红帽构建的 Keycloak 容器也使用 BCFIPS jars （而不是 BC jars）并在启动时正确选项。

因此，最好在容器中运行 红帽构建的 Keycloak 来构建自己的容器镜像，并调整 使用 BCFIPS 等。

例如，您可以在当前目录中创建子目录 并添加 ：

- 如上所述，BC FIPS jar 文件
- 自定义密钥存储文件 - 名为 keycloak-fips.keystore.bcfks

- 安全文件 kc.java.security 带有为 SAML 添加的供应商

然后，在当前目录中创建 Dockerfile 类似如下：

Dockerfile：

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:24 as builder

ADD files /tmp/files/

WORKDIR /opt/keycloak
RUN cp /tmp/files/*.jar /opt/keycloak/providers/
RUN cp /tmp/files/keycloak-fips.keystore.* /opt/keycloak/conf/server.keystore
RUN cp /tmp/files/kc.java.security /opt/keycloak/conf/

RUN /opt/keycloak/bin/kc.sh build --features=fips --fips-mode=strict

FROM registry.redhat.io/rhbk/keycloak-rhel9:24
COPY --from=builder /opt/keycloak/ /opt/keycloak/

ENTRYPOINT ["/opt/keycloak/bin/kc.sh"]
```

然后，构建 FIPS 作为优化的 Docker 镜像，并按照容器中运行的红帽 Keycloak 所述启动它。这些步骤要求您使用启动镜像时所描述的参数。

14.8. 从非 FIPS 环境迁移

如果您之前在非 fips 环境中使用了 Keycloak 的红帽构建，则可以将其迁移到 FIPS 环境，包括其数据。但是，如上一节所述，存在限制和注意事项，即：

- 确保所有红帽构建的 Keycloak 功能依赖于密钥存储s 仅使用受支持的密钥存储类型。这根据是否使用严格的或非限制模式而有所不同。

- Kerberos 身份验证可能无法正常工作。如果您的身份验证流使用 Kerberos 身份验证器，则此验证器将在迁移到 FIPS 环境时自动切换到 DISABLED。建议您从您的域中删除任何 Kerberos 用户存储供应商，并在切换到 FIPS 环境前在 LDAP 供应商中禁用 Kerberos 相关功能。

除了前面的要求外，请务必在切换到 FIPS 严格模式前再次检查它：

- 确保所有红帽构建的 Keycloak 功能都依赖于密钥（如域或客户端密钥）至少使用 2048 位的 RSA 密钥
- 确保依赖 Signed JWT 带有客户端 Secret 的客户端至少使用 14 个字符长 secret （最好生成的 secret）
- 如前文所述，密码长度限制。如果您的用户有较短的密码，请务必像前面所述，将 max padding length 设置为 14 个 PBKDF2 供应商时启动服务器。如果您希望避免这个选项，您可以要求所有用户在新环境中第一次身份验证期间重置其密码(例如，禁止密码 链接)。

14.9. 在非 FIPS 系统中构建 KEYCLOAK FIPS 模式

Red Hat build of Keycloak 在启用了 FIPS 的 RHEL 8 系统和 ubi8 镜像上被支持并测试。它还支持 RHEL 9 （和 ubi9 镜像）。在非 RHEL 兼容平台或非FIPS 启用平台上运行，无法严格保证 FIPS 合规性，且无法正式支持。

如果您仍然仅限于在这样的系统上运行红帽构建的 Keycloak，您可以至少更新在 java.security 文件中配置的安全供应商。在这个版本中，没有 FIPS 合规性，但至少设置更接近它。它可以通过提供自定义安全文件来实现，且仅提供覆盖的安全供应商列表，如前面所述。有关推荐供应商列表，请参阅 OpenJDK 17 文档。

您可以在启动时检查红帽 Keycloak 服务器日志的构建，以查看是否使用了正确的安全供应商。应为与 Keycloak 软件包相关的加密红帽构建启用 TRACE 日志记录，如前面 Keycloak 启动命令所述。

第 15 章 启用红帽构建的 KEYCLOAK 健康检查

红帽构建的 Keycloak 内置了对健康检查的支持。本章论述了如何启用和使用红帽构建的 Keycloak 健康检查。

15.1. 红帽构建的 KEYCLOAK 健康检查端点

红帽构建的 Keycloak 会公开 4 个健康端点：

- /health/live
- /health/ready
- /health/started
- /health

有关每个端点的含义的信息，请参阅 Quarkus SmallRye Health docs。

这些端点在成功或 503 Service Unavailable 时响应 HTTP 状态 200 OK，以及类似如下的 JSON 对象：

在没有额外的 per-check 信息的情况下对端点成功响应：

```
{
    "status": "UP",
    "checks": []
}
```

使用数据库连接的信息成功响应端点：

```
{
    "status": "UP",
    "checks": [
        {
            "name": "Keycloak database connections health check",
            "status": "UP"
        }
    ]
}
```

15.2. 启用健康检查

可以使用 启用了构建时间选项的构建选项启用健康检查 ：

```
bin/kc.[sh|bat] build --health-enabled=true
```

默认情况下，不会从健康端点返回检查。

15.3. 使用健康检查

建议由外部 HTTP 请求监控健康端点。由于安全措施，从红帽构建的 Keycloak 容器镜像中删除 curl 和其他软件包，本地的监控将无法正常工作。

如果您没有在容器中使用红帽构建的 Keycloak，请使用任何访问健康检查端点的内容。

15.3.1. curl

您可以使用简单的 HTTP HEAD 请求来确定红帽构建的 Keycloak 的实时或 就绪状态。curl 是良好的 HTTP 客户端，用于此目的。

如果红帽构建的 Keycloak 部署在容器中，则必须因为前面提到的安全措施而从它外部运行这个命令。例如：

```
curl --head -fsS http://localhost:8080/health/ready
```

如果命令返回状态为 0，则红帽构建的 Keycloak 处于活动状态或 就绪，具体取决于您调用的端

点。否则会出现问题。

15.3.2. Kubernetes

定义 HTTP 探测，以便 Kubernetes 可以外部监控健康端点。不要使用存活度命令。

15.3.3. HEALTHCHECK

Dockerfile 镜像 HEALTHCHECK 指令定义了一个命令，它将在容器运行时定期执行。红帽构建的 Keycloak 容器没有安装任何 CLI HTTP 客户端。考虑将 curl 安装为额外的 RPM，如 容器运行红帽构建的 Keycloak 中 详述。请注意，由于此原因，您的容器可能不太安全。

15.4. 可用的检查

下表显示了可用的检查。

检查	描述	需要指标
数据库	返回数据库连接池的状态。	是

对于某些检查，您还需要启用 Requires Metrics 列所示的指标。要启用指标，请使用 启用了 metrics 的选项，如下所示：

```
bin/kc.[sh|bat] build --health-enabled=true --metrics-enabled=true
```

15.5. 相关选项

	value
health-enabled ▮ 如果服务器应公开健康检查端点。 如果启用，则健康检查位于 **/health**、**/health/ready** 和 **/health/live** 端点中。 CLI: **--health-enabled** Env: **KC_HEALTH_ENABLED**	**true**,**false** （默认）

第 16 章 启用红帽构建的 KEYCLOAK 指标

红帽构建的 Keycloak 内置了指标支持。本章论述了如何启用和配置服务器指标。

16.1. 启用指标

可以使用 启用了构建时间选项指标启用指标 ：

```
bin/kc.[sh|bat] start --metrics-enabled=true
```

16.2. 查询指标

红帽构建的 Keycloak 在以下端点公开指标：

- /metrics

端点的响应使用 application/openmetrics-text 内容类型， 它基于 Prometheus (OpenMetrics)文本格式。片断 bellow 是响应的示例：

