Red Hat JBoss Enterprise Application Platform 8.0

保护 JBoss EAP 中的凭证存储

在凭证存储中安全地存储凭证的指南

Last Updated: 2024-02-10

Red Hat JBoss Enterprise Application Platform 8.0 保护 JBoss EAP 中的凭证存储

在凭证存储中安全地存储凭证的指南

法律通告

摘要

在凭证存储中安全地存储凭证的指南。

目录

提供有关 JBOSS EAP 文档的反馈 3

使开源包含更多 4

第 1 章 ELYTRON 中的凭证存储 5
1.1. ELYTRON 提供的凭证存储类型 5
1.2. ELYTRON 中的凭证类型 5
1.3. ELYTRON 凭证存储支持的凭证类型 6
1.4. 使用 JBOSS EAP 管理 CLI 的凭据存储操作 7
1.5. 使用 WILDFLY ELYTRON 工具进行凭证存储操作 19
1.6. 自动更新凭证存储在凭证存储中 27
1.7. 使用 ELYTRON 客户端的凭证存储示例 28
1.8. 创建 FIPS 140-2 兼容凭证存储 29

第 2 章 为 JBOSS EAP 提供一个初始密钥以解锁安全的资源 38
2.1. ELYTRON 中加密的表达式 38
2.2. 在 ELYTRON 中创建加密的表达式 39
2.3. 使用加密表达式来保护 KEYSTORE/CREDENTIAL-STORE 41

第 3 章 参考 42
3.1. AGGREGATE-PROVIDERS 属性 42
3.2. CREDENTIAL-STORE 属性 42
3.3. CREDENTIAL-STORE 实现属性 43
3.4. EXPRESSION=ENCRYPTION 属性 43
3.5. PROVIDER-LOADER 属性 44
3.6. SECRET-KEY-CREDENTIAL-STORE 属性 44

提供有关 JBOSS EAP 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

流程

1. 单击以下链接 **以创建 ticket**。
2. 在 **Summary** 中输入问题的简短描述。
3. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
4. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 ELYTRON 中的凭证存储

1.1. ELYTRON 提供的凭证存储类型

Elytron 提供两个默认凭证类型，您可以使用它来保存您的凭证：KeyStoreCredentialStore 和 PropertiesCredentialStore。您可以使用 JBoss EAP 管理 CLI 来管理凭据存储，或者您可以使用 WildFly Elytron 工具脱机管理它们。除了两种默认存储类型外，您还可以创建、使用和管理您自己的自定义凭据存储。

1.1.1. Elytron 中的 KeyStoreCredentialStore/credential-store

您可以将所有 Elytron 凭证类型存储在 KeyStoreStore 中。**elytron** 子系统中的 KeyStore 的资源名称为 **credentials-store**。KeyStoreCredentialStore 使用 Java Development Kit(JDK)中的 KeyStore 实现提供的机制来保护您的凭证。

在管理 CLI 中访问 KeyStoreCredentialStore，如下所示：

```
/subsystem=elytron/credential-store
```

其他资源

- 为独立服务器创建一个 **credential-store**
- 为受管域创建一个 **credential-store**
- 使用 WildFly Elytron 工具创建一个 **credential-store**
- 使用 Bouncy Castle 供应程序创建一个 **credential-store**
- **credential-store** 属性

1.1.2. Elytron 中的 PropertiesCredentialStore/secret-key-credential-store

为了正确启动，JBoss EAP 需要初始密钥来解锁某些安全资源。使用 PropertiesCredentialStore 提供这个初始 secret 密钥来解锁这些必要的服务器资源。您还可以使用 PropertiesCredentialStore 存储 SecretKeyCredential，它支持存储高级加密标准(AES)secret 密钥。使用文件系统权限来限制对凭据存储的访问。理想情况下，您应该只向应用服务器授予访问权限，以限制对此凭据存储的访问。

PropertiesCredentialStore 的 **elytron** 子系统中的资源名称是 **secret-key-credential-store**，您可以在管理 CLI 中访问它，如下所示：

```
/subsystem=elytron/secret-key-credential-store
```

其他资源

- 为独立服务器创建一个 **secret-key-credential-store**
- 使用 WildFly Elytron 工具创建一个 **secret-key-credential-store**
- **secret-key-credential-store** 属性

1.2. ELYTRON 中的凭证类型

Elytron 提供了以下三种凭证类型来满足您的各种安全需求，您可将这些凭据存储在 Elytron 的凭据存储中。

PasswordCredential

使用这个凭证类型，您可以安全地存储纯文本或未加密的密码。对于需要密码的 JBoss EAP 资源，请使用指向 PasswordCredential 而不是纯文本密码的引用来维护密码的保密。

连接到数据库的示例

```
data-source add ... --user-name=db_user --password=StrongPassword
```

在这个示例数据库连接命令中，您可以看到密码： **StrongPassword**。这意味着其他人也可以在服务器配置文件中看到它。

使用 PasswordCredential 连接到数据库的示例

```
data-source add ... --user-name=db_user --credential-reference=
{store=exampleKeyStoreCredentialStore, alias=passwordCredentialAlias}
```

当使用凭证引用而不是密码连接到数据库时，其他人只能看到配置文件中的凭证引用，而不是您的密码

KeyPairCredential

您可以将 Secure Shell(SSH)和公钥密码标准(PKCS)密钥对用作 KeyPairCredential。密钥对包括共享的公钥和只有给定用户知道的私钥。
您只能使用 WildFly Elytron 工具管理 KeyPairCredential。

SecretKeyCredential

SecretKeyCredential 是高级加密标准(AES)密钥，可用于在 Elytron 中创建加密的表达式。

其他资源

- Elytron 提供的凭证存储
- 凭证存储支持的凭证类型

1.3. ELYTRON 凭证存储支持的凭证类型

下表描述了哪个凭证存储支持哪个凭证类型：

凭证类型	KeyStoreCredentialStore/credential-store	PropertiesCredentialStore/secret-key-credential-store
PasswordCredential	是	否
KeyPairCredential	是	否
SecretKeyCredential	是	是

其他资源

- Elytron 中的凭证类型
- Elytron 提供的凭证存储

1.4. 使用 JBOSS EAP 管理 CLI 的凭据存储操作

要在运行的 JBoss EAP 服务器中管理 JBoss EAP 凭据，请使用提供的管理 CLI 操作。您可以使用 JBoss EAP 管理 CLI 管理 **PasswordCredential** 和 **SecretKeyCredential**。

注意

您只能对可修改的凭证存储进行这些操作。所有凭据存储类型默认为可修改。

1.4.1. 为独立服务器创建一个 credential-store

为在文件系统上的任意目录中作为独立服务器运行的 JBoss EAP 创建一个 **credential-store**。为安全起见，只有有限用户才可以访问包含存储的目录。

先决条件

- 您对在其下运行 JBoss EAP 的用户帐户至少提供了对包含 KeyStoreCredentialStore 目录的读/写权限。

注意

credential-store 和 **secret-key-credential-store** 不能同名，因为它们实现相同的 Elytron 功能：**org.wildfly.security.credential-store**。

流程

- 使用以下管理 CLI 命令，创建一个 KeyStoreCredentialStore：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:add(path="<path_to_store_file>", relative-to=<base_path_to_store_file>, credential-reference={clear-text=<store_password>}, create=true)
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:add(path="exampleKeyStoreCredentialStore.jceks", relative-to=jboss.server.data.dir, credential-reference={clear-text=password}, create=true)
  {"outcome" => "success"}
  ```

其他资源

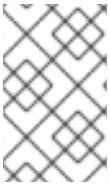
- Elytron 中的 KeyStoreCredentialStore/**credential-store**
- 使用 JBoss EAP 管理 CLI 的凭据存储操作
- **credential-store** 属性

1.4.2. 为受管域创建一个 credential-store

您可以在受管域中创建一个 **credential-store**，但必须首先使用 WildFly Elytron 工具来准备您的 KeyStoreCredentialStore。如果在单个受管域中有多个主机控制器，请选择以下选项之一：

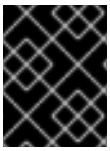
- 在每个主机控制器中创建一个 **credential-store**，并向每一个 **credential-store** 添加凭证。
- 将填充的 **credential-store** 从一个主机控制器拷贝到所有其他主机控制器。
- 将 **credential-store** 文件保存在网络文件系统(NFS)中，然后对您创建的所有 **credential-store** 资源使用该文件。

或者，您可以在主机控制器上创建一个带有凭证的 **credential-store** 文件，而无需使用 WildFly Elytron 工具。



注意

您不必在每台服务器上定义一个 **credential-store** 资源，因为同一配置文件上的每个服务器都包含您的 **credential-store** 文件。您可以在服务器的 **data** 目录 **relative-to=jboss.server.data.dir** 中找到 **credential-store** 文件 。



重要

credential-store 和 **secret-key-credential-store** 不能同名，因为它们实现相同的 Elytron 功能：**org.wildfly.security.credential-store**。

以下流程描述了如何使用 NFS 来向所有主机控制器提供 **credential-store** 文件。

流程

1. 使用 WildFly Elytron 工具来创建一个 **credential-store** 存储文件。有关此内容的更多信息，请参阅 WildFly Elytron 工具 **credential-store** 操作。
2. 分发存储文件。例如，使用 **scp** 命令将它分配给每个主机控制器，或者将其存储在 NFS 中，将其用于所有 **credential-store** 资源。

注意

为保持一致性，对于多个资源和主机控制器使用的且存储在 NFS 中的 **credential-store** 文件，您必须在只读模式下使用 **credential-store**。另外，请确保您为 **credential-store** 文件提供了一个绝对路径。

语法

