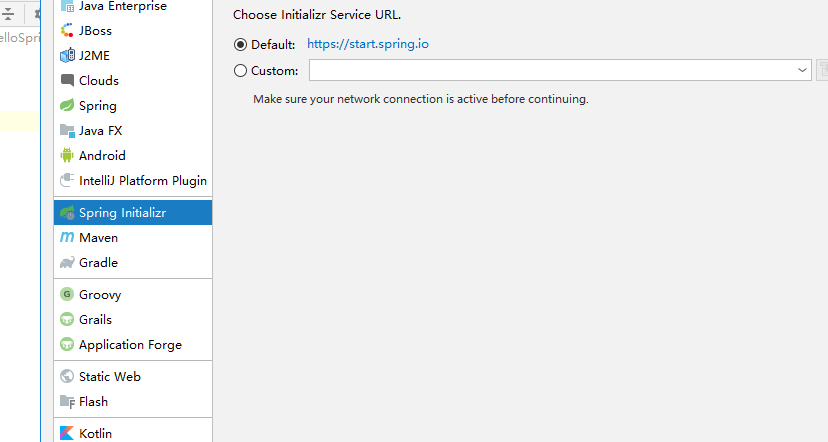
1. 快速搭建spring应用

 使用 Spring Tool Suite 生成 Starter 代码；

第一个方案是使用“File | New | Spring Starter Project”导航菜单

 使用 IntelliJ IDEA 14.1，它对 Spring Boot 提供了良好的支持；

File-new-project-Spring Initializr



 借助 Spring 站点，从 http://start.Spring.io 上下载可配置的 ZIP 文件；

 使用到 http://start.Spring.io 站点的 curl 命令来达到相同的效果。

**$ curl http://start.Spring.io/starter.tgz \**

**-d name=masterSpringMvc \**

**-d dependencies=web \**

**-d language=java \**

**-d JavaVersion=1.8 \**

**-d type=gradle-project \**

**-d packageName=masterSpringMvc \**

**-d packaging=jar \**

**-d baseDir=app | tar -xzvf -**

1.5

GIT

如果你还不了解 Git 的话，我推荐下面的两个教程：  https://try.github.io，这是一个很好的交互式教程，可以引导你一的 Git 命令；  http://pcottle.github.io/learnGitBranching，这是一个很棒的教程，它将 树形的结构进行了可视化，它同时展现了 Git 的基本和高级功能。

1.5.1

Gradle

生成包装器的命令是 gradle wrapper：

**$ ./gradlew bootrun**

执行上面的命令将会运行一个嵌入式的 Tomcat，应用会位于它里面！

Spring Boot 插件将会产生一个胖（fat）JAR 文件，其中包含了项目的所有依赖。要构

建的话，只需输入：

**./gradlew build**

我们将会在“build/libs”目录下找到这个 JAR 文件。该目录下会包含两个文件，其中

一个为胖 JAR 包，名为 masterSpringMvