20.开发者工具

Spring Boot包含了一系列额外工具可以让你的开发体验更好一点。spring-boot-devtools模块可以包含在任何项目中以提供额外的 development-time 功能。如果你想使用这些工具，你应该添加如下依赖：

Maven: org.springframework.boot spring-boot-devtools true

Gradle:

```
dependencies {
    compileOnly("org.springframework.boot:spring-boot-devtools")
}
```

运行完整打包的应用程序时，开发人员工具会自动禁用。如果你的应用程序是使用`java -jar`运行的或者使用一个特殊的类路径，那么它将会被考虑作为一个产品应用。

20.1 默认属性

Spring Boot支持的一些库使用缓存来提高性能。例如，模板引擎将会缓存被便衣的模板，来避免重复解析模板文件。另外，Spring MVC可以在提供静态资源时将HTTP缓存头添加到响应中。

虽然缓存在生产中非常有用，但它在开发过程中可能会产生反作用，从而阻止您看到您在应用程序中所做的更改。基于这个原因，spring-boot-devtools默认将会禁用所有的缓存选项。

缓存选项通常会被`application.properities`文件配置。例如，Thymeleaf提供了`spring.thymeleaf.cache`属性。不需要手动设置属性，`spring-boot-devtools`模块将会自动应用合理的开发环境配置。

20.2 自动重启

使用了`spring-boot-devtools`的应用无论什么时候，类路径中的文件进行更改，它都会重启。当使用一个IDE进行开发时，这是一个非常有用的特点，因为它为代码更改提供了一个非常快速的反馈循环。默认情况下，指向文件夹的类路径上的任何条目都将会监视是否进行更改。值得注意的是，某些资源，例如静态资源和视图模板的更改不需要重启应用程序。

触发重启

因为DevTools仅仅监视类路径资源，唯一地触发重启的方法是更新类路径。更新类路径的方法取决于你使用的IDE。在Eclipse中，保存一个被更新的文件将会造成类路径更新并且触发一个重启。在IDEA中，建造一个项目（`Build -> Build Project`）将会产生相同的效果。

因为DevTools需要隔离的应用程序类加载器才能正常运行，只要你的forking功能是被开启的，你还可以通过支持的构建插件（如Maven或Gradle）启动你的应用程序。

当和LiveReload一起使用时，自动重启会很好的工作。关于更多的细节，请看下面。如果您使用JRebel，则自动重新启动将被禁用，以支持动态类重新加载。其他的devtools功能（例如LiveReload和属性覆盖）依旧能被使用。

重启过程中，DevTools 依赖于应用上下文的 shutdown 钩子去关闭它。如果你禁用了 shutdown 钩子（SpringApplication.setRegisterShutdownHook(false)）它将不能正常工作。

当类路径中的某项进行更改时，决定是否触发重启时，DevTools将会自动忽略名为spring-boot、spring-boot-devtools、spring-boot-autoconfigure、spring-boot-actuator、spring-boot-stater的项目。

DevTools需要去自定义ApplicationContext使用的ResourceLoader。如果你的应用程序已经提供了一个，它将会被包装。直接覆盖ApplicationContext的getResource方法是不被支持的。

重启和重新加载

Spring Boot 提供的重启技术靠两个类加载器实现。没有改变的类（例如，那些第三方包的部分）将会被加载到一个base类加载器中。你开发过程中动态改变的类被加载到一个restart类加载器中。当应用程序被重启时，restart加载器将会被丢弃并建立一个新的。这个方法意味着应用程序重启会比"冷启动"更快，因为base类加载器已经提供并且被填充了。

如果你感觉对于你的应用程序，重启依然不够快，或者你遇到类加载问题，你应该考虑重载技术，例如从ZeroTurnaround的JRebel。这些工作通过在加载类时重写类，使它们更易于重新加载。Spring Loaded提供了另一种选择，但它不支持许多框架，并且它没有商业支持。

20.2.1 包含资源

当某些资源进行改变时，没必要去触发重启。例如，Thymeleaf模板可以就地编辑。默认的在/META-INF/maven、/META-INF/resources、/resources、/static、/public或者/templates将会不触发重启，但是将会触发动态重加载。如果你想自定义这些目录，你可以使用spring.devtools.restart.exclude属性。例如，如果你只想包含/static和/public两个目录，可以按照如下设置：

```
spring.devtools.restart.exclude=static/**,public/**
```

如果你想要保持默认并且添加新的目录，你可以使用spring.devtools.restart.additional-exclude属性代替。

20.2.2 查看另外的目录

你可能想要你的应用程序中，有些文件不是在类路径中的进行改变后，应用程序也会重启或者重新加载。为了做到这个，你可以使用spring.devtools.restart.additional-paths属性去配置另外的需要监听改变的路径。你可以使用上面描述的spring.devtools.restart.exclude属性去控制控制其他路径下的更改是否会触发完全重新启动或仅实时重新加载。