```
/profile=<profile_name>/subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<absolute_path_to_store_keystore>,credential-reference={clear-text="<store_password>"},create=false,modifiable=false)
```

示例

```
/profile=full-ha/subsystem=elytron/credential-store=exampleCredentialStoreDomain:add(path=/usr/local/etc/example-cred-store.cs,credential-reference={clear-text="password"},create=false,modifiable=false)
```

3. *可选：* 如果您需要在配置集中定义 **credential-store** 资源，请使用存储文件来创建资源。

语法

```
/profile=<profile_name>/subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<path_to_store_file>,credential-reference={clear-text="<store_password>"})
```

Example

```
/profile=full-ha/subsystem=elytron/credential-store=exampleCredentialStoreHA:add(path=/usr/local/etc/example-cred-store-ha.cs,credential-reference={clear-text="password"})
```

4. *可选：* 为主机控制器创建 **credential-store** 资源。

语法

```
/host=<host_controller_name>/subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<path_to_store_file>,credential-reference={clear-text="<store_password>"})
```

Example

```
/host=master/subsystem=elytron/credential-store=exampleCredentialStoreHost:add(path=/usr/local/etc/example-cred-store-host.cs,credential-reference={clear-text="password"})
```

其他资源

- Elytron 中的 KeyStoreCredentialStore/ **credential-store**

- 使用 WlidFly Elytron 工具进行凭证存储操作
- **credential-store** 属性

1.4.3. 为独立服务器创建一个 secret-key-credential-store

使用管理 CLI 创建一个 **secret-key-credential-store**。当您创建 **secret-key-credential-store** 时，JBoss EAP 默认会生成一个 secret 密钥。生成的密钥的名称是 **key**，其大小为 256 位。

先决条件

- JBoss EAP 正在运行。
- 您为在其下运行 JBoss EAP 的用户帐户至少提供了对包含 **secret-key-credential-store** 的目录的读/写权限。

流程

- 使用以下命令，使用管理 CLI 来创建一个 **secret-key-credential-store** ：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:add(path="<path_to_the_credential_store>", relative-to=<path_to_store_file>)
  ```

  Example

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:add(path=examplePropertiesCredentialStore.cs, relative-to=jboss.server.config.dir)
  {"outcome" => "success"}
  ```

1.4.4. 向 credential-store 添加一个 PasswordCredential

为那些需要将密码作为 PasswordCredential 的资源，向 **credential-store** 中添加一个明文密码，以便在配置文件中隐藏该密码。然后，您可以引用该存储的凭证来访问这些资源，而不用暴露您的密码。

先决条件

- 您已创建了一个 **credential-store**。
  有关创建 **credential-store** 的详情，请参考 为独立服务器创建一个 **credential-store** 。

流程

- 将新的 PasswordCredential 添加到 **credential-store** 中：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:add-alias(alias=<alias>, secret-value=<secret-value>)
  ```

示例

```
/subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:add-
alias(alias=passwordCredentialAlias, secret-value=StrongPassword)
{"outcome" => "success"}
```

验证

- 发出以下命令来验证 PasswordCredential 是否已添加到 **credential-store** 中：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => ["passwordcredentialalias"]
  }
  ```

其他资源

- Elytron 中的 KeyStoreCredentialStore/ **credential-store**
- **credential-store** 属性

1.4.5. 在 credential-store 中生成一个 SecretKeyCredential

在 **credential-store** 中生成一个 SecretKeyCredential。默认情况下，Elytron 创建 256 位密钥。如果需要不同的大小，您可以在 **key-size** 属性中指定 128 位或 192 位密钥。

先决条件

- 您已创建了一个 **credential-store**。
  有关创建 **credential-store** 的详情，请参考 为独立服务器创建一个 **credential-store** 。

流程

- 使用以下管理 CLI 命令，在 **credential-store** 中生成一个 SecretKeyCredential：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:generate-secret-
  key(alias=<alias>, key-size=<128_or_192>)
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:generate-secret-
  key(alias=secretKeyCredentialAlias)
  ```

验证

- 发出以下命令以验证您的 SecretKeyCredential 是否存储在 **credential-store** 中：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => [
          "secretkeycredentialalias"
      ]
  }
  ```

其他资源

- Elytron 中的 KeyStoreCredentialStore/**credential-store**
- **credential-store** 属性

1.4.6. 在 secret-key-credential-store 中生成一个 SecretKeyCredential

在 **secret-key-credential-store** 中生成一个 SecretKeyCredential。默认情况下，Elytron 创建 256 位密钥。如果需要不同的大小，您可以在 **key-size** 属性中指定 128 位或 192 位密钥。

当您生成 SecretKeyCredential 时，Elytron 生成一个新的随机 secret 密钥，并将其保存为 SecretKeyCredential。您可以对 **secret-key-credential-store** 使用 export 操作来查看凭证的内容。

重要

确保您创建了一个 **secret-key-credential-store**，SecretKeyCredential 的备份或两者的备份，因为 JBoss EAP 无法解密或检索丢失的 Elytron 凭证。

您可以对 **secret-key-credential-store** 使用 **export** 操作来获取 SecretKeyCredential 的值。然后您可以将这个值保存为备份。

先决条件

- 您已创建了一个 **secret-key-credential-store**。
  有关创建 **secret-key-credential-store** 的详情，请参考 为独立服务器创建一个 **secret-key-credential-store** 。

流程

- 使用以下管理 CLI 命令在 **secret-key-credential-store** 中生成一个 SecretKeyCredential：

  语法

```
/subsystem=elytron/secret-key-credential-store=<name_of_the_properties_credential_store>:generate-secret-key(alias=<alias>, key-size=<128_or_192>)
```

示例

```
/subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:generate-secret-key(alias=secretKeyCredentialAlias)
{"outcome" => "success"}
```

验证

- 发出以下命令以验证 Elytron 创建了 SecretKeyCredential:

  语法

```
/subsystem=elytron/secret-key-credential-store=<name_of_the_properties_credential_store>:read-aliases()
```

  示例

```
/subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:read-aliases()
{
    "outcome" => "success",
    "result" => [
        "secretkeycredentialalias",
        "key"
    ]
}
```

其他资源

- Elytron 中的 PropertiesCredentialStore/**secret-key-credential-store**
- **secret-key-credential-store** 属性

1.4.7. 将 SecretKeyCredential 导入到 secret-key-credential-store 中

您可以将在 **secret-key-credential-store** 外创建的 SecretKeyCredential 导入到 Elytron **secret-key-credential-store** 中。假设您从另一个凭证存储中导出了一个 SecretKeyCredential，例如，您可以将 **credential-store** 导入到 **secret-key-credential-store** 中。

先决条件

- 您已创建了一个 **secret-key-credential-store**。
  有关创建 **secret-key-credential-store** 的详情，请参考 为独立服务器创建一个 **secret-key-credential-store** 。
- 您已导出了一个 SecretKeyCredential。
  有关导出 SecretKeyCredential 的详情，请参考从 **secret-key-credential-store**中导出 SecretKeyCredential。

流程

1. 使用以下命令在管理 CLI 中禁用命令缓存：

重要

如果不禁用缓存，则可以访问管理 CLI 历史记录文件的任何人都可以看到 secret 密钥。

```
history --disable
```

2. 使用以下管理 CLI 命令导入 secret 密钥：

语法

```
/subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:import-secret-key(alias=<alias>, key="<secret_key>")
```

示例

```
/subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:import-secret-key(alias=imported,
key="RUxZAUs+Y1CzEPw0g2AHHOZ+oTKhT9osSabWQtoxR+O+42o11g==")
```

3. 使用以下管理 CLI 命令重新启用命令缓存：