```
# HELP base_gc_total Displays the total number of collections that have occurred. This attribute lists
-1 if the collection count is undefined for this collector.
# TYPE base_gc_total counter
base_gc_total{name="G1 Young Generation",} 14.0
# HELP jvm_memory_usage_after_gc_percent The percentage of long-lived heap pool used after the
last GC event, in the range [0..1]
# TYPE jvm_memory_usage_after_gc_percent gauge
jvm_memory_usage_after_gc_percent{area="heap",pool="long-lived",} 0.0
# HELP jvm_threads_peak_threads The peak live thread count since the Java virtual machine
started or peak was reset
# TYPE jvm_threads_peak_threads gauge
jvm_threads_peak_threads 113.0
# HELP agroal_active_count Number of active connections. These connections are in use and not
available to be acquired.
# TYPE agroal_active_count gauge
agroal_active_count{datasource="default",} 0.0
# HELP base_memory_maxHeap_bytes Displays the maximum amount of memory, in bytes, that
can be used for memory management.
# TYPE base_memory_maxHeap_bytes gauge
base_memory_maxHeap_bytes 1.6781410304E10
# HELP process_start_time_seconds Start time of the process since unix epoch.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.675188449054E9
# HELP system_load_average_1m The sum of the number of runnable entities queued to available
```

```
processors and the number of runnable entities running on the available processors averaged over a period of time
# TYPE system_load_average_1m gauge
system_load_average_1m 4.005859375

...
```

16.3. 可用指标

下表总结了可用的指标组：

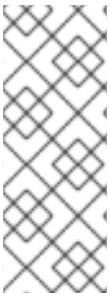
指标	描述
System	与 CPU 和内存用量相关的一组系统级指标。
JVM	与 GC 和堆相关的 Java 虚拟机(JVM)的一组指标。
数据库	数据库连接池中的一组指标（如果使用数据库）。
Cache	来自 Infinispan 缓存的一组指标。如需了解更多详细信息，请参阅配置分布式缓存。

16.4. 相关选项

	value
metrics-enabled 如果服务器应该公开指标。 如果启用，指标位于 **/metrics** 端点。 CLI: **--metrics-enabled** Env: **KC_METRICS_ENABLED**	**true**,**false** （默认）

第 17 章 导入和导出域

在本章中，您将了解使用 JSON 文件导入和导出域的不同方法。



注意

导出和导入单个文件可能会生成大型文件，因此如果您的数据库包含 500 多个用户，则导出到目录而不是单个文件。使用目录性能更好，因为目录提供商为每个"页面"（用户的文件）使用单独的事务。每个文件和每个事务的用户的默认计数为 fifty。增大这个值会导致执行时间指数级增长。

17.1. 为数据库连接参数提供选项

当使用下面的 导出和导入 命令时，红帽构建的 Keycloak 需要了解如何连接到存储域、客户端、用户和其他实体的信息的数据库。如 配置红帽构建的 Keycloak 所述，该信息可作为命令行参数、环境变量或配置文件提供。对每个命令使用 --help 命令行选项来查看可用选项。

有些配置选项是构建时间配置选项。默认情况下，如果红帽检测到构建时间参数的变化，红帽构建的 Keycloak 将自动针对 导出和导入 命令重新构建。

如果您已使用 build 命令构建了一个经过优化的 Keycloak 版本，如 配置红帽构建的 Keycloak 所述，请使用命令行选项 --optimized 使红帽构建的 Keycloak 会跳过构建检查，以加快启动时间。在执行此操作时，从命令行中删除构建时间选项，并只保留运行时选项。

17.2. 将 REALM 导出至目录

要导出域，您可以使用 export 命令。在调用此命令时，不能启动您的红帽构建的 Keycloak 服务器实例。

```
bin/kc.[sh|bat] export --help
```

要将域导出到目录，您可以使用 --dir <dir> 选项。

```
bin/kc.[sh|bat] export --dir <dir>
```

将域导出到目录时，服务器将为要导出的每个域创建单独的文件。

17.2.1. 配置如何导出用户

您还可以通过设置 **--users <strategy >** 选项来配置如何导出用户。此选项的值有：

different_files

用户会根据 **--users-per-file** 设置的每个文件的最大用户数导出到不同的 **json** 文件中。这是默认值。

skip

跳过导出用户。

realm_file

用户将导出到与域设置相同的文件。对于名为 **"foo"** 的域，这将是**"foo-realm.json"**，域数据和用户。

same_file

所有用户都导出到一个明确的文件。因此，您将获得一个域的两个 **json** 文件，一个包含 **realm** 数据和一个用户。

如果您要使用 **different_files** 策略导出用户，您可以通过设置 **--users-per-file** 选项来设置所需的每个文件的用户数量。默认值为 **50**。

```
bin/kc.[sh|bat] export --dir <dir> --users different_files --users-per-file 100
```

17.3. 将 REALM 导出至文件

要将域导出到文件，您可以使用 **--file <file>** 选项。

```
bin/kc.[sh|bat] export --file <file>
```

将域导出到文件时，服务器将使用相同的 文件来存储正在导出的所有域的配置。

17.4. 导出特定域

如果您没有指定要导出的特定域，则会导出所有域。要导出单个域，您可以使用 **--realm** 选项，如下所示：

```
bin/kc.[sh|bat] export [--dir|--file] <path> --realm my-realm
```

17.5. 从目录导入 REALM

要导入域，您可以使用 import 命令。在调用此命令时，不能启动您的红帽构建的 Keycloak 服务器实例。

```
bin/kc.[sh|bat] import --help
```

将域导出到目录后，您可以使用 --dir <dir& gt; 选项将域导入服务器，如下所示：

```
bin/kc.[sh|bat] import --dir <dir>
```

当使用 import 命令导入域时，您可以设置现有域是否被跳过，或者应该使用新配置覆盖它们。为此，您可以设置 --override 选项，如下所示：

```
bin/kc.[sh|bat] import --dir <dir> --override false
```

默认情况下，-- override 选项被设置为 true，因此域始终被新配置覆盖。

17.6. 从文件导入 REALM

要导入之前在单个文件中导出的域，您可以使用 --file <file> 选项，如下所示：

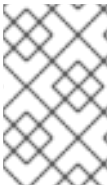
```
bin/kc.[sh|bat] import --file <file>
```

17.7. 在启动过程中导入 REALM

您还可以使用 --import-realm 选项，在服务器启动时导入域。

```
bin/kc.[sh|bat] start --import-realm
```

当您设置 --import-realm 选项时，服务器将尝试从 data/import 目录中导入任何 realm 配置文件。只有使用 .json 扩展名的常规文件才会从该目录读取，会忽略子目录。



注意

对于红帽构建的 Keycloak 容器，导入目录为 /opt/keycloak/data/import

如果服务器中已存在域，则会跳过导入操作。这个行为的主要原因是避免重新创建域并在服务器重启之间可能松散状态。

要重新创建域，您应该在启动服务器前显式运行 import 命令。

不支持导入 master 域，因为它是一个非常敏感的操作。

17.7.1. 在 Realm 配置文件中使用环境变量

在启动时导入域时，您可以使用占位符解析任何域配置的环境变量中的值。

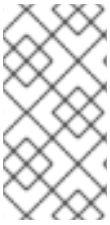
使用占位符的域配置