20.2.3 禁用重启

如果你不想用重启这个功能，你可以使用spring.devtools.restart.enabled属性禁用该功能。在大多数情况下你可以在你的application.properties（这样依旧会初始化重启类加载器，但是不会监视文件变化）。

如果你需要完全禁用重启支持，因为它不能和一个特定的库一同工作，你需要在调用SpringApplication.run(...)之前设置一个System属性。例如：

```
public static void main(String [] args)
{
```

```
    System.setProperty("spring.devtools.restart.enabled","false");
    SpringApplication.run(MyApp.class,args);
}
```

20.2.4 使用一个触发器文件

如果您使用持续编译更改文件的IDE，则可能只希望在特定时间触发重新启动。为了完成这项工作，你可以使用一个"触发器文件"，它是个特殊的文件，当你想要触发一个重启检测时，这个文件必须被更改。更改文件只会触发检测，当Devtools检查到它必须要做一些事情时，重启操作才会发生。这个触发器文件会被手动更新，或者通过一个IDE插件更新。

如果你想使用一个触发器文件，请使用`spring.devtools.restart.trigger-file`属性。

你也可以将`spring.devtools.restart.trigger-file`设置成一个全局设置，以便于你的项目都以相同的方式运行。

20.2.5 自定义重启类加载器

正如你前面章节看到的，重启功能的实现使用了两个类加载器。对于大多数应用来说，这样的工作机制是很好的，然而，有时会造成类加载问题。

默认情况下，你的IDE中，任何打开的项目都会使用`restart`类加载器加载，任何常规的`.jar`文件都会被使用`base`类加载器加载。如果你开发的是一个多模块的项目，并且不是每个模块都被你的IDE导入，你可能需要去自定义一些东西。为了完成这个，你需要去创建一个`META-INF/spring-devtools.properties`文件。

这个`spring-devtools.properties`文件包含`restart.exclude.`和`restart.include`前缀的属性。`include`元素都会被拉入到`restart`类加载器，`exclude`元素都会被推进到`base`类加载器中。该属性的值是一个将应用于类路径的正则表达式模式。例如：

```
restart.exclude.companycommonlibs=/mycorp-common-[\\w-]+\.jar
restart.include.projectcommon=/mycorp-myproj-[\\w-]+\.jar
```

所有的属性的键必须是唯一的。只要一个属性是以`restart.include.`或者`restart.exclude.`开头的都会被考虑。

所有的在类路径里的`META-INF/spring-devtools.properties`文件都会被加载。您可以将文件打包到您的项目中，也可以打包到项目使用的库中。

20.2.6 已知的限制

对于使用标准`ObjectInputStream`进行反序列化的对象，重新启动功能不起作用。如果你需要去反序列化数据，你可能需要去使用Spring的`ConfigurableObjectInputStream`并且与`Thread.currentThread().getContextClassLoader()`相组合。

不幸的是，有些第三方库反序列化而没有考虑上下文类加载器。 如果您发现这样的问题，您需要向原作者请求修复。

20.3 LiveReload

spring-boot-devtools模块包括一个嵌入式LiveReload服务器，当一个资源被改变时，它会触发浏览器更新。LiveReload浏览器扩展可免费用于livereload.com的Chrome，Firefox和Safari。

如果你想当你的应用启动时，不启动LiveReload服务器，那么你要去设置spring.devtools.livereload.enabled属性为false。

你只能同时启动一个LiveReload服务器。在你开始启动你的应用程序之前，请确保没有其他的LiveReload服务器运行。如果在你的IDE中同时启动多个应用，仅仅第一个会得到LiveReload支持。

20.4 全局设置

你可以配置全局的devtools设置，靠添加一个名为.spring-boot-devtools.properties到你的$HOME文件夹中（注意文件名以“.”开始）。任何被添加到该文件中的属性都将会被应用到你机器中所有的使用devtools的Spring Boot 应用中。例如，配置使用一个触发器文件时来控制重启时，你应该天下如下内容：

~/.spring-boot-devtools.properties.

```
spring.devtools.reload.trigger-file=.reloadtrigger
```

20.5 远程应用

Spring Boot 开发者工具不仅仅限制在本地开发。你可以在远程运行时使用这几个功能。远程支持是可选的，为了启用它，您需要确保devtools包含在重新打包的存档中：

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludeDevtools>false</excludeDevtools>
            </configuration>
        </plugin>
    </plugins>
</build>
```

然后你需要去设置一个spring.devtools.remote.secret属性，例如：