```
history --enable
```

其他资源

- Elytron 中的 PropertiesCredentialStore/**secret-key-credential-store**
- **secret-key-credential-store** 属性

1.4.8. 列出**credential-store** 中的凭证

要查看 **credential-store** 中存储的所有凭证，您可以使用管理 CLI 列出它们。

流程

- 使用以下管理 CLI 命令列出存储在 **credential-store** 中的凭证：

语法

```
/subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
```

示例

```
{
    "outcome" => "success",
    "result" => [
```

```
        "passwordcredentialalias",
        "secretkeycredentialalias"
    ]
}
```

其他资源

- Elytron 中的 KeyStoreCredentialStore/**credential-store**
- **credential-store** 属性

1.4.9. 从 credential-store 中导出 SecretKeyCredential

您可以从 **credential-store** 中导出现一个现有的 SecretKeyCredential，以使用 SecretKeyCredential 或创建一个 SecretKeyCredential 的备份。

先决条件

- 您已在 **credentials-store** 中生成了一个 SecretKeyCredential。
  有关在 **credential-store** 中生成 SecretKeyCredential 的信息，请参阅在 **credential-store**中生成 SecretKeyCredential。

流程

- 使用以下管理 CLI 命令，从 **credential-store** 中导出 SecretKeyCredential：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:export-secret-key(alias=<alias>)
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:export-secret-key(alias=secretKeyCredentialAlias)
  {
      "outcome" => "success",
      "result" => {"key" =>
  "RUxZAUui+8JkoDCE6mFyA3cCIbSAZaXq5wgYejj1scYgdDqWiw=="}
  }
  ```

其他资源

- Elytron 中的 KeyStoreCredentialStore/**credential-store**
- **credential-store** 属性

1.4.10. 从 secret-key-credential-store 中导出 SecretKeyCredential

您可以从 **secret-key-credential-store** 中导出一个现有的 SecretKeyCredential 以使用 SecretKeyCredential，或创建一个 SecretKeyCredential 的备份。

先决条件

- 您已在 **secret-key-credential-store** 中生成了一个 SecretKeyCredential，或者已将其导入到其中。
  有关在 **secret-key-credential-store** 中生成一个 SecretKeyCredential 的信息，请在 **secret-key-credential-store**中生成一个 Secret KeyCredential。

  有关将 SecretKeyCredential 导入到 **secret-key-credential-store** 中的信息，请参阅将 SecretKeyCredential 导入到 **secret-key-credential-store** 中。

流程

- 使用以下管理 CLI 命令从 **secret-key-credential-store** 中导出一个 SecretKeyCredential：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:export-secret-key(alias=<alias>)
  ```

  示例

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:export-secret-key(alias=secretkeycredentialalias)
  {
      "outcome" => "success",
      "result" => {"key" => "RUxZAUtxXcYvz0aukZu+odOynIr0ByLhC72iwzlJsi+ZPmONgA=="}
  }
  ```

其他资源

- Elytron 中的 PropertiesCredentialStore/**secret-key-credential-store**
- **secret-key-credential-store** 属性

1.4.11. 从 credential-store 中删除一个凭证

您可以在 **credential-store** 中存储每个凭证类型，但默认情况下，当您删除凭证时，Elytron 会假定它是 PasswordCredential。如果要删除其他凭证类型，请在 **entry-type** 属性中指定它。

流程

- 使用以下管理 CLI 命令，从 **credential-store** 中删除一个凭证：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:remove-alias(alias=<alias>, entry-type=<credential_type>)
  ```

  删除 PasswordCredential 的示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:remove-alias(alias=passwordCredentialAlias)
  {
  ```

```
    "outcome" => "success",
    "response-headers" => {"warnings" => [{
        "warning" => "Update dependent resources as alias 'passwordCredentialAlias' does not
exist anymore",
        "level" => "WARNING",
        "operation" => {
            "address" => [
                ("subsystem" => "elytron"),
                ("credential-store" => "exampleKeyStoreCredentialStore")
            ],
            "operation" => "remove-alias"
        }
    }]}
}
```

删除 SecretKeyCredential 的示例

```
/subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:remove-
alias(alias=secretKeyCredentialAlias, entry-type=SecretKeyCredential)
{
    "outcome" => "success",
    "response-headers" => {"warnings" => [{
        "warning" => "Update dependent resources as alias 'secretKeyCredentialAl
ias' does not exist anymore",
        "level" => "WARNING",
        "operation" => {
            "address" => [
                ("subsystem" => "elytron"),
                ("credential-store" => "exampleKeyStoreCredentialStore")
            ],
            "operation" => "remove-alias"
        }
    }]}
}
```

验证

- 发出以下命令来验证 Elytron 是否删除了凭证：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => []
  }
  ```

  您删除的凭证不会被列出。

其他资源

- Elytron 中的 KeyStoreCredentialStore/**credential-store**
- **credential-store** 属性

1.4.12. 从 secret-key-credential-store 中删除一个凭证

您只能将 SecretKeyCredential 类型存储在 **secret-key-credential-store** 中。这意味着，当您从 **secret-key-credential-store** 中删除一个凭证时，您不必指定一个 **entry-type**。

流程

- 使用以下命令，从 **secret-key-credential-store** 中删除一个 SecretKeyCredential：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:remove-alias(alias=<alias>)
  ```

  示例

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:remove-alias(alias=secretKeyCredentialAlias)
  {
      "outcome" => "success",
      "response-headers" => {"warnings" => [{
          "warning" => "Update dependent resources as alias 'secretKeyCredentialAlias' does not exist anymore",
          "level" => "WARNING",
          "operation" => {
              "address" => [
                  ("subsystem" => "elytron"),
                  ("secret-key-credential-store" => "examplePropertiesCredentialStore")
              ],
              "operation" => "remove-alias"
          }
      }]}
  }
  ```

验证

- 发出以下命令来验证 Elytron 是否删除了凭证：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:read-aliases()
  ```

```
{
    "outcome" => "success",
    "result" => []
}
```

您删除的凭证不会被列出。

其他资源

- Elytron 中的 PropertiesCredentialStore/**secret-key-credential-store**
- **secret-key-credential-store** 属性

1.5. 使用 WILDFLY ELYTRON 工具进行凭证存储操作

您可以使用 WildFly Elytron 工具对凭证存储执行各种操作。

1.5.1. 使用 WildFly Elytron 工具创建一个 credential-store

在 Elytron 中，您可以在保存所有凭证类型的地方离线创建一个 **credential-store**。

流程

- 使用以下命令，使用 WildFly Elytron 工具创建一个 **credential-store** ：

  语法

  ```
  $ EAP_HOME/bin/elytron-tool.sh credential-store --create --location "<path_to_store_file>" --password <store_password>
  ```

  示例

  ```
  $ EAP_HOME/bin/elytron-tool.sh credential-store --create --location "../cred_stores/example-credential-store.jceks" --password storePassword
  Credential Store has been successfully created
  ```

  如果您不想将存储密码包含在命令中，请省略该参数，然后在提示符下手动输入密码。您还可以使用由 WildFly Elytron 工具生成的已屏蔽密码。有关生成屏蔽密码的详情，请参考 使用 WildFly Elytron 工具生成屏蔽的加密字符串。

其他资源

- Elytron 中的 KeyStoreCredentialStore/ **credential-store**
- 使用 WildFly Elytron 工具生成屏蔽的加密字符串
- WildFly Elytron 工具 **credential-store** 操作

1.5.2. 使用 Bouncy Castle 提供程序创建一个 credential-store

使用 Bouncy Castle 提供程序创建一个 **credential-store**。

先决条件

- 确保您的环境已配置为使用 Bouncy Castle。

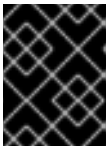
注意

credential-store 和 **secret-key-credential-store** 不能同名，因为它们实现相同的 Elytron 功能：**org.wildfly.security.credential-store**。

流程

1. 定义 Bouncy Castle FIPS 密钥存储(**BCFKS**)密钥存储。FIPS 代表联邦信息处理标准。如果您已有一个，请转到下一步。

```
$ keytool -genkeypair -alias <key_pair_alias> -keyalg <key_algorithm> -keysize <key_size> -storepass <key_pair_and_keystore_password> -keystore <path_to_keystore> -storetype BCFKS -keypass <key_pair_and_keystore_password>
```



重要

确保密钥存储 **keypass** 和 **storepass** 属性相同。如果不是这样，**elytron** 子系统中的 **BCFKS** keystore 将无法定义它们。