```
{
    "realm": "${MY_REALM_NAME}",
    "enabled": true,
    ...
}
```

在上例中，将值设为 MY_REALM_NAME 环境变量来设置 realm 属性。

17.8. 使用管理控制台导入和导出

您还可以使用管理控制台导入和导出域。此功能与前面部分中描述的其他 CLI 选项不同，因为管理控制台仅提供 部分 导出域的功能。在这种情况下， 可以导出当前的域设置，以及客户端、角色和组等某些资源。无法 使用此方法导出该域的用户。



注意

使用 Admin Console 导出时，realm 和 selected 资源始终导出到名为 realm-export.json 的文件中。另外，密码和客户端 secret 等所有敏感值都使用 * 符号屏蔽。

要使用管理控制台导出域，请执行以下步骤：

1. 选择一个 realm。
2. 点菜单中的 Realm settings。
3. 指向域设置屏幕右上角的 Action 菜单，然后选择 Partial export。

   一个资源列表以及 realm 配置一起会出现。
4. 选择您要导出的资源。
5. 单击 Export。

注意

从管理控制台导出的域不适用于在服务器之间备份或数据传输。只有 CLI 导出适合服务器之间备份或数据传输。

警告

如果域包含多个组、角色和客户端，则操作可能会导致服务器在一段时间内响应用户请求。请谨慎使用此功能，特别是在生产系统上。

同样，您可以导入之前导出的域。执行以下步骤：

1. 点菜单中的 Realm settings。
2. 指向域设置屏幕右上角的 Action 菜单，然后选择 Partial import。

   此时会出现一个提示，您可以在其中选择要导入的文件。根据此文件，您可以看到您可以导入的资源以及 realm 设置。

3. 点 Import。

如果导入的资源已存在，您还可以控制红帽构建的 Keycloak 应该做什么。这些选项存在：

导入失败

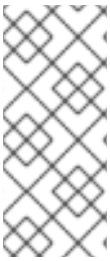
中止导入。

skip

跳过重复资源而不中止进程

覆盖

将现有资源替换为正在导入的资源。



注意

管理控制台部分导入也可以导入 CLI export 命令创建的文件。换句话说，可以使用管理控制台导入 CLI 创建的完整导出。如果文件中包含用户，则这些用户也可以导入到当前域中。

第 18 章 使用密码库

红帽构建的 Keycloak 提供 Vault SPI 的两个开箱即用实现：基于纯文本文件的 vault 和基于 Java KeyStore 的 vault。

基于文件的库实现对于 Kubernetes/OpenShift secret 特别有用。您可以将 Kubernetes secret 挂载到红帽构建的 Keycloak Container 中，数据字段将位于挂载的文件夹中，并带有一个 flat-file 结构。

基于 Java KeyStore 的库实现对于在裸机安装中存储 secret 非常有用。您可以使用 KeyStore vault，该 vault 使用密码加密。

18.1. 可用的集成

存储在密码库中的 secret 可以在管理控制台的以下位置使用：

- 获取 SMTP 邮件服务器密码
- 使用基于 LDAP 的用户联邦时获取 LDAP 绑定凭证
- 集成外部身份提供程序时获取 OIDC 身份提供程序客户端 Secret

18.2. 启用密码库

要启用基于文件的库，您需要首先使用以下构建选项构建 Keycloak：

```
bin/kc.[sh|bat] build --vault=file
```

另外，对于基于 Java KeyStore 的来说，您需要指定以下构建选项：

```
bin/kc.[sh|bat] build --vault=keystore
```

18.3. 配置基于文件的库

18.3.1. 将基础目录设置为查找 secret

Kubernetes/OpenShift secret 基本上是挂载的文件。要配置应该挂载这些文件的目录，请输入以下命令：

```
bin/kc.[sh|bat] start --vault-dir=/my/path
```

18.3.2. 特定于域的 secret 文件

Kubernetes/OpenShift Secret 在红帽构建的 Keycloak 基础上使用，这需要对该文件进行命名规则：

```
${vault.<realmname>_<secretname>}
```

18.3.3. 在名称中使用下划线

要正确处理 secret，把 <realmname> 或 <secretname> 中的所有下划线加倍，用单个下划线分开。

Example

- realm Name: sso_realm
- 所需名称： ldap_credential
- 生成的文件名：

```
sso__realm_ldap__credential
```

请注意 sso 和 realm 之间的双重下划线，以及在 ldap 和凭证 之间。

18.4. 配置基于 JAVA KEYSTORE 的库

要使用基于 Java KeyStore 的密码库，您需要首先创建一个 KeyStore 文件。您可以使用以下命令进行此操作：

```
keytool -importpass -alias <realm-name>_<alias> -keystore keystore.p12 -storepass keystorepassword
```

然后，输入您要存储在密码库中的值。请注意，-alias 参数的格式取决于所使用的密钥解析器。默认密钥解析器为 REALM_UNDERSCORE_KEY。

默认情况下，这会导致在 SecretKeyEntry 中以通用 PBEKey （基于密码的加密）的形式存储值。

然后，您可以使用以下运行时选项启动红帽 Keycloak 构建：

