### 4.7.4 内置的Servlet容器支持

Spring Boot内置了很多对Servlet容器的支持类，这些Servlet容器包括Tomcat、Jetty、与Undertow等，大多数的开发者都会使用一个合适的Servlet Starter得到一个已经完全配置好的容器实例，缺省情况下，内置的web服务器监听8080端口上的HTTP请求。

- Servlets、Filters与Listeners

当使用内置的servlet容器时，你可以通过使用Spring Bean或者和扫描Servlet组件的方式注册Servlet标准中的servlets、过滤器、监听器等。

- 通过Spring Bean的方式注册Servlets、过滤器、监听器

任何以Spring Bean形式存在的Servlet、Filter、或者监听器对象实例，都会注册到内嵌的web容器中，This can be particularly convenient if you want to refer to a value from your application.properties during configuration.

缺省情况下，如果上下文中只包含一个Servlet，它映射的路径就是/，在多个Servlet的情况下，servlet的名字会作为映射的路径的前缀，过滤器都是映射到/\*路径的。

# 第五部分 Spring Boot特性

## 5.29 创建自己的自动配置

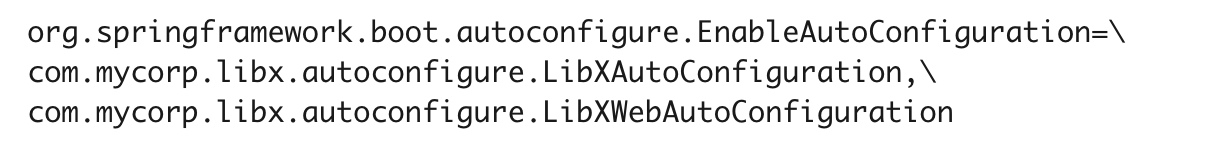
如果你正在开发库，而不是业务代码，你可能想要开发自己的自动配置功能，自动配置的相关的类可以分布在外部的jar包中，并且是可以被Spring Boot处理到的；自动配置的代码与其他前面的自动配置类似，通常是分布在一个一个starter的库中，我们将讲述如何创建自己的自动配置，然后讲述如何按照标准方式创建自己的starter。

#### 5.29.1 理解自动配置Bean

实际上，自动配置是通过@Configuration注解与被注解的类实现的，额外的@Conditional注解可以用来限制自动配置的应用场景，通常，自动配置类都会使用更好用的@ConditionalOnClass与@ConditionalOnMissingBean注解，这会确保，自动配置涉及的相关类存在并且你没有声明你自己的@Configuration，你可以查看spring-boot-autoconfigure的源代码来查看Spring提供的@Configuration类或者查看META-INF/spring.factories文件的内容。

#### 5.29.2 加载自动配置的候选者

Spring Boot会检查你发布的jar包中是有有META-INF/spring.factories，这个文件会把自动配置类列在org.springframework.boot.autoconfigure.EnableAutoConfiguration这个key后面，如下所示：



自动配置类只能通过这样的方式加载，你需要确保它们定义在一个不能被组件扫描扫描到的包空间下，此外，自动配置类不应该开启组件扫描来加载额外的组件应该使用@Import的方式来加载Bean；如果需要特定的自动配置加载的顺序，可以使用@AutoConfigureAfter与@AutoConfigureBefore注解，比如，你想应用一些与Web有关的自动配置，你的自动配置类可能想在WebMvcAutoConfiguration类之后加载。如果你就想安排一些可能没啥关联的自动配置的顺序，你可以使用@AutoConfigureOrder，这个注解提供了@Order注解类似的功能；自动配置类应用的顺序只会影响BeanDefinition的顺序，或者菜单的顺序，bean的实际创建顺序是不受影响的，只会由依赖关系与@DependsOn决定。

### 5.29.3 条件注解

大多数场景下，你都可能需要在自动配置类上使用@Conditional注解，@ConditionalOnMissingBean注解更加常用的条件，可以让开发者覆盖自动的配置；Spring提供了大量的@Conditional注解，你可以用来条件化@Configuration类与@Bean类，这些注解包括4类。

#### 5.29.3.1 类条件

@ConditionalOnClass与@ConditionalOnMissingClass注解让自动配置类在特定类存在或者不存在的情况下加载，由于注解元数据是由ASM处理的，你可以使用value指定类即使类不存在classpath目录下，也可以使用name指定class的名字，当在@Bean的场景下且返回类型就是Condition定义的类型下，这个机制不会生效，因为，在条件应用前，JVM已经加载了目标类型的类；这种情况下，可以把Bean隔离成一个单独的@Configuration注解类，如下图：