2. 为 **credential-store** 生成一个 secret 密钥。

```
$ keytool -genseckey -alias <key_alias> -keyalg <key_algorithm> -keysize <key_size> -keystore <path_to_keystore> -storetype BCFKS -storepass <key_and_keystore_password> -keypass <key_and_keystore_password>
```

3. 使用以下命令，使用 WildFly Elytron 工具定义 **credential-store** ：

```
$ EAP_HOME/bin/elytron-tool.sh credential-store -c -a <alias> -x <alias_password> -p <key_and_keystore_password> -l <path_to_keystore> -u "keyStoreType=BCFKS;external=true;keyAlias=<key_alias>;externalPath=<path_to_credential_store>"
```

其他资源

- Elytron 中的 KeyStoreCredentialStore/ **credential-store**
- WildFly Elytron 工具 **credential-store** 操作

1.5.3. 使用 WildFly Elytron 工具创建一个 secret-key-credential-store

在 Elytron 中，您可以在保存 SecretKeyCredential 实例的地方离线创建一个 **secret-key-credential-store** 。

流程

- 使用以下命令，使用 WildFly Elytron 工具创建 PropertiesCredentialStore：

  语法

  -

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --create --location "<path_to_store_file>" --type PropertiesCredentialStore
```

示例

```
$ bin/elytron-tool.sh credential-store --create --location=standalone/configuration/properties-credential-store.cs --type PropertiesCredentialStore
Credential Store has been successfully created
```

其他资源

- Elytron 中的 PropertiesCredentialStore/**secret-key-credential-store**
- WildFly Elytron 工具 **secret-key-credential-store** 操作

1.5.4. WildFly Elytron 工具 credential-store 操作

您可以使用 WildFly Elytron 工具执行各种 **credential-store** 任务，其中包括：

添加一个 PasswordCredential

您可以使用以下 WildFly Elytron 工具命令向 **credential-store** 添一个 PasswordCredential ：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password <store_password> --add <alias> --secret <sensitive_string>
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-store.jceks" --password storePassword --add examplePasswordCredential --secret speci@l_db_pa$$_01
Alias "examplePasswordCredential" has been successfully stored
```

如果您不想将 secret 放在命令中，请省略该参数，然后在提示时手动输入 secret。

生成一个 SecretKeyCredential

您可以使用以下 WildFly Elytron 工具命令向 **credential-store** 添加一个 SecretKeyCredential ：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location=<path_to_the_credential_store> --password <store_password>
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location "../cred_stores/example-credential-store.jceks" --password storePassword
Alias "example" has been successfully stored
```

如果您不想将 secret 放在命令中，请省略该参数，然后在提示时手动输入 secret。

默认情况下，当您在 JBoss EAP 中创建 SecretKeyCredential 时，您可以创建一个 256 位 secret 密钥。如果要更改大小，您可以指定 **--size=128** 或 **--size=192** 来分别创建 128 位或 192 位密钥。

导入 SecretKeyCredential

您可以使用以下 WildFLy Elytron 工具命令导入 SecretKeyCredential：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --
location=<path_to_credential_store> --password=<store_password>
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --
location=../cred_stores/example-credential-store.jceks --password=storePassword
```

输入您要导入的 secret 密钥。

列出所有的凭证

您可以使用以下 WildFly Elytron 工具命令列出 **credential-store** 中的凭证：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password
<store_password> --aliases
```

例如：

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-
store.jceks" --password storePassword --aliases
Credential store contains following aliases: examplepasswordcredential example
```

检查是否存在别名

使用以下命令检查凭证存储中是否存在别名：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password
<store_password> --exists <alias>
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-
store.jceks" --password storePassword --exists examplepasswordcredential
Alias "examplepasswordcredential" exists
```

导出 SecretKeyCredential

您可以使用以下命令从 **credential-store** 中导出一个 SecretKeyCredential：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=<alias> --
location=<path_to_credential_store> --password=storePassword
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=example --
location=../cred_stores/example-credential-store.jceks --password=storePassword
Exported SecretKey for alias
example=RUxZAUtBiAnoLP1CA+i6DtcbkZHfybBJxPeS9mlVOmEYwjjmEA==
```

删除一个凭证

您可以使用以下命令从凭证存储中删除一个凭证：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password
<store_password> --remove <alias>
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-
store.jceks" --password storePassword --remove examplepasswordcredential
Alias "examplepasswordcredential" has been successfully removed
```

其他资源

- Elytron 中的 KeyStoreCredentialStore/ **credential-store**
- Elytron 中的凭证类型

1.5.5. WildFly Elytron 工具 secret-key-credential-store 操作

您可以使用 WildFly Elytron 工具为 SecretKeyCredential 执行以下 **secret-key-credential-store** 操作：

生成一个 SecretKeyCredential

您可以使用以下 WildFly Elytron 工具命令在 **secret-key-credential-store** 中生成一个 **SecteKeyCredential** ：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location
"<path_to_the_credential_store>" --type PropertiesCredentialStore
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location
"standalone/configuration/properties-credential-store.cs" --type PropertiesCredentialStore
Alias "example" has been successfully stored
```

导入 SecretKeyCredential

您可以使用以下 WildFLy Elytron 工具命令导入 SecretKeyCredential：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --
location=<path_to_credential_store> --type PropertiesCredentialStore
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --location
"standalone/configuration/properties-credential-store.cs" --type PropertiesCredentialStore
```

列出所有的凭证

您可以使用以下 WildFly Elytron 工具列出 **secret-key-credential-store** 中的凭证：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>"  --aliases --type
PropertiesCredentialStore
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store  --location "standalone/configuration/properties-
credential-store.cs" --aliases --type PropertiesCredentialStore
Credential store contains following aliases: example
```

导出 SecretKeyCredential

您可以使用以下命令从 **secret-key-credential-store** 中导出一个 SecretKeyCredential：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=<alias> --location
"<path_to_credential_store>"  --type PropertiesCredentialStore
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=example --location
"standalone/configuration/properties-credential-store.cs" --type PropertiesCredentialStore
Exported SecretKey for alias
example=RUxZAUt1EZM7PsYRgMGypkGirSel+5Eix4aSgwop6jfxGYUQaQ==
```

删除一个凭证

您可以使用以下命令从凭证存储中删除一个凭证：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --remove
<alias> --type PropertiesCredentialStore
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "standalone/configuration/properties-credential-store.cs"  --remove example --type PropertiesCredentialStore
Alias "example" has been successfully removed
```

其他资源

- Elytron 中的 PropertiesCredentialStore/**secret-key-credential-store**
- Elytron 中的凭证类型

1.5.6. 将使用 WildFly Elytron 工具创建的 credential-store 添加到 JBoss EAP 服务器

在使用 WildFly Elytron 工具创建了 **credential-store** 后，您可以将其添加到正在运行的 JBoss EAP 服务器中。

先决条件

- 您已使用 WildFly Elytron 工具创建了凭据存储。
  如需更多信息，请参阅 使用 WildFly Elytron 工具创建一个 **credential-store** 。

流程

- 使用以下管理 CLI 命令，将凭证存储添加到正在运行的 JBoss EAP 服务器中：

  语法

  ```
  /subsystem=elytron/credential-store=<store_name>:add(location="<path_to_store_file>",credential-reference={clear-text=<store_password>})
  ```

  Example

  ```
  /subsystem=elytron/credential-store=my_store:add(location="../cred_stores/example-credential-store.jceks",credential-reference={clear-text=storePassword})
  ```

将凭证存储添加到 JBoss EAP 配置后，您可以使用 **credential-reference** 属性引用存储在凭证存储中的密码或敏感的字符串。

如需更多信息，请使用 ***EAP_HOME*/bin/elytron-tool.sh credential-store --help** 命令获得可用选项的详细列表。

其他资源

- **credential-store** 属性

1.5.7. WildFly Elytron 工具密钥对管理操作

您可以使用以下参数操作 **elytron-tool.sh**，来处理凭据存储，例如生成一个可在凭证存储中的一个别名下存储的新密钥对。

生成一个密钥对

使用 **generate-key-pair** 命令创建一个密钥对。然后您可以将密钥对存储在凭证存储的一个别名下。以下示例显示了如何创建一个 *RSA* 密钥对，给其分配的大小为 *3072* 位，存储在为凭证存储指定的位置上。提供给密钥对的别名是 **example**。

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location=<path_to_store_file> --generate-key-pair example --algorithm RSA --size 3072
```

导入一个密钥对

使用 **import-key-pair** 命令将现有的 SSH 密钥对导入到具有指定别名的凭证存储中。以下示例导入一个密钥对，其别名 *example* 来自 */home/user/.ssh/id_rsa* 文件，该文件包含采用 OpenSSH 格式的私钥：

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-key-pair example --private-key-location /home/user/.ssh/id_rsa --location=<path_to_store_file>
```