```
bin/kc.[sh|bat] start  --vault-file=/path/to/keystore.p12 --vault-pass=<value> --vault-type=
<value>
```

请注意，--vault-type 参数是可选的，默认为 PKCS12。

然后，存储在密码库中的机密可以通过以下占位符访问（假设使用 REALM_UNDERSCORE_KEY 密钥解析器）：${vault.realm-name_alias}。

18.5. 示例：在管理控制台中使用 LDAP 绑定凭证 SECRET

设置示例

- 名为 secrettest的域
- 绑定凭证所需的名称 ldapBc
- 生成的文件名： secrettest_ldapBc

管理控制台中的使用量

然后，您可以在配置 LDAP 用户联邦时，使用 ${vault.ldapBc} 作为 Bind Credential 的值来从管理控制台中使用此 secret。

18.6. 相关选项

	value
Vault criu 启用 vault 供应商。 CLI: **--vault** Env: **KC_VAULT**	**文件，密钥存储**
vault-dir 如果设置，可以通过读取给定目录中文件的内容来获取 secret。 CLI: **--vault-dir** Env: **KC_VAULT_DIR**	
vault-file 密钥存储文件的路径。 CLI: **--vault-file** Env: **KC_VAULT_FILE**	
vault-pass vault 密钥存储的密码。 CLI: **--vault-pass** Env: **KC_VAULT_PASS**	
vault-type 指定密钥存储文件的类型。 CLI: **--vault-type** Env: **KC_VAULT_TYPE**	**PKCS12**（默认）

第 19 章 所有配置

19.1. CACHE

	value
cache 定义高可用性的缓存机制。 默认情况下，在生产模式中，使用 **ispn** 缓存在多个服务器节点之间创建集群。在开发模式中，**本地缓存** 会禁用集群，并用于开发和测试目的。 CLI: **--cache** Env: **KC_CACHE**	**ISPN** （默认）、**local**
cache-config-file 定义应从中加载缓存配置的文件。 配置文件相对于 **conf/** 目录。 CLI: **--cache-config-file** Env: **KC_CACHE_CONFIG_FILE**	
cache-embedded-mtls-enabled 加密 Keycloak 服务器之间的网络通信。 CLI: **--cache-embedded-mtls-enabled** Env: **KC_CACHE_EMBEDDED_MTLS_ENABLED**	**true**,**false** （默认）
cache-embedded-mtls-key-store-file Keystore 文件路径。 Keystore 必须包含由 TLS 协议使用的证书。默认情况下，它会在 conf/ 目录下查找 **cache-mtls-keystore.p12**。 CLI: **--cache-embedded-mtls-key-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_FILE**	
cache-embedded-mtls-key-store-password 用于访问密钥存储的密码。 CLI: **--cache-embedded-mtls-key-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_PASSWORD**	

	value
cache-embedded-mtls-trust-store-file Truststore 文件路径。 它应包含签署证书的可信证书或证书颁发机构。默认情况下，它会在 conf/ 目录下查找 **cache-mtls-truststore.p12**。 CLI: **--cache-embedded-mtls-trust-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_FILE**	
cache-embedded-mtls-trust-store-password 访问 Truststore 的密码。 CLI: **--cache-embedded-mtls-trust-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_PASSWORD**	
cache-remote-host 远程存储配置的远程服务器的主机名。 它替换通过 XML 文件指定的配置的 **remote-server** 标签的 **host** 属性（请参阅 **cache-config-file** 选项）。如果指定了 选项，则需要 **cache-remote-username** 和 **cache-remote-password**，并且 XML 文件中的相关配置不应存在。 CLI: **--cache-remote-host** Env: **KC_CACHE_REMOTE_HOST**	
cache-remote-password 远程服务器对远程存储进行身份验证的密码。 它替换通过 XML 文件指定的配置的 **digest** 标签的 **password** 属性（请参阅 **cache-config-file** 选项）。如果指定了 选项，则需要 **cache-remote-host** 和 **cache-remote-username**，并且 XML 文件中的相关配置不应存在。 CLI: **--cache-remote-password** Env: **KC_CACHE_REMOTE_PASSWORD**	
cache-remote-port 远程服务器用于远程存储配置的端口。 它替换通过 XML 文件指定的配置的 **remote-server** 标签的 **port** 属性（请参阅 **cache-config-file** 选项）。 CLI: **--cache-remote-port** Env: **KC_CACHE_REMOTE_PORT**	**11222**（默认）

	value
cache-remote-username 远程存储的远程服务器身份验证的用户名。 它替换通过 XML 文件指定的配置的 **digest** 标签的 **username** 属性（请参阅 **cache-config-file** 选项）。如果指定了 选项，则需要 **cache-remote-host** 和 **cache-remote-password**，并且 XML 文件中的相关配置不应存在。 CLI: **--cache-remote-username** Env: **KC_CACHE_REMOTE_USERNAME**	
cache-stack ▮ 定义用于集群通信和节点发现的默认堆栈。 只有在 **缓存** 设置为 **ispn** 时，此选项才会生效。默认：udp。 CLI: **--cache-stack** Env: **KC_CACHE_STACK**	**tcp**, **udp**, **kubernetes**, **ec2**, **azure**, **google**

19.2. 数据库

	value
db ▮ 数据库供应商。 CLI: **--db** Env: **KC_DB**	**dev-file**（默认）、**dev-mem**、**mariadb**、**mssql**、**mysql**、**oracle**、**postgres**
db-driver ▮ JDBC 驱动程序的完全限定类名称。 如果没有设置，则会将默认驱动程序相应地设置为所选数据库。 CLI: **--db-driver** Env: **KC_DB_DRIVER**	
db-password 数据库用户的密码。 CLI: **--db-password** Env: **KC_DB_PASSWORD**	

	value
db-pool-initial-size 连接池的初始大小。 CLI: **--db-pool-initial-size** Env: **KC_DB_POOL_INITIAL_SIZE**	
db-pool-max-size 连接池的最大大小。 CLI: **--db-pool-max-size** Env: **KC_DB_POOL_MAX_SIZE**	**100**（默认）
db-pool-min-size 连接池的最小大小。 CLI: **--db-pool-min-size** Env: **KC_DB_POOL_MIN_SIZE**	
db-schema 要使用的数据库架构。 CLI: **--db-schema** Env: **KC_DB_SCHEMA**	
db-url 完整的数据库 JDBC URL。 如果没有提供，会根据所选数据库厂商设置默认 URL。例如，如果使用 **postgres**，默认的 JDBC URL 将是 **jdbc:postgresql://localhost/keycloak**。 CLI: **--db-url** Env: **KC_DB_URL**	
db-url-database 设置所选供应商的默认 JDBC URL 的数据库名称。 如果设置了 **db-url** 选项，则忽略这个选项。 CLI: **--db-url-database** Env: **KC_DB_URL_DATABASE**	

	value
db-url-host 设置所选供应商的默认 JDBC URL 的主机名。 如果设置了 **db-url** 选项，则忽略这个选项。 CLI: **--db-url-host** Env: **KC_DB_URL_HOST**	
db-url-port 设置所选供应商的默认 JDBC URL 的端口。 如果设置了 **db-url** 选项，则忽略这个选项。 CLI: **--db-url-port** Env: **KC_DB_URL_PORT**	
db-url-properties 设置所选供应商的默认 JDBC URL 的属性。 确保将属性相应地设置为数据库供应商期望的格式，并在此属性值的开头附加正确的字符。如果设置了 **db-url** 选项，则忽略这个选项。 CLI: **--db-url-properties** Env: **KC_DB_URL_PROPERTIES**	
db-username 数据库用户的用户名。 CLI: **--db-username** Env: **KC_DB_USERNAME**	

19.3. TRANSACTIONS

	value
transaction-xa-enabled 如果设置为 false，则 Keycloak 在数据库不支持 XA 事务时使用非 XA 数据源。 CLI: **--transaction-xa-enabled** Env: **KC_TRANSACTION_XA_ENABLED**	**true**（默认）、**false**

19.4. 功能

	value
功能 wagon 启用一组一个或多个功能。 CLI: **--features** Env: **KC_FEATURES**	**account-api[:v1]**, **account2[:v1]**, **account3[:v1]**, **admin-api[:v1]**, **admin-fine-grained-authz[:v1]**, **admin2[:v1]**, **authorization[:v1]**, **ciba[:v1]**, **client-policies[:v1]**, **client-secret-rotation[:v1]**, **client-types[:v1]**, **declarative-ui[:v1]**, **device-flow[:v1]**, **docker[:v1]**, **dpop[:v1]**, **dynamic-scopes[:v1]**, **fips[:v1]**, **hostname[:v1]**, **impersonation[:v1]**, **js-adapter[:v1]**, **kerberos[:v1]**, **linkedin-oauth[:v1]**, **login2[:v1]**, **Multi-site[:v1]**, **offline-session-preloading[:v1]**, **oid4vc-vci[:v1]**, **par[:v1]**, **preview**,**recovery-codes[:v1]**, **scripts[:v1]**, **step-up-authentication[:v1]**, **token-exchange[:v1]**, **transient-users[:v1]**, **update-email[:v1]**, **web-authn[:v1]**

	value
features-disabled ▮ 禁用一组一个或多个功能。 CLI: **--features-disabled** Env: **KC_FEATURES_DISABLED**	**account-api**,**account2**,**account3**,**admin-api**,**admin-fine-grained-authz**,**admin2**,**authorization**,**ciba**,**client-policies**,**client-secret-rotation**,**client-types**,**declarative-ui**, **device-flow**,**docker**,**dpop**, dpop ,**dynamic-scopes**,**fips**,**impersonation**,**js-adapter**,**kerberos**,**linkedin-oauth**,**login2**,**multi-site**,**offline-session-preloading**, **oid4vc-vci**,**par**,**preview**,**recovery-codes**,**script**,**step-up-authentication**,**token-exchange**,**transient-users**,**update-email**,**web-authn**