#### 5.29.3.2 Bean条件

@ConditionalOnbean注解与@ConditionalOnMissingBean注解可以在bean出现或者没有出现的情况下决定是否加载，使用value指定bean的类型，使用name指定bean的name，search指定搜索bean的ApplicationContext；当应用在@Bean方法上时，目标的bean的类型就是方法返回的类型，如下图：



你必须要注意bean definition加载的顺序，因为条件计算时只会依据当前已经加载的上下文环境计算，所以，我们建议你只在自动配置类上使用@ConditionalOnBean或者@ConditionalOnMissingBean注解，这是因为自动配置类一定是在用户自定的bean定义加载后才会加载；@ConditionalOnBean或者@ConditionalOnMissingBean注解不会阻止@Configuration类的创建，放在类上的情况除外，放在@bean方法上就会创建@Configuration类的bean，当使用@Bean的方式声明一个bean时，方法的返回类型尽可能是比较具体的类型，比如是具体类而不是接口，因为这对对于bean条件的计算是特别有帮助的。

#### 5.29.3.3 属性条件

@ConditionalOnProperty注解是根据Spring环境中的属性来判断是否加载。

#### 5.29.3.4 资源条件

@ConditionalOnResource注解是判断是否存在指定的资源，资源使用常规的Spring描方式

#### 5.29.3.5 Web应用条件

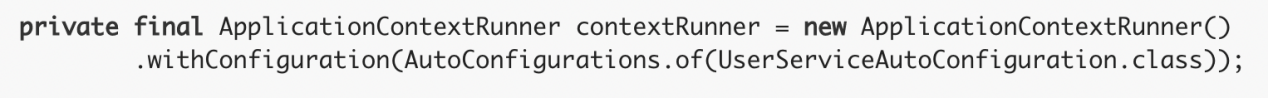
@ConditionalOnWebApplication与@ConditionalOnNotWebApplication注解根据依据是否是一个Web应用决定是否加载。

#### 5.29.3.6 SpEL表达式条件

@ConditionalOnExpression 注解可以让配置依据SpEL表达式的条件计算结果确定是否加载。在表达式中应用一个bean会造成bean被很早的初始化，因此，bean可能不会被post-processing处理（比如configuration properties绑定），因此他的状态可能不是完整的。

### 5.29.4 测试你的自动配置

自动配置收到很多方面的影响，用户配置（用户的@bean或者环境配置）、条件计算逻辑或者其他，具体的，每个测试都需要创建一个ApplicationContext，ApplicationContextRunner可以帮忙创建，ApplicationContextRunner通常定义成测试类的一个成员，收集一些共同的基础的配置，如下图：



每个测试都可以使用这个runner来执行一个测试用例，如下图：



也可以非常容易的定制Enviroment，如下图：



Runner也可以用来展示条件的计算结果，这个计算结果，可以通过日志的方式打印，如下：

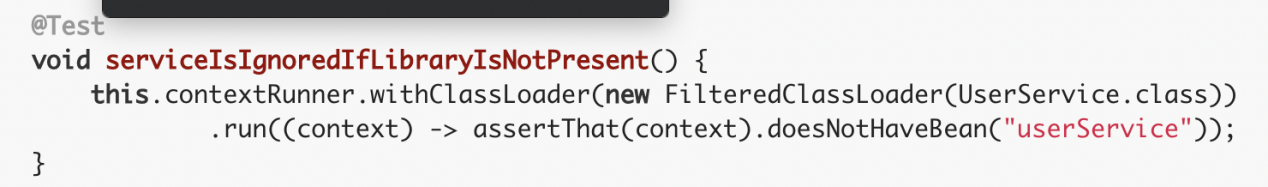


#### 5.29.4.1 仿真一个Web环境

如果你想要测试基于Servlet的Web环境的自动配置，可以使用WebApplicationContextRunner；

#### 5.29.4.2 覆盖classpath

也可以测试当一个类或者一个包在运行时不存在会发生什么，可以使用FilteredClassLoader来模拟类不存在情况，如下图：



### 5.29.5 创建你自己的Starter

一个传统的Spring Boot starter会包含自动配置的代码与定制化基础设置的技术（ACME），为了让自动配置更容易扩展，把大量的用于自动配置的属性抽离出来，放到开放的环境中，最终，一个用于简化Spring开发的starter依赖就有了。