导出一个密钥对

使用 **export-key-pair-public-key** 命令显示密钥对的公钥。公钥具有 OpenSSH 格式的指定别名。以下示例显示了别名 *example* 的公钥：

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location=<path_to_store_file> --export-key-pair-public-key example

Credential store password:
Confirm credential store password:
ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBMfncZuHmR7uglb0M96ieArRFtp42xPn9+ugukbY8dyjOXoi
cZrYRyy9+X68fylEWBMzyg+nhjWkxJlJ2M2LAGY=
```

注意

在发出 **export-key-pair-public-key** 命令后，会提示您输入凭证存储密码短语。如果不存在密码短语，请将提示留空。

1.5.8. 在 Elytron 配置文件中使用存储的密钥对示例

密钥对由两个独立的，但匹配的加密密钥组成：公钥和私钥。在 **elytron** 配置文件中引用密钥对之前，您需要将密钥对存储在凭证存储中。然后，您可以提供对 Git 的访问权限，来管理您的独立服务器配置数据。

以下示例引用了 **elytron** 配置文件的 **<credential-stores>** 元素中的凭证存储及其属性。**<credential>** 元素引用凭证存储和别名，用于存储密钥对。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.6">

    <credential-stores>
      <credential-store name="${credential_store_name}">
        <protection-parameter-credentials>
```

```
            <clear-password password="${credential_store_password}"/>
          </protection-parameter-credentials>
          <attributes>
            <attribute name="path" value="${path_to_credential_store}"/>
          </attributes>
        </credential-store>
      </credential-stores>

      <authentication-rules>
        <rule use-configuration="${configuration_file_name}"/>
      </authentication-rules>

      <authentication-configurations>
        <configuration name="${configuration_file_name}">
          <credentials>
            <credential-store-reference store="${credential_store_name}" alias="${alias_of_key_pair}"/>
          </credentials>
        </configuration>
      </authentication-configurations>

    </authentication-client>
  </configuration>
```

配置 **elytron** 配置文件后，密钥对可用于 SSH 身份验证。

其他资源

- WildFly Elytron 工具密钥对管理操作

1.5.9. 使用 WildFly Elytron 工具生成屏蔽的加密字符串

您可以使用 WildFly Elytron 工具来生成用于凭证存储的加密的字符串，而不是使用明文密码。

流程

- 要生成屏蔽的字符串，请使用以下命令，并为 salt 和iteration 数提供值：

  ```
  $ EAP_HOME/bin/elytron-tool.sh mask --salt <salt> --iteration <iteration_count> --secret <password>
  ```

  例如：

  ```
  $ EAP_HOME/bin/elytron-tool.sh mask --salt 12345678 --iteration 123 --secret supersecretstorepassword

  MASK-8VzWsSNwBaR676g8ujiIDdFKwSjOBHCHgnKf17nun3v;12345678;123
  ```

  如果您不想在命令中提供 secret，可以省略该参数，系统将提示您使用标准输入输入 secret。

如需更多信息，请使用 ***EAP_HOME*/bin/elytron-tool.sh mask --help** 命令来获取可用选项的详细列表。

1.6. 自动更新凭证存储在凭证存储中

如果您有凭证存储，则在从凭证引用引用它们之前，您不必添加凭证或更新现有的凭证。Elytron 自动化了此过程。配置凭证引用时，请同时指定 **store** 和 **clear-text** 属性。Elytron 会在 **store** 属性指定的凭证存储中自动添加或更新凭证。另外，您还可以指定 **alias** 属性。

Elytron 更新凭证存储，如下所示：

- 如果您指定了一个别名：
  - 如果存在别名的条目，现有的凭证会被指定的明文密码替换。
  - 如果别名的条目不存在，则新条目和明文密码将被添加到具有指定别名的凭证存储。
- 如果您没有指定别名，Elytron 会生成一个别名，并将新条目和指定的明文密码添加到具有生成的别名的凭证存储。

当凭证存储被更新时，**clear-text** 属性会从管理模型中删除。

以下示例演示了如何创建用来指定 **store**, **clear-text**, 和 **alias** 属性的凭证引用：

```
/subsystem=elytron/key-store=exampleKS:add(relative-to=jboss.server.config.dir,
path=example.keystore, type=JCEKS, credential-reference=
{store=exampleKeyStoreCredentialStore, alias=myNewAlias, clear-text=myNewPassword})
{
    "outcome" => "success",
    "result" => {"credential-store-update" => {
        "status" => "new-entry-added",
        "new-alias" => "myNewAlias"
    }}
}
```

您可以使用以下命令更新添加到之前定义的凭证存储中的 **myNewAlias** 条目的凭证：

```
/subsystem=elytron/key-store=exampleKS:write-attribute(name=credential-reference.clear-
text,value=myUpdatedPassword)
{
    "outcome" => "success",
    "result" => {"credential-store-update" => {"status" => "existing-entry-updated"}},
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

注意

如果包含 **credentials-reference** 参数的操作失败，则不会发生自动凭证存储更新。

由 **credentials-reference** 属性指定的凭据存储不会改变。

1.7. 使用 ELYTRON 客户端的凭证存储示例

连接到 JBoss EAP（如 Jakarta Enterprise Bean）的客户端可以使用 Elytron 客户端进行身份验证。无法访问正在运行的 JBoss EAP 服务器的用户可以使用 WildFly Elytron 工具创建和修改凭证存储，然后客户端可以使用 Elytron 客户端访问凭证存储中的敏感字符串。

以下示例演示了如何在 Elytron 客户端配置文件中使用凭证存储。

带有 Credential Store 的 custom-config.xml 示例

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.7">
    ...
    <credential-stores>
      <credential-store name="my_store"> 1
        <protection-parameter-credentials>
          <credential-store-reference clear-text="pass123"/> 2
        </protection-parameter-credentials>
        <attributes>
          <attribute name="location" value="/path/to/my_store.jceks"/> 3
        </attributes>
      </credential-store>
    </credential-stores>
    ...
    <authentication-configurations>
      <configuration name="my_user">
        <set-host name="localhost"/>
        <set-user-name name="my_user"/>
        <set-mechanism-realm name="ManagementRealm"/>
        <use-provider-sasl-factory/>
        <credentials>
          <credential-store-reference store="my_store" alias="my_user"/> 4
        </credentials>
      </configuration>
    </authentication-configurations>
    ...
  </authentication-client>
</configuration>
```

1. 在 Elytron 客户端配置文件中使用的凭证存储的名称。
2. 凭据存储的密码。
3. 凭证存储文件的路径。
4. 存储在凭证存储中的敏感字符串的凭证引用。

其他资源

- 使用 WildFly Elytron 工具进行凭证存储操作。

1.8. 创建 FIPS 140-2 兼容凭证存储

您可以在 Elytron 中配置联邦信息处理标准(FIPS) 140-2 兼容凭证存储。FIPS 140-2 是由美国开发的计算机安全标准。政府行业工作组以验证加密模块的质量。FIPS 发布（包括 140-2）可在 URL 中找到：http://csrc.nist.gov/publications/PubsFIPS.html

您可以使用两个不同的供应商在 Elytron 中配置 FIPS 140-2 兼容凭证存储：

- SunPKCSFeature 提供程序和网络安全服务(NSS)数据库。

如需更多信息，请参阅使用 SunPKCS Features 提供商和 NSS 数据库创建 FIPS 140-2 兼容凭证存储。

- BouncyCastle 提供程序.
  如需更多信息，请参阅使用 BouncyCastle 供应商创建 FIPS 140-2 兼容凭证存储 。

重要

JBoss EAP 本身未经过 FIPS 认证。JBoss EAP 中的 FIPS 支持级别是 JBoss EAP 可用于 FIPS 认证的加密实现。经过测试的实施是 BouncyCastle 和 SunPKCS facilities。

1.8.1. 使用 SunPKCS demonstrates 供应商和 NSS 数据库创建 FIPS 140-2 兼容凭证存储

NSS 是支持跨平台安全的客户端和服务器应用程序的一组库。您可以在 JBoss EAP 中使用 SunPKCS 提供者和 NSS 库来实现 FIPS 140-2 兼容加密。有关 NSS 的详情，请查看 Mozilla 文档 - 网络安全服务 (NSS)。有关 SunPKCS powerful 供应商的详情，请参考 PKCS featured 参考指南 。

1.8.1.1. 在使用 SunPKCS Features 供应商和 NSS 数据库时支持 FIPS 的 JDK

并非所有 Java 开发套件(JDK)供应商都支持使用网络安全服务(NSS)软件令牌（由 NSS 库实现），为联邦信息处理标准(FIPS) 140-2 合规性所需的网络安全服务(NSS)软件令牌进行配置。在 JBoss EAP 中使用 SunPKCS Features 供应商和 NSS 数据库配置 FIPS 之前，请确保您的 JDK 支持它。