19.5. 主机名

	value
hostname Keycloak 服务器的主机名。 CLI: **--hostname** Env: **KC_HOSTNAME**	
hostname-admin 用于访问管理控制台的主机名。 如果您使用值设置为 hostname 选项以外的主机名公开管理控制台，**请使用这个选项**。 CLI: **--hostname-admin** Env: **KC_HOSTNAME_ADMIN**	

	value
hostname-admin-url 设置用于访问管理控制台的基本 URL，包括方案、主机、端口和路径 CLI: **--hostname-admin-url** Env: **KC_HOSTNAME_ADMIN_URL**	
hostname-debug 切换可通过 /realms/master/hostname-debug 访问的主机名调试页面 CLI: **--hostname-debug** Env: **KC_HOSTNAME_DEBUG**	**true**,**false** （默认）
hostname-path 如果代理为 Keycloak 使用不同的 context-path，则应设置此项。 CLI: **--hostname-path** Env: **KC_HOSTNAME_PATH**	
hostname-port 代理在公开主机名时使用的端口。 如果代理使用默认 HTTP 和 HTTPS 端口以外的端口，则设置这个选项。 CLI: **--hostname-port** Env: **KC_HOSTNAME_PORT**	**-1** （默认）
hostname-strict 禁用从请求标头动态解析主机名。 在生产环境中应始终设置为 true，除非代理验证 Host 标头。 CLI: **--hostname-strict** Env: **KC_HOSTNAME_STRICT**	**true** （默认）、**false**
hostname-strict-backchannel 默认情况下，后端通道 URL 从请求标头动态解析，以允许内部和外部应用程序。 如果所有应用都使用公共 URL，则应启用此选项。 CLI: **--hostname-strict-backchannel** Env: **KC_HOSTNAME_STRICT_BACKCHANNEL**	**true**,**false** （默认）

	value
hostname-url 设置 frontend URL 的基本 URL，包括方案、主机、端口和路径。 CLI: **--hostname-url** Env: **KC_HOSTNAME_URL**	

19.6. HTTP(S)

	value
http-enabled 启用 HTTP 侦听器。 CLI: **--http-enabled** Env: **KC_HTTP_ENABLED**	**true**,**false** （默认）
http-host 使用的 HTTP 主机。 CLI: **--http-host** Env: **KC_HTTP_HOST**	**0.0.0.0** （默认）
http-max-queued-requests 排队的 HTTP 请求的最大数量。 使用它来保证负载过载。过量请求将返回 "503 Server not Available" 响应。 CLI: **--http-max-queued-requests** Env: **KC_HTTP_MAX_QUEUED_REQUESTS**	
http-pool-max-threads 线程的最大数量。 如果没有指定，则会自动调整到最大 8 个()可用处理器的数量和 200。例如，如果有 4 个处理器，最大线程数为 200。如果有 48 个处理器，它将是 384。 CLI: **--http-pool-max-threads** Env: **KC_HTTP_POOL_MAX_THREADS**	
http-port 使用的 HTTP 端口。 CLI: **--http-port** Env: **KC_HTTP_PORT**	**8080** （默认）

	value
http-relative-path ▮ 为服务资源设置相对于 / 的路径。 该路径必须以 / 开头。 CLI: **--http-relative-path** Env: **KC_HTTP_RELATIVE_PATH**	/ （默认）
https-certificate-file PEM 格式的服务器证书或证书链的文件路径。 CLI: **--https-certificate-file** Env: **KC_HTTPS_CERTIFICATE_FILE**	
https-certificate-key-file PEM 格式到私钥的文件路径。 CLI: **--https-certificate-key-file** Env: **KC_HTTPS_CERTIFICATE_KEY_FILE**	
https-cipher-suites 要使用的密码套件。 如果未指定，则会选择合理的默认值。 CLI: **--https-cipher-suites** Env: **KC_HTTPS_CIPHER_SUITES**	
https-client-auth ▮ 将服务器配置为 require/request 客户端身份验证。 CLI: **--https-client-auth** Env: **KC_HTTPS_CLIENT_AUTH**	**none** （默认）、请求、**必需**
https-key-store-file 保存证书信息的密钥存储，而不是指定单独的文件。 CLI: **--https-key-store-file** Env: **KC_HTTPS_KEY_STORE_FILE**	
https-key-store-password 密钥存储文件的密码。 CLI: **--https-key-store-password** Env: **KC_HTTPS_KEY_STORE_PASSWORD**	**密码** （默认）

	value
https-key-store-type 密钥存储文件的类型。 如果没有给定，会根据文件名自动检测类型。如果将 **fips-mode** 设置为 **strict** 且没有设置值，则默认为 **BCFKS**。 CLI: **--https-key-store-type** Env: **KC_HTTPS_KEY_STORE_TYPE**	
https-port 使用的 HTTPS 端口。 CLI: **--https-port** Env: **KC_HTTPS_PORT**	**8443**（默认）
https-protocols 要显式启用的协议列表。 CLI: **--https-protocols** Env: **KC_HTTPS_PROTOCOLS**	**[TLSv1.3,TLSv1.2]**（默认）
https-trust-store-file 包含要信任的证书信息的信任存储。 CLI: **--https-trust-store-file** Env: **KC_HTTPS_TRUST_STORE_FILE** **已弃用。**使用 System Truststore 替代，请参阅文档了解详情。	
https-trust-store-password 信任存储文件的密码。 CLI: **--https-trust-store-password** Env: **KC_HTTPS_TRUST_STORE_PASSWORD** **已弃用。**使用 System Truststore 替代，请参阅文档了解详情。	
https-trust-store-type 信任存储文件的类型。 如果没有给定，会根据文件名自动检测类型。如果将 **fips-mode** 设置为 **strict** 且没有设置值，则默认为 **BCFKS**。 CLI: **--https-trust-store-type** Env: **KC_HTTPS_TRUST_STORE_TYPE** **已弃用。**使用 System Truststore 替代，请参阅文档了解详情。	

19.7. HEALTH

	value
health-enabled ▮ 如果服务器应公开健康检查端点。 如果启用，则健康检查位于 **/health**、**/health/ready** 和 **/health/live** 端点中。 CLI: **--health-enabled** Env: **KC_HEALTH_ENABLED**	**true**,**false** （默认）

19.8. CONFIG

	value
config-keystore 指定 KeyStore 配置源的路径。 CLI: **--config-keystore** Env: **KC_CONFIG_KEYSTORE**	
config-keystore-password 指定 KeyStore 配置源的密码。 CLI: **--config-keystore-password** Env: **KC_CONFIG_KEYSTORE_PASSWORD**	
config-keystore-type 指定 KeyStore 配置源的类型。 CLI: **--config-keystore-type** Env: **KC_CONFIG_KEYSTORE_TYPE**	**PKCS12** （默认）

19.9. 指标

	value
metrics-enabled ▮ 如果服务器应该公开指标。 如果启用，指标位于 **/metrics** 端点。 CLI: **--metrics-enabled** Env: **KC_METRICS_ENABLED**	**true**,**false** （默认）

19.10. PROXY

	value
proxy 如果服务器位于反向代理后面，则代理地址转发模式。 CLI: **--proxy** Env: **KC_PROXY** **已弃用。**使用： **proxy-headers**。	**none**（默认）、边缘、**重新加密**、**透传**
proxy-headers 服务器应接受的代理标头。 错误配置可能会使服务器暴露给安全漏洞。优先于已弃用的代理选项。 CLI: **--proxy-headers** Env: **KC_PROXY_HEADERS**	转发，**xforwarded**

19.11. VAULT

	value
Vault criu 启用 vault 供应商。 CLI: **--vault** Env: **KC_VAULT**	**文件**，**密钥存储**
vault-dir 如果设置，可以通过读取给定目录中文件的内容来获取 secret。 CLI: **--vault-dir** Env: **KC_VAULT_DIR**	
vault-file 密钥存储文件的路径。 CLI: **--vault-file** Env: **KC_VAULT_FILE**	

	value
vault-pass vault 密钥存储的密码。 CLI: **--vault-pass** Env: **KC_VAULT_PASS**	
vault-type 指定密钥存储文件的类型。 CLI: **--vault-type** Env: **KC_VAULT_TYPE**	**PKCS12** （默认）

19.12. 日志记录

	value
log 在逗号分隔列表中启用一个或多个日志处理程序。 CLI: **--log** Env: **KC_LOG**	**控制台**、**文件**
log-console-color 登录到控制台时启用或禁用颜色。 CLI: **--log-console-color** Env: **KC_LOG_CONSOLE_COLOR**	**true**,**false** （默认）
log-console-format 非结构化控制台日志条目的格式。 如果格式有空格，请使用 "<format>" 转义值。 CLI: **--log-console-format** Env: **KC_LOG_CONSOLE_FORMAT**	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c](%t)%s%e%n** (default)
log-console-output 将日志输出设置为 JSON 或 default (plain)无结构的日志。 CLI: **--log-console-output** Env: **KC_LOG_CONSOLE_OUTPUT**	**Default** (default), **json**

	value
log-file 设置日志文件路径和文件名。 CLI: **--log-file** Env: **KC_LOG_FILE**	**data/log/keycloak.log** （默认）
log-file-format 设置特定于文件日志条目的格式。 CLI: **--log-file-format** Env: **KC_LOG_FILE_FORMAT**	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c](%t)%s%e%n** (default)
log-file-output 将日志输出设置为 JSON 或 default (plain)无结构的日志。 CLI: **--log-file-output** Env: **KC_LOG_FILE_OUTPUT**	**Default** (default), **json**