具体来说，一个自定义的starter可以包含如下的部分：

* 包含自动配置代码的自动配置模块；
* starter模块，提供了自动配置的依赖与其他的一些额外的依赖。

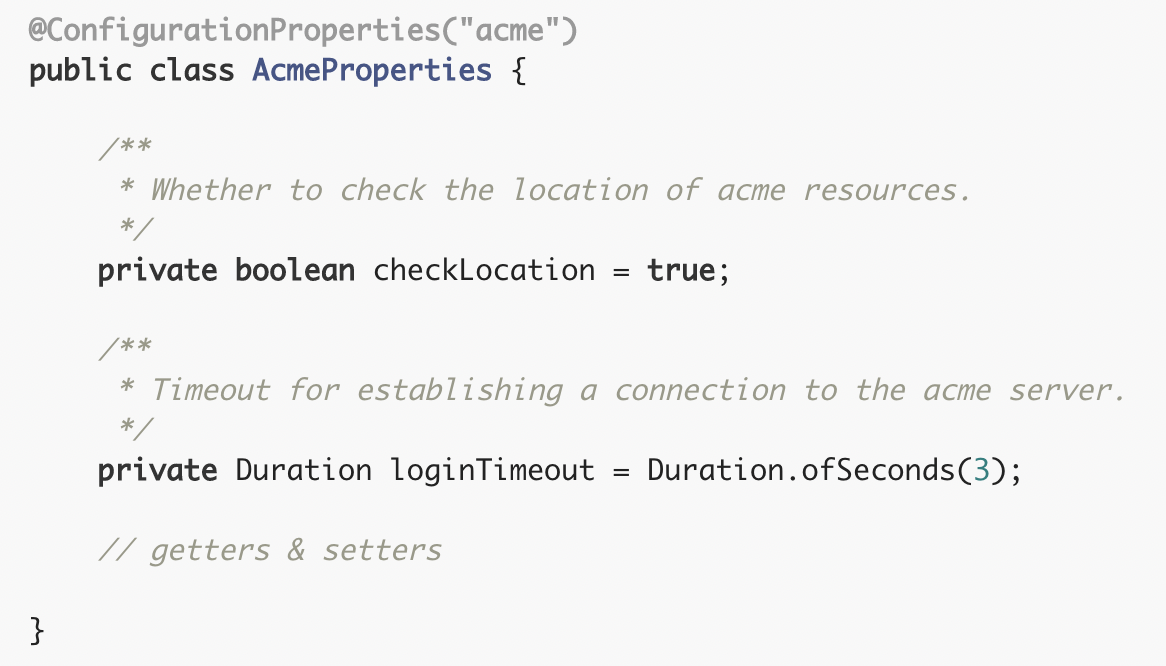
分为2个模块是完全没有必要的，但是如果自动配置有几种不同的形式或者选项或者可选的特性等，那么最好是分为2个模块，因为这样，你可以在不同的starter中提供这些可选的特性，其他人也可以基于你的autoconfigure模块构造它们自己的starter；如果你的自动配置是简单并明了的，没有一些可选的特性，那么只需要提供一个starter模块就可以了。

#### 5.29.5.1 命名

你必须给你的starter设置一个适当的名字空间，不要让你的模块名叫spring-boot；作为一个经验法则，你需要给starter的各个模块命名，例如：假设你要创建一个一个acme的starter，你的自动配置模块可能叫acme-spring-boot， starter模块叫做acme-spring-boot-starter，如果你只有一个starter，那么叫做acme-spring-boot-starter就可以了。

#### 5.29.5.2 属性配置

如果你的starter提供了配置属性，最好放到一个名字空间下面，最好不要定义在Spring Boot的标准名字空间下，因为标准名字空间未来可能会发生变更，经验使然，名嘴空间是你持有的并且唯一的会更好一些。确保每个属性都通过javadoc的方式做了文档化注释，如下：



下面是一些属性配置注释文档的编写规则：

* 不要用the或者A开头；
* 对于boolean类型，使用Whether或者Enable开头；
* 对于集合类型，使用逗号分隔列表；
* 使用Duration类型不要使用long类型表示时间；
* 不要在注释中写默认值，除非默认值只有在运行时决定的情况除外；

确保 [trigger meta-data generation](https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-configuration-metadata.html#configuration-metadata-annotation-processor)，这样IDE辅助可以提供对与属性配置的提示功能，你可能想要查看生成器的metadata（META-INF/spring-configuration-metadata.json）来确保你的属性配置被正确的文档化了。

#### 5.29.5.3 autoconfigure模块

autoconfigure模块包括启动应用时，库做的所有的启动工作，也包括属性定义，还有一些回调接口用来进一步定制化组件的初始化；Spring Boot使用注解处理器收集元数据文件自动配置类上的条件，