以下是 JBoss EAP 支持的 JDK 列表，支持配置 SunPKCSCapabilities 安全性：

- OpenJDK 11
- OpenJDK 17

其他资源

- 使用 FIPS 在 RHEL 上配置 OpenJDK 11
- 使用 FIPS 在 RHEL 上配置 OpenJDK 17

1.8.1.2. 在启用了 FIPS 的 RHEL 中使用 SUNPKCSPKCS provider 和 NSS 数据库创建 FIPS 140-2 兼容凭证存储

从 Red Hat Enterprise Linux 8.4 开始，如果您启用了联邦信息处理标准(FIPS)系统范围的加密策略，则还会自动启用 Java 的 FIPS。您可以使用默认网络安全服务(NSS)数据库来创建 FIPS 140-2 兼容凭证存储。

在此过程中，**$JAVA_HOME** 代表 JDK 安装路径。以 root 用户身份运行此流程中的命令。

先决条件

- RHEL 中启用了 FIPS。
  您可以使用以下命令检查是否启用了 FIPS：

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

  有关在 RHEL 中启用 FIPS 的详情，请查看以下资源：

  - 在 Red Hat Enterprise Linux 文档中以 FIPS 模式 安装系统。

- 在 Red Hat Enterprise Linux 文档中 将系统切换到 FIPS 模式。

- 已安装 NSS 工具。
  在 Red Hat Enterprise Linux 中，您可以使用 DNF 软件包管理器安装 NSS 工具，如下所示：

```
# dnf install -y nss-tools
```

- 您的 Java 开发套件(JDK)支持使用 NSS 库配置 PKCS vary。
  有关支持 FIPS 的 JDK 的详情，请参考 支持 FIPS 的 JDK。

- JBoss EAP 正在运行。

流程

1. 将 **$JAVA_HOME/conf/security/nss.fips.cfg** 文件中的 **nssDbMode** 值更新为 **readWrite**。

   nss.fips.cfg 内容示例

```
name = NSS-FIPS
nssLibraryDirectory = /usr/lib64
nssSecmodDirectory = sql:/etc/pki/nssdb
nssDbMode = readWrite
nssModule = fips

attributes(*,CKO_SECRET_KEY,CKK_GENERIC_SECRET)={ CKA_SIGN=true }
```

2. 生成 AES secret 密钥以加密凭据存储。

   注意

   您必须在命令中使用存储密码 **NONE**。

   语法

```
# keytool -genseckey -keystore NONE -storetype PKCS11 -storepass NONE -alias
<key_alias> -keyalg <symmetric_key_algorithm> -keysize <key_size>
```

   Example

```
# keytool -genseckey -keystore NONE -storetype PKCS11 -storepass NONE -alias
exampleKeyAlias -keyalg AES -keysize 256
```

3. 验证您可以读取 secret 密钥。

```
# keytool -list -storetype pkcs11 -storepass NONE

Keystore type: PKCS11
Keystore provider: SunPKCS11-NSS-FIPS

Your keystore contains 1 entry

exampleKeyAlias, SecretKeyEntry,
```

4. 将 **$JAVA_HOME/conf/security/nss.fips.cfg** 文件中的 **nssDbMode** 值更新为 **readOnly**。

   nss.fips.cfg 内容示例

   ```
   name = NSS-FIPS
   nssLibraryDirectory = /usr/lib64
   nssSecmodDirectory = sql:/etc/pki/nssdb
   nssDbMode = readOnly
   nssModule = fips

   attributes(*,CKO_SECRET_KEY,CKK_GENERIC_SECRET)={ CKA_SIGN=true }
   ```

5. 在管理 CLI 中，将 SunJCE 提供程序添加到提供程序列表中。

   a. 为 SunJCE 添加供应商加载程序。

   ```
   /subsystem=elytron/provider-loader=SunJCE:add(class-names=
   [com.sun.crypto.provider.SunJCE])
   {"outcome" => "success"}
   ```

   b. 在聚合提供程序中配置 Elytron 和 SunJCE。

   语法

   ```
   /subsystem=elytron/aggregate-providers=<aggregate_provider_name>:add(providers=
   [elytron,SunJCE])
   ```

   Example

   ```
   /subsystem=elytron/aggregate-providers=exampleAggregateProvider:add(providers=
   [elytron,SunJCE])
   {"outcome" => "success"}
   ```

   注意

   提供程序按照命令中定义的顺序调用。

6. 使用 SunJCE 供应商在 Elytron 中创建凭证存储。

   语法

   ```
   /subsystem=elytron/credential-store=<credential_store_name>:add(implementation-
   properties={keyStoreType => PKCS11, external => true, keyAlias => <key_alias>,
   externalPath => <path_where_credential_store_is_to_be_saved>}, modifiable=true,
   credential-reference={clear-text=<password>}, create=true, other-
   providers=<aggregate_provider_name>)
   ```

   Example

   ```
   /subsystem=elytron/credential-store=exampleFipsCredentialStore:add(implementation-
   properties={keyStoreType => PKCS11, external => true, keyAlias => exampleKeyAlias,
   externalPath => /home/ashwin/example.store}, modifiable=true, credential-reference={clear-
   ```

```
text=secret}, create=true, other-providers=exampleAggregateProvider)
{"outcome" => "success"}
```

验证

1. 为凭据存储添加一个别名。

   语法

   ```
   /subsystem=elytron/credential-store=<credential_store_name>:add-alias(alias=<alias>,
   secret-value=<secret_value>)
   ```

   Example

   ```
   /subsystem=elytron/credential-store=exampleFipsCredentialStore:add-
   alias(alias=exampleAlias, secret-value=secret)
   {"outcome" => "success"}
   ```

2. 列出凭证存储中的别名。

   语法

   ```
   /subsystem=elytron/credential-store=<credential_store_name>:read-aliases()
   ```

   Example

   ```
   /subsystem=elytron/credential-store=exampleFipsCredentialStore:read-aliases()
   {
       "outcome" => "success",
       "result" => ["examplealias"]
   }
   ```

创建的凭证存储兼容 FIPS 140-2。

其他资源

- **aggregate-providers** 属性
- **credential-store** 属性
- **credential-store** 实现属性

1.8.2. 使用 BouncyCastle 供应商创建 FIPS 140-2 兼容凭证存储

BouncyCastle 为 Java 和 C# 提供轻量级加密 API。您可以将以下 BouncyCastle 供应商与 JBoss EAP 搭配使用，以创建 FIPS 140-2 兼容凭证存储：

- 用于 Java Cryptography 扩展(JCE)和 Java Cryptography 架构(JCA)的 BouncyCastle FIPS 供应商。
- Java 安全套接字扩展(JSSE)的 BouncyCastle FIPS 供应商。

有关 BouncyCastle 的详情，请查看 Bouncy Castle 的 Legion。

1.8.2.1. 使用 BouncyCastle 供应商创建 FIPS 140-2 兼容凭证存储

从 Red Hat Enterprise Linux 8.4 开始，如果您启用了联邦信息处理标准(FIPS)系统范围的加密策略，OpenJDK 会自动启用不同的安全供应商。其中一个安全供应商是在 FIPS 模式中配置的 SunPKCS11 供应商。您可以按照以下步骤使用 BouncyCastle 供应商创建联邦信息处理标准(FIPS) 140-2 兼容凭证存储。

先决条件

- RHEL 中启用了 FIPS。
  您可以使用以下命令检查是否启用了 FIPS：

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

  有关在 RHEL 中启用 FIPS 的详情，请查看以下资源：

  - 在 Red Hat Enterprise Linux 文档中以 FIPS 模式 安装系统。
  - 在 Red Hat Enterprise Linux 文档中 将系统切换到 FIPS 模式。
- 您的 Java 开发套件(JDK)支持使用 BouncyCastle 供应商配置 FIPS。
  如需更多信息，请参阅 Bouncy Castle - FIPS 资源页的 Java 相关问题。

流程

1. 从以下链接下载 BouncyCastle jars：
   - bc-fips-*N* jar： Bouncy Castle Provider (FIPS 分发) Maven .
   - bctls-fips-*N* jar： Bouncy Castle TLS/JSSE API (FIPS 分发)
     其中 *N* 代表 BouncyCastle FIPS 供应商版本。

     您可以在 Bouncy Castle 的 Legion 处找到有关符合您环境的 BouncyCastle FIPS 供应商的最新认证版本的信息。
2. 创建名为 **java.security** 的配置文件，其内容如下：