```
spring.devtools.remote.secret=mysecret
```

启动spring-boot-devtools的远程应用功能是具有安全风险的。在产品部署时，我们从不会启动这个支持。

远程devtools支持分为两个部分；服务器中断接受连接，客户机应用程序运行在你的IDE里。当spring.devtools.remote.secret属性被设置时，服务器组件将会自动启用。客户机组件一定要手动运行。

20.5.1 运行远程客户端程序

远程客户端应用被设计在你的IDE上运行。你需要运行和你连接的项目具有相同类路径的`org.springframework.boot.RemoteSpringApplication`。传递给应用程序的非选项参数应该是您要连接到的远程URL。

例如，如果你是用Eclipse或者STS，并且你有一个部署在Cloud Foundry的叫做`my-app`的项目。你可以按照如下方法做：

- 从`Run`目录中选择`Run Configurations...`。
- 创建一个新的`Java 应用程序`“运行配置”。
- 浏览`my-app`项目。
- 使用`org.springframework.boot.devtools.RemoteSpringApplication`作为主类。
- 添加`https://myapp.cfapps.io`到`Program arguments`中（或者无论你的远程URL是什么）。

运行远程客户端的结果如下：

```
  .   ____          _                                              __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _         ___                _      \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` |        | _ \___ _ __  ___| |_ ___ \ \ \ \
\\/  ___)| |_)| | | | | || (_| []::::::[]   / -_) '  \/ _ \  _/ -_) ) ) ) )
 '  |____| .__|_| |_|_| |_\__, |        |_|_\___|_|_|_\___/\__\___|/ / / /
 =========|_|==============|___/=================================/_/_/_/
:: Spring Boot Remote :: 1.5.12.RELEASE

2015-06-10 18:25:06.632  INFO 14938 --- [           main]
o.s.b.devtools.RemoteSpringApplication   : Starting RemoteSpringApplication on
pwmbp with PID 14938 (/Users/pwebb/projects/spring-boot/code/spring-boot-
devtools/target/classes started by pwebb in /Users/pwebb/projects/spring-
boot/code/spring-boot-samples/spring-boot-sample-devtools)
2015-06-10 18:25:06.671  INFO 14938 --- [           main]
s.c.a.AnnotationConfigApplicationContext : Refreshing
org.springframework.context.annotation.AnnotationConfigApplicationContext@2a17b7b6
: startup date [Wed Jun 10 18:25:06 PDT 2015]; root of context hierarchy
2015-06-10 18:25:07.043  WARN 14938 --- [           main]
o.s.b.d.r.c.RemoteClientConfiguration    : The connection to http://localhost:8080
is insecure. You should use a URL starting with 'https://'.
2015-06-10 18:25:07.074  INFO 14938 --- [           main]
o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port
35729
2015-06-10 18:25:07.130  INFO 14938 --- [           main]
o.s.b.devtools.RemoteSpringApplication   : Started RemoteSpringApplication in 0.74
seconds (JVM running for 1.105)
```

因为远程客户端使用的类路径和真实的应用程序相同，因此它能直接读取应用程序属性。这是为什么`spring.devtools.remote.secret`属性被读和传到服务器用于验证。

建议使用`https://`作为连接协议，以便流量加密并且密码不会被拦截。

如果你需要使用代理连接远程应用，配置`spring.devtools.remote.proxy.host`和`spring.devtools.remote.proxy.port`属性。

20.5.2 远程更新

远程客户端将会监视你的应用程序类路径的改变，就像本地重启一样。任何更新的资源都将会被推到远程应用中然后触发重启。如果你正在迭代一个你的本地没有的云服务功能，这可能是十分有用的。通常，远程更新和重启比完整的重建和部署周期快得多。

只有在远程客户端运行时文件才能被监视。如果是在启动远程客户端之前改变一个文件，这些更新将不会被推到远程服务器中。

20.5.3 远程debug通道

Java远程调试在诊断远程应用程序的问题时非常有用。不幸的事，当你的应用程序部署在你的数据中心之外时，启动远程debug不是总是能实现的。如果您正在使用基于容器的技术（如Docker），则远程调试也可能会非常棘手。

为了解决这些限制，devtools支持通过HTTP对远程调试流量进行隧道传输。建立连接后，调试流量将通过HTTP发送到远程应用程序。 如果你想使用不同的端口，你可以使用spring.devtools.remote.debug.local-port属性。您需要确保您的远程应用程序启动时启用了远程调试。 通常可以通过配置JAVA_OPTS来实现。 例如，使用Cloud Foundry，您可以将以下内容添加到您的manifest.yml中：

```
---
    env:
        JAVA_OPTS: "-Xdebug -Xrunjdwp:server=y,transport=dt_socket,suspend=n"
```