log-level root 类别的日志级别，或者以逗号分隔的单个类别列表及其级别。 对于 root 类别，您不需要指定类别。 CLI: **--log-level** Env: **KC_LOG_LEVEL**	**[info]** （默认）

19.13. TRUSTSTORE

	value
tls-hostname-verifier 传出 HTTPS 和 SMTP 请求的 TLS 主机名验证策略。 CLI: **--tls-hostname-verifier** Env: **KC_TLS_HOSTNAME_VERIFIER**	**ANY**,**WILDCARD** (default), **STRICT**
truststore-paths pkcs12 (p12 或 pfx 文件扩展)、PEM 文件或目录包含要用作系统信任存储的文件的目录。 CLI: **--truststore-paths** Env: **KC_TRUSTSTORE_PATHS**	

19.14. 安全性

	value
fips-mode ▮ 设置 FIPS 模式。 如果设置了 **非限制**，则启用了 FIPS，但在非批准模式下。要获得完整的 FIPS 合规性，请将 **strict** 设置为在批准的模式下运行。**当禁用 fips** 功能时，这个选项默认为禁用，这是默认设置。当启用 **fips** 功能时，这个选项默认为 **non-strict**。 CLI: **--fips-mode** Env: **KC_FIPS_MODE**	**non-strict**,**strict**

19.15. EXPORT

	value
dir 设置使用导出的数据创建文件的目录路径。 CLI: **--dir** Env: **KC_DIR**	
realm 将域的名称设置为 export。 如果没有设置，则会导出所有域。 CLI: **--realm** Env: **KC_REALM**	
用户 设置如何导出用户。 CLI: **--users** Env: **KC_USERS**	**skip**,**realm_file**,**same_file**,**different_files** (default)
users-per-file 设置每个文件的用户数。 只有在 **用户** 设置为 **different_files** 时，才会使用它。增加这个数字会导致导出时间指数增加。 CLI: **--users-per-file** Env: **KC_USERS_PER_FILE**	**50** （默认）

19.16. IMPORT

	value
file 设置要读取的文件的路径。 CLI: **--file** Env: **KC_FILE**	
override 设置现有数据是否应覆盖。 如果设置为 false，则忽略数据。 CLI: **--override** Env: **KC_OVERRIDE**	**true**（默认）、**false**

第 20 章 所有供应商配置

20.1. AUTHENTICATION-SESSIONS

20.1.1. Infinispan

	value
spi-authentication-sessions-infinispan-auth-sessions-limit 每个 RootAuthenticationSession 的最大并发身份验证会话数。 CLI: **--spi-authentication-sessions-infinispan-auth-sessions-limit** Env: **KC_SPI_AUTHENTICATION_SESSIONS_INFINISPAN_AUTH_SESSIONS_LIMIT**	**300** （默认）或任何 **int**

20.2. CIBA-AUTH-CHANNEL

20.2.1. ciba-http-auth-channel

	value
spi-ciba-auth-channel-ciba-http-auth-channel-http-authentication-channel-uri 身份验证频道的 HTTP (S) URI。 CLI: **--spi-ciba-auth-channel-ciba-http-auth-channel-http-authentication-channel-uri** Env: **KC_SPI_CIBA_AUTH_CHANNEL_CIBA_HTTP_AUTH_CHANNEL_HTTP_AUTHENTICATION_CHANNEL_URI**	**任何字符串**

20.3. CONNECTIONS-HTTP-CLIENT

20.3.1. default

	value
spi-connections-http-client-default-client-key-password 密钥密码。 CLI: **--spi-connections-http-client-default-client-key-password** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEY_PASSWORD**	**-1**（默认）或**任何字符串**
spi-connections-http-client-default-client-keystore 密钥存储的文件路径，从中读取密钥资料到设置 TLS 连接。 CLI: **--spi-connections-http-client-default-client-keystore** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEYSTORE**	**任何字符串**
spi-connections-http-client-default-client-keystore-password 密钥存储密码。 CLI: **--spi-connections-http-client-default-client-keystore-password** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEYSTORE_PASSWORD**	**任何字符串**
spi-connections-http-client-default-connection-pool-size 分配最大连接值。 CLI: **--spi-connections-http-client-default-connection-pool-size** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_POOL_SIZE**	任何 **int**
spi-connections-http-client-default-connection-ttl-millis 为持久连接设置最长时间（以毫秒为单位）。 CLI: **--spi-connections-http-client-default-connection-ttl-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_TTL_MILLIS**	**-1**（默认）或任何长

	value
spi-connections-http-client-default-disable-cookies 禁用状态(cookie)管理。 CLI: **--spi-connections-http-client-default-disable-cookies** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_DISABLE_COOKIES**	**true** （默认）、**false**
spi-connections-http-client-default-disable-trust-manager 禁用信任管理和主机名验证。 请注意，这是一个安全漏洞，因此只有当您无法或不想验证您要通信的主机身份时，才设置这个选项。 CLI: **--spi-connections-http-client-default-disable-trust-manager** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_DISABLE_TRUST_MANAGER**	**true**,**false** （默认）
spi-connections-http-client-default-establish-connection-timeout-millis 尝试进行初始套接字连接时，超时是什么？ CLI: **--spi-connections-http-client-default- establish-connection-timeout-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_ESTABLISH_CONNECTION_TIMEOUT_MILLIS**	**-1** （默认）或任何 长
spi-connections-http-client-default-max-connection-idle-time-millis 设置从池中驱除闲置连接的时间（以毫秒为单位）。 CLI: **--spi-connections-http-client-default-max-connection-idle-time-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_MAX_IDLE_TIME_MILLIS**	**900000** （默认）或任何 长
spi-connections-http-client-default-max-pooled-per-route 为每个路由值分配最大连接。 CLI: **--spi-connections-http-client-default-max-pooled-per-route** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_MAX_POOLED_PER_ROUTE**	**64** （默认）或任何 **int**

	value
spi-connections-http-client-default-proxy-mappings 以 hostnamePattern;proxyUri 的形式表示基于 regex 的主机名模式和 proxy-uri 的组合。 CLI: **--spi-connections-http-client-default-proxy-mappings** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_PROXY_MAPPINGS**	**任何字符串**
spi-connections-http-client-default-reuse-connections 如果连接应该被重复使用。 CLI: **--spi-connections-http-client-default-reuse-connections** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_REUSE_CONNECTIONS**	**true**（默认）、**false**
spi-connections-http-client-default-socket-timeout-millis 套接字不活跃超时。 CLI: **--spi-connections-http-client-default-socket-timeout-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_SOCKET_TIMEOUT_MILLIS**	**5000**（默认）或任何 长

20.4. CONNECTIONS-INFINISPAN

20.4.1. Quarkus

	value
spi-connections-infinispan-quarkus-site-name 多站点部署的站点名称 CLI: **--spi-connections-infinispan-quarkus-site-name** Env: **KC_SPI_CONNECTIONS_INFINISPAN_QUARKUS_SITE_NAME**	**任何字符串**

20.5. CONNECTIONS-JPA

20.5.1. Quarkus

	value
spi-connections-jpa-quarkus-initialize-empty 如果为空，则初始化数据库。 如果设置为 false，则必须手动初始化数据库。如果要手动将数据库设置 migrationStrategy 改为 manual，这会使用 SQL 命令创建文件来初始化数据库。 CLI: **--spi-connections-jpa-quarkus-initialize-empty** Env: **KC_SPI_CONNECTIONS_ hypervisor_QUARKUS_INITIALIZE_EMPTY**	**true**（默认）、**false**
spi-connections-jpa-quarkus-migration-export 编写手动数据库初始化/迁移文件的路径。 CLI: **--spi-connections-jpa-quarkus-migration-export** Env: **KC_SPI_CONNECTIONS_ hypervisor_QUARKUS_MIGRATION_EXPORT**	**任何字符串**
spi-connections-jpa-quarkus-migration-strategy 用于迁移数据库的策略。 有效值为 update、manual 和 validate。update 将自动迁移数据库架构。手动将使用 SQL 命令将所需的更改导出到文件，您可以手动对数据库执行。验证将简单检查数据库是最新的。 CLI: **--spi-connections-jpa-quarkus-migration-strategy** Env: **KC_SPI_CONNECTIONS_setuptools_QUARKUS_MIGRATION_STRATEGY**	**更新**（默认）、**手动**、**验证**