   ```
   fips.provider.1=org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
   fips.provider.2=org.bouncycastle.jsse.provider.BouncyCastleJsseProvider fips:BCFIPS
   fips.provider.3=SUN
   fips.provider.4=SunEC
   fips.provider.5=com.sun.net.ssl.internal.ssl.Provider
   ```

   注意

   不要在默认的 **java.security** 文件中修改 FIPS 供应商。建议您使用您自己的 **java.security** 属性文件，如此流程所述。
3. 生成 AES secret 密钥以加密凭据存储。

   语法

   ```
   $ keytool -J-Djava.security.properties=<java_security_file> -genseckey -keystore
   "<keystore_name>" -storetype BCFKS -storepass <store_password> -alias <key_alias> -
   ```

```
keyalg <symmetric_key_algorithm> -keysize <key_size> -keypass <key_password>  -
provider org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider -providerpath
<path_to_bc-fips_jar>  -dname "<certificate_contents>" -validity <validity_in_days>
```

Example

```
$ keytool -J-Djava.security.properties=<path_to_java_security_file>/java.security -genseckey
-keystore "examplekeystore.bcfks" -storetype BCFKS -storepass password -alias
exampleKeyAlias -keyalg AES -keysize 256 -keypass password -provider
org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider -providerpath <path_to_bc-
fips_jar>/bc-fips-1.0.2.jar -dname "CN=localhost" -validity 365
```

4. 验证您可以读取 secret 密钥。

语法

```
$ keytool -J-Djava.security.properties=<java_security_file> -list -keystore <keystore_name> -
storetype BCFKS -storepass <store_password> -provider
org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider -providerpath <path_to_bc-
fips_jar>
```

Example

```
$ keytool -J-Djava.security.properties=<path_to_java_security_file>/java.security -list -
keystore examplekeystore.bcfks -storetype BCFKS -storepass password -provider
org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider -providerpath <path_to_bc-
fips_jar>/bc-fips-1.0.2.jar
```

输出示例

```
Keystore type: BCFKS
Keystore provider: BCFIPS

Your keystore contains 1 entry

exampleKeyAlias, Mar 1, 2023, SecretKeyEntry,
```

5. 启动服务器。

6. 使用管理 CLI，将 JBoss EAP 配置为使用 BouncyCastle 提供程序。

   a. 将 SunJCE 提供程序添加到提供商列表中。

   ```
   /subsystem=elytron/provider-loader=SunJCE:add(class-names=
   [com.sun.crypto.provider.SunJCE])
   ```

   b. 在 JBoss EAP 中将 BouncyCastle 提供程序 jar 作为模块添加。

   语法

   ```
   module add --name=org.bouncycastle.fips --resources=<path_to_bc-
   fips_jar>:<path_to_bctls-fips_jar>
   ```

Example

```
module add --name=org.bouncycastle.fips --resources=<path_to_bc-fips_jar>/bc-fips-1.0.2.jar:<path_to_bctls-fips_jar>/bctls-fips-1.0.2.jar
```

c. 为 BouncyCastle 提供程序添加供应商加载程序。

语法

```
/subsystem=elytron/provider-loader=<provider_loader_name>:add(module=org.bouncycastle.fips)
```

Example

```
/subsystem=elytron/provider-loader=exampleProviderLoader:add(module=org.bouncycastle.fips)
```

d. 在聚合提供程序中配置 BouncyCastle、SunJCE 和 combined-providers。

语法

```
/subsystem=elytron/aggregate-providers=<aggregate_provider_name>:add(providers=[<provider_loader_name>,SunJCE,combined-providers])
```

Example

```
/subsystem=elytron/aggregate-providers=exampleAggregateProvider:add(providers=[exampleProviderLoader,SunJCE,combined-providers])
```

注意

提供程序按照命令中定义的顺序调用。

e. 重新加载服务器。

```
reload
```

7. 使用 BouncyCastle 供应商在 Elytron 中创建凭证存储。

语法

```
/subsystem=elytron/credential-store=<credential_store_name>:add(credential-reference={clear-text=<key_and_keystore_password>},implementation-properties={keyAlias=<key_alias>,external=true,externalPath=<path_to_BCFKS_credential_store>,keyStoreType=BCFKS},create=true,path=<path_to_keystore>,modifiable=true, other-providers=<aggregate_provider_name>)
```

示例

```
/subsystem=elytron/credential-store=exampleFipsCredentialStore:add(credential-reference={clear-text=password},implementation-properties={keyAlias=exampleKeyAlias,external=true,
```

```
externalPath=credentialStore.bcfks, keyStoreType=BCFKS}, create=true,
path=__<path_to_keystore>__/examplekeystore.bcfks, modifiable=true, other-
providers=exampleAggregateProvider)
```

验证

1. 为凭据存储添加一个别名。

   语法

   ```
   /subsystem=elytron/credential-store=<credential_store_name>:add-alias(alias=<alias>,
   secret-value=<secret_value>)
   ```

   示例

   ```
   /subsystem=elytron/credential-store=exampleFipsCredentialStore:add-
   alias(alias=exampleAlias, secret-value=secret)
   ```

2. 列出凭证存储中的别名。

   语法

   ```
   /subsystem=elytron/credential-store=<credential_store_name>:read-aliases()
   ```

   示例

   ```
   /subsystem=elytron/credential-store=exampleFipsCredentialStore:read-aliases()
   {
       "outcome" => "success",
       "result" => ["examplealias"]
   }
   ```

创建的凭证存储兼容 FIPS 140-2。

其他资源

- **aggregate-providers** 属性
- **credential-store** 属性
- **credential-store** 实现属性
- **provider-loader** 属性
- 要了解更多有关 **module add** 命令的信息，您可以在 JBoss EAP 管理 CLI 中运行 **--help** 命令：

  ```
  module add --help
  ```

第 2 章 为 JBOSS EAP 提供一个初始密钥以解锁安全的资源

2.1. ELYTRON 中加密的表达式

要保持敏感字符串的机密性，您可以在服务器配置文件中使用加密的表达式而不是敏感字符串。

加密的表达式是结果，从加密字符串与 SecretKeyCredential 进行加密，然后将它与其编码前缀和解析器名称合并。编码前缀告知 Elytron 表达式是一个加密的表达式。解析器将加密表达式映射到凭据存储中对应的 SecretKeyCredential。

Elytron 中的 **expression=encryption** 资源使用加密表达式来解码运行时其中的加密字符串。通过在配置文件中使用加密的表达式而不是敏感字符串本身，您可以保护字符串的机密性。加密的表达式采用以下格式：

使用特定解析器的语法

```
${ENC::RESOLVER_NAME:ENCRYPTED_STRING}
```

ENC 是表示加密表达式的前缀。

RESOLVER_NAME 是解析程序用来解密加密的字符串。

示例

```
${ENC::initialresolver:RUxZAUMQE+L5zx9LmCRLyh5fjdfl1WM7lhfthKjeoEU+x+RMi6s=}
```

如果您使用默认解析器创建了一个加密表达式，如下所示：

使用默认解析器的语法

```
${ENC::ENCRYPTED_STRING}
```

示例

```
${ENC::RUxZAUMQE+L5zx9LmCRLyh5fjdfl1WM7lhfthKjeoEU+x+RMi6s=}
```

在这种情况下，Elytron 使用您在 **expression=encryption** 资源中定义的默认解析器来解密表达式。您可以对支持它的任何资源属性使用加密表达式。要找出某个属性是否支持加密表达式，请使用 **read-resource-description** 操作，例如：

有关 mail/mail-session的 read-resource-description 示例

```
/subsystem=mail/mail-session=*/:read-resource-description(recursive=true,access-control=none)
{
  "outcome"=>"success",
  "result"=>[{
  ...
    "from"=>{
      ...
      "expression-allowed"=>true,
```

```
    ...
  }]
}
```

在本例中，属性 **from** 支持加密表达式。这意味着，您可以通过加密表达式在 **from** 字段中隐藏您的电子邮件地址。

其他资源

- 在 Elytron 中创建加密的表达式
- **expression=encryption** 属性

2.2. 在 ELYTRON 中创建加密的表达式

从敏感字符串和 SecretKeyCredential 创建一个加密表达式。在管理模型（服务器配置文件）中使用此加密表达式而不是的敏感字符串，来维护敏感字符串的机密性。

先决条件

- 您已在其中创建了一个 PropertiesCredentialStore 和一个 secret 密钥。
  如需更多信息，请参阅为 独立服务器创建 PropertiesCredentialStore/**secret-key-credential-store** 。