20.6. COOKIE

20.6.1. default

	value
spi-cookie-default-same-site-legacy 添加没有 SameSite 参数的传统 Cookie CLI: **--spi-cookie-default-same-site-legacy** Env: **KC_SPI_COOKIE_DEFAULT_SAME_SITE_LEGACY**	**true**（默认）、**false**

20.7. DBLOCK

20.7.1. jpa

	value
spi-dblock-jpa-lock-wait-timeout 等待数据库锁定的最长时间。 CLI: **--spi-dblock-jpa-lock-wait-timeout** Env: **KC_SPI_DBLOCK_SPEC_LOCK_WAIT_TIMEOUT**	任何 **int**

20.8. EVENTS-LISTENER

20.8.1. email

	value
spi-events-listener-email-exclude-events 以逗号分隔的事件列表，这些事件不应通过电子邮件发送给用户帐户。 CLI: **--spi-events-listener-email-exclude-events** Env: **KC_SPI_EVENTS_LISTENER_EMAIL_EXCLUDE_EVENTS**	**authreqid_to_token**,**authreqid_to_token_error**,**client_delete_error**,**client_info**,**client_info_error**,**client_initiated_account_linking**,**client_initiated_account_linking_error**, **client_login**,**client_login_error**,**client_register _error**, client_update ,**client_update_error**,**code_to_token**,**code_to_token_error**,**custom_required_action**, **custom_required_action_error**,**delete_account**,**delete_account_error**,**execute_action_token**,**execute_action_token_error**,**execute_actions**,**execute_actions_error**,**federated_identity_link**,**federated_identity_link_error**, **grant_consent**,**grant_consent_error**,**identity_provider_first_login**,**identity_provider_first_login_error**,**identity_provider_link_account**,**identity_provider_link_account_**

	value
	error,identity_provider_login, identity_provider_login_error,identity_provider_post_login,identity_provider_post_login_error,identity_provider_response_error,identity_provider_retrieve_token,identity_provider_retrieve_token_error, impersonate,impersonate_error,introspection_token,introspection_token_error,invalid_signature,invalid_signature_error,login,login_error,logout,logout_error, oauth2_device_auth, oauth2_device_auth_error,oauth2_device_code_to_token,oauth2_device_code_to_token_error,oauth2_device_verify_user_code,oauth2_device_verify_user_code_error,oauth2_extension_grant, oauth2_extension_grant_error,permission_token,permission_token_error,pushed_authorization_request _error , refresh_token ,refresh_token_error, register, register_error, register_node,register_node_error,remove_federated_identity, remove_federated_identity_error,remove_totp,remove_totp_error,reset_password_error , restart_authentication ,restart_authentication, restart_authentication_error,revoke_grant_error,send_identity_provider_link,send_identity_provider_lin

	value
	k_error,send_reset_password,send_reset_password_error, send_verify_email,send_verify_email_error,token_exchange,token_exchange_error,unregister_node_error,update_consent,update_consent_error,update_email, update_email_error, update_password,update_password_error,update_profile,update_profile_error,update_totp_error,user_disabled_by_permanent_lockout,user_disabled_by_permanent_lockout_error, user_disabled_by_temporary_lockout,user_disabled_by_temporary_lockout_error, user_info_request,user_info_request_error, validate_access_token ,**validate_access_token _error , verify_** email ,**verify_email**, **verify_email_error,verify_profile,verify_profile_error**
spi-events-listener-email-include-events 以逗号分隔的事件列表，这些事件应通过电子邮件发送到用户帐户。 CLI: **--spi-events-listener-email-include-events** Env: **KC_SPI_EVENTS_LISTENER_EMAIL_INCLUDE_EVENTS**	**authreqid_to_token,authreqid_to_token_error,client_delete_error,client_info,client_info_error,client_initiated_account_linking,client_initiated_account_linking_error, client_login,client_login_error,client_register _error**, client_update ,**client_update _error,code_to_token,code_to_token_error,custom_required_action, custom_required_action_error,delete_acc**

	value
	ount,delete_account_error,execute_action_token,execute_action_token_error,execute_actions,execute_actions_error,federated_identity_link,federated_identity_link_error, **grant_consent,grant_consent_error,identity_provider_first_login,identity_provider_first_login_error,identity_provider_link_account,identity_provider_link_account_error,identity_provider_login**, **identity_provider_login_error,identity_provider_post_login,identity_provider_post_login_error,identity_provider_response_error,identity_provider_retrieve_token,identity_provider_retrieve_token_error**, **impersonate,impersonate_error,introspection_token,introspection_token_error,invalid_signature,invalid_signature_error,login,login_error,logout,logout_error**, **oauth2_device_auth**, **oauth2_device_auth_error,oauth2_device_code_to_token,oauth2_device_code_to_token_error,oauth2_device_verify_user_code,oauth2_device_verify_user_code_error,oauth2_extension_grant**, **oauth2_extension_grant_error,permission_token,permission_token_error,pushed_authorization_request _error , refresh_** token ,**refresh_token_error**, register**,**

	value
	register_error, **register_node**,**register_node_error**,**remove_federated_identity**, **remove_federated_identity_error**,**remove_totp**,**remove_totp_error**,**reset_password_error** , **restart_** authentication ,**restart_authentication**, **restart_authentication_error**,**revoke_grant_error**,**send_identity_provider_link**,**send_identity_provider_link_error**,**send_reset_password**,**send_reset_password_error**, **send_verify_email**,**send_verify_email_error**,**token_exchange**,**token_exchange_error**,**unregister_node_error**,**update_consent**,**update_consent_error**,**update_email**, **update_email_error**, **update_password**,**update_password_error**,**update_profile**,**update_profile_error**,**update_totp_error**,**user_disabled_by_permanent_lockout**,**user_disabled_by_permanent_lockout_error**, **user_disabled_by_temporary_lockout**,**user_disabled_by_temporary_lockout_error**, **user_info_request**,**user_info_request_error**, validate_access_token ,**validate_access_token _error** , **verify_** email ,**verify_email**, **verify_email_error**,**verify_profile**,**verify_profile_error**

20.8.2. jboss-logging

	value