流程

1. 使用以下管理 CLI 命令，创建一个在凭证存储中引用现有 SecretKeyCredential 的别名的解析器：

   语法

   ```
   /subsystem=elytron/expression=encryption:add(resolvers=[{name=<name_of_the_resolver>, credential-store=<name_of_credential_store>, secret-key=<secret_key_alias>}])
   ```

   示例

   ```
   /subsystem=elytron/expression=encryption:add(resolvers=[{name=exampleResolver, credential-store=examplePropertiesCredentialStore, secret-key=key}])
   ```

   如果显示与重复资源相关的错误消息，请使用 **list-add** 操作而不是 **add**，如下所示：

   语法

   ```
   /subsystem=elytron/expression=encryption:list-add(name=resolvers, value={name=<name_of_the_resolver>, credential-store=<name_of_credential_store>, secret-key=<secret_key_alias>})
   ```

   示例

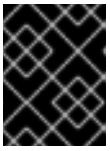
   ```
   /subsystem=elytron/expression=encryption:list-add(name=resolvers,value={name=exampleResolver, credential-store=examplePropertiesCredentialStore, secret-key=key})
   {
   ```

```
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

2. 重新加载服务器。

```
reload
```

3. 在管理 CLI 中禁用命令缓存：



重要

如果不禁用缓存，则可以访问管理 CLI 历史记录文件的任何人都可以看到 secret 密钥。

```
history --disable
```

4. 使用以下管理 CLI 命令创建一个加密表达式：

语法

```
/subsystem=elytron/expression=encryption:create-expression(resolver=<existing_resolver>,
clear-text=<sensitive_string_to_protect>)
```

示例

```
/subsystem=elytron/expression=encryption:create-expression(resolver=exampleResolver,
clear-text=TestPassword)
{
    "outcome" => "success",
    "result" => {"expression" =>
"${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2uGurI=}"}
}
```

${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2uGurI=} 是您在管理模型中使用的加密表达式，而不是 **TestPassword**。

如果您在不同的地方使用相同的纯文本，则在每次使用加密表达式而不是该处的纯文本之前，请重复此命令。当您为同一纯文本重复使用同一命令时，您可以获得同一密钥的不同结果，因为 Elytron 为每个调用使用唯一的初始化向量。

通过使用不同的加密表达式，您可以确保，如果某一字符串的某个加密表达式受到某种程度的破坏，用户无法发现任何其他加密表达式也可能会包含相同的字符串。

5. 使用以下管理 CLI 命令重新启用命令缓存：

```
history --enable
```

其他资源

其他资源

- 使用加密表达式来保护 KeyStore/**credential-store**
- **expression=encryption** 属性

2.3. 使用加密表达式来保护 KEYSTORE/CREDENTIAL-STORE

您可以使用加密表达式来保护 KeyStore。

先决条件

- 您已创建了加密的表达式。
  有关创建加密表达式的详情，请参考 在 Elytron 中创建一个加密的表达式。

流程

- 创建一个 KeyStoreCredentialStore，它使用加密表达式作为 **明文** ：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<path_to_the_credential_store>, create=true, modifiable=true, credential-reference={clear-text=<encrypted_expression>})
  ```

  示例

  ```
  /subsystem=elytron/credential-store=secureKeyStoreCredentialStore:add(path="secureKeyStoreCredentialStore.jceks", relative-to=jboss.server.data.dir, create=true, modifiable=true, credential-reference={clear-text=${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2uGurl=}})
  {"outcome" => "success"}
  ```

其他资源

- **expression=encryption** 属性
- **credential-store** 属性

使用加密表达式保护 KeyStoreCredentialStore 后，您可以在 KeyStoreCredentialStore 中生成一个 **SecretKeyCredential**，并使用 secret 密钥来创建另一个加密表达式。然后，您可以在管理模型（服务器配置文件）中使用这个新加密的表达式而不是敏感字符串。为了安全起见，您可以创建一个完整的凭证存储链。此链使得猜测敏感字符串变得更加困难，因为字符串会受到如下方式的保护：

- 第一个加密表达式为 KeyStoreCredentialStore 的安全。
- 另一个加密表达式保护敏感字符串。
- 要解码敏感字符串，您需要解密加密的表达式。

随着加密表达式的链变得较长，解密敏感字符串会变得比较困难。

第 3 章 参考

3.1. AGGREGATE-PROVIDERS 属性

您可以通过设置 **providers** 属性来配置 **aggregate-providers**。

表 3.1. aggregate-providers Attributes

属性	描述
providers	要聚合的供应商列表。Elytron 使用列表上找到的第一个合适的供应商。

3.2. CREDENTIAL-STORE 属性

您可以通过设置其属性来配置 **credential-store**。

表 3.2. credential-store 属性

属性	描述
create	指定凭证存储是否应该在存储不存在时创建一个。默认值为 **false**。
credential-reference	对用于创建保护参数的凭证的引用。这可以是明文形式，或作为对存储在 **credential-store** 中凭证的引用。
implementation-properties	凭据存储特定实现的属性的映射。
modifiable	能否修改凭证存储。默认值为 **true**。
other-providers	获取提供程序的名称，以搜索可以在凭证存储中创建所需 Jakarta Connectors 对象的那个提供程序。这对于基于 keystore 凭证存储是有效的。如果未指定，则改为使用全局供应商列表。
path	凭证存储的文件名。
provider-name	用于实例化 **CredentialStoreSpi** 的提供程序的名称。如果未指定提供程序，则使用第一个发现的能够创建指定类型实例的提供程序。
providers	获取提供程序的名称，以搜索可以创建所需凭证存储类型的提供程序。如果未指定，则改为使用全局供应商列表。
relative-to	此凭证存储路径相对的基本路径。
type	凭据存储的类型，如 **KeyStoreCredentialStore**。

3.3. CREDENTIAL-STORE 实现属性

您可以通过设置其属性来配置 **credential-store** 实现。

表 3.3. credential-store 实现属性

属性	描述
cryptoAlg	用于加密外部存储处条目的加密算法名称。此属性仅在启用了 **external** 时有效。默认为 **AES**。
external	指定数据是否存储到外部存储并由 **keyAlias** 加密。默认值为 **false**。
externalPath	指定到外部存储的路径。此属性仅在启用了 **external** 时有效。
keyAlias	凭证存储中的 secret 密钥别名，用于加密或解密数据到外部存储。
keyStoreType	密钥存储类型，如 **PKCS11**。默认为 **KeyStore.getDefaultType （）**。

3.4. EXPRESSION=ENCRYPTION 属性

您可以通过设置其属性来配置 **expression=encryption**。

表 3.4. expression=encryption 属性

属性	描述
default-resolver	可选属性。当加密表达式没有使用解析器定义时，要使用的解析器。例如，如果您将 "exampleResolver" 设为 **default-resolver**，并且您使用命令 **/subsystem=elytron/expression=encryption:create-expression(clear-text=TestPassword)** 创建了一个加密表达式，则 Elytron 会使用"exampleResolver"作为该加密表达式的解析器。
prefix	在加密表达式中使用的前缀。默认为 **ENC**。为可能已经定义 **ENC** 的情形中提供此属性。除非与已定义的 **ENC** 前缀冲突，否则不应更改这个值。
解析器	定义的解析器列表。解析器具有以下属性： • **name** - 用于引用它的独立配置的名称。 • **credential-store** - 对包含此解析器使用的 secret 密钥的凭证存储实例的引用。 • **secret-key** - 从给定凭证存储中使用 secret 键的别名。

3.5. PROVIDER-LOADER 属性

您可以通过设置其属性来配置 **provider-loader**。

表 3.5. provider-loader 属性

属性	描述
参数	在 **提供程序** 实例化时，要传递给构造器的参数。
class-names	要加载的供应商完全限定类名称的列表。它们会在服务加载器发现的提供程序后加载，并将跳过任何重复项。
配置	要传递给提供程序的键和值配置，以初始化它。
module	从中加载提供程序的模块名称。
path	用于初始化提供程序的文件的路径。
relative-to	配置文件的基本路径。

3.6. SECRET-KEY-CREDENTIAL-STORE 属性

您可以通过设置其属性来配置 **secret-key-credential-store**。

表 3.6. secret-key-credential-store 属性

属性	描述
create	如果您不希望 Elytron 在其不存在时创建它，请将该值设为 **false**。默认为 **true**。
default-alias	默认生成的密钥的别名名称。默认值为 **key**。
key-size	生成密钥的大小。默认大小为 256 位。您可以将值设为以下之一： • **128** • **192** • **256**
path	凭证存储的路径。
populate	如果凭证存储不包含 **default-alias**，此属性表示 Elytron 是否应该创建一个。默认值是 **true**。
relative-to	对之前定义的属性 **path** 相对的路径的引用。