spi-events-listener-jboss-logging-error-level 错误消息的日志级别。 CLI: **--spi-events-listener-jboss-logging-error-level** Env: **KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_ERROR_LEVEL**	**debug**,**error**,**fatal**,**info**,**trace**,**warn** （默认）
spi-events-listener-jboss-logging-quotes 要用作值的引号，它应该是一个字符，如 " 或 '。 如果不需要引号，则使用 "none"。 CLI: **--spi-events-listener-jboss-logging- quotes** Env: **KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_QUOTES**	**"**（默认）或 **任意字符串**
spi-events-listener-jboss-logging-sanitize 如果为 true，则日志消息被清理以避免换行符。 如果 false 消息没有清理。 CLI: **--spi-events-listener-jboss-logging-sanitize** Env: **KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_SANITIZE**	**true** （默认）、**false**
spi-events-listener-jboss-logging-success-level 成功消息的日志级别。 CLI: **--spi-events-listener-jboss-logging-success-level** Env: **KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_SUCCESS_LEVEL**	**debug** （默认）、**error**,**fatal**,**info**,**trace**,**warn**

20.9. EXPORT

20.9.1. dir

	value
spi-export-dir-dir 要导出到的目录 CLI: **--spi-export-dir-dir** Env: **KC_SPI_EXPORT_DIR_DIR**	**任何字符串**

	value
spi-export-dir-realm-name 导出的域 CLI: **--spi-export-dir-realm-name** Env: **KC_SPI_EXPORT_DIR_REALM_NAME**	**任何字符串**
spi-export-dir-users-export-strategy 用户导出策略 CLI: **--spi-export-dir-users-export-strategy** Env: **KC_SPI_EXPORT_DIR_USERS_EXPORT_STRATEGY**	**DIFFERENT_FILES**（默认）或 **任何字符串**
spi-export-dir-users-per-file 每个导出的文件的用户 CLI: **--spi-export-dir-users-per-file** Env: **KC_SPI_EXPORT_DIR_USERS_PER_FILE**	**50**（默认）或任何 **int**

20.9.2. single-file

	value
spi-export-single-file-file 要导出到的文件 CLI: **--spi-export-single-file-file** Env: **KC_SPI_EXPORT_SINGLE_FILE_FILE**	**任何字符串**
spi-export-single-file-realm-name 导出的域 CLI: **--spi-export-single-file-realm-name** Env: **KC_SPI_EXPORT_SINGLE_FILE_REALM_NAME**	**任何字符串**

20.10. IMPORT

20.10.1. dir

	value
spi-import-dir-dir 从中导入的目录 CLI: **--spi-import-dir-dir** Env: **KC_SPI_IMPORT_DIR_DIR**	**任何字符串**
spi-import-dir-realm-name 导出的域 CLI: **--spi-import-dir-realm-name** Env: **KC_SPI_IMPORT_DIR_REALM_NAME**	**任何字符串**
spi-import-dir-strategy 导入策略：IGNORE_EXISTING, OVERWRITE_EXISTING CLI: **--spi-import-dir-strategy** Env: **KC_SPI_IMPORT_DIR_STRATEGY**	**任何字符串**

20.10.2. single-file

	value
spi-import-single-file-file 要从中导入的文件 CLI: **--spi-import-single-file-file** Env: **KC_SPI_IMPORT_SINGLE_FILE_FILE**	**任何字符串**
spi-import-single-file-realm-name 导出的域 CLI: **--spi-import-single-file-realm-name** Env: **KC_SPI_IMPORT_SINGLE_FILE_REALM_NAME**	**任何字符串**
spi-import-single-file-strategy 导入策略：IGNORE_EXISTING, OVERWRITE_EXISTING CLI: **--spi-import-single-file-strategy** Env: **KC_SPI_IMPORT_SINGLE_FILE_STRATEGY**	**任何字符串**

20.11. PUBLIC-KEY-STORAGE

20.11.1. Infinispan

	value
spi-public-key-storage-infinispan-max-cache-time 通过所有密钥方法检索密钥时缓存的最大间隔（以秒为单位）。 当检索该条目的所有密钥时，无法检测密钥是否缺失（例如，通过 ID 检索密钥时与情况不同）。在这种情况下，这个选项会强制从时间刷新。默认 24 小时。 CLI: **--spi-public-key-storage-infinispan-max-cache-time** Env: **KC_SPI_PUBLIC_KEY_STORAGE_INFINISPAN_MAX_CACHE_TIME**	**86400** （默认）或任何 **int**
spi-public-key-storage-infinispan-min-time-between-requests 两个请求之间检索新公钥之间的最小间隔（以秒为单位）。 当请求单个密钥且未找到时，服务器将始终尝试下载新的公钥。但是，如果以前的刷新在 10 秒前完成的时间（默认为）时，它会避免下载。此行为用于避免对外部密钥端点的 DoS 攻击。 CLI: **--spi-public-key-storage-infinispan-min-time-between-requests** Env: **KC_SPI_PUBLIC_KEY_STORAGE_INFINISPAN_MIN_TIME_BETWEEN_REQUESTS**	**10** （默认）或任何 **int**

20.12. RESOURCE-ENCODING

20.12.1. gzip

	value
spi-resource-encoding-gzip-excluded-content-types 要从编码中排除的 content-types 列表。 CLI: **--spi-resource-encoding-gzip-excluded-content-types** Env: **KC_SPI_RESOURCE_ENCODING_GZIP_EXCLUDED_CONTENT_TYPES**	**image/png** **image/jpeg** （默认）或**任何字符串**

20.13. STICKY-SESSION-ENCODER

20.13.1. Infinispan

	value
spi-sticky-session-encoder-infinispan-should-attach-route 如果路由应附加到 Cookie，以反映拥有特定会话的节点。 CLI: **--spi-sticky-session-encoder-infinispan-should-attach-route** Env: **KC_SPI_STICKY_SESSION_ENCODER_INFINISPAN_SHOULD_ATTACH_ROUTE**	**true** （默认）、**false**

20.14. TRUSTSTORE

20.14.1. file

	value
spi-truststore-file-file DEPRECATED ：来自读取证书的信任存储的文件路径，以验证 TLS 连接。 CLI: **--spi-truststore-file-file** Env: **KC_SPI_TRUSTSTORE_FILE_FILE**	**任何字符串**
spi-truststore-file-hostname-verification-policy DEPRECATED ：主机名验证策略。 CLI: **--spi-truststore-file-hostname-verification-policy** Env: **KC_SPI_TRUSTSTORE_FILE_HOSTNAME_VERIFICATION_POLICY**	**任何**,**wildcard** （默认）、**strict**
spi-truststore-file-password DEPRECATED ：信任存储密码。 CLI: **--spi-truststore-file-password** Env: **KC_SPI_TRUSTSTORE_FILE_PASSWORD**	**任何字符串**
spi-truststore-file-type DEPRECATED ：信任存储的类型。 如果没有提供，会根据 truststore 文件扩展或平台默认类型检测到类型。 CLI: **--spi-truststore-file-type** Env: **KC_SPI_TRUSTSTORE_FILE_TYPE**	**任何字符串**

20.15. USER-PROFILE

20.15.1. declarative-user-profile

	value
spi-user-profile-declarative-user-profile-admin-read-only-attributes 用于标识应被视为只读字段的正则表达式数组，以便管理员不能更改它们。 CLI: **--spi-user-profile-declarative-user-profile-admin-read-attributes** Env: **KC_SPI_USER_PROFILE_DECLARATIVE_USER_PROFILE_ADMIN_READ_ONLY_ATTRIBUTES**	任何 **MultivaluedString**
spi-user-profile-declarative-user-profile-max-email-local-part-length 设置用户配置文件最大电子邮件本地部分长度 CLI: **--spi-user-profile-declarative-user-profile-max-email-local-length** Env: **KC_SPI_USER_PROFILE_DECLARATIVE_USER_PROFILE_MAX_EMAIL_LOCAL_PART_LENGTH**	**任何字符串**
spi-user-profile-declarative-user-profile-read-only-attributes 用于标识应只读字段的正则表达式数组，以便用户无法更改它们。 CLI: **--spi-user-profile-declarative-user-profile-read-attributes** Env: **KC_SPI_USER_PROFILE_DECLARATIVE_USER_PROFILE_READ_ONLY_ATTRIBUTES**	任何 **MultivaluedString**

20.16. 知名

20.16.1. openid-configuration

	value
spi-well-known-openid-configuration-include-client-scopes 如果应该使用客户端范围来计算支持的范围列表。 CLI: **--spi-well-known-openid-configuration-include-client-scopes** Env: **KC_SPI_WELL_KNOWN_OPENID_CONFIGURATION_INCLUDE_CLIENT_SCOPES**	**true** （默认）、**false**

	value
spi-well-known-openid-configuration-openid-configuration-override 应从中加载元数据的文件路径。 您可以使用绝对文件路径，或者文件位于服务器类路径中，请使用 **classpath:** 前缀来加载 classpath 中的文件。 CLI: **--spi-well-known-openid-configuration-override** Env: **KC_SPI_WELL_KNOWN_OPENID_CONFIGURATION_OPENID_CONFIGURATION_OVERRIDE**	**任何字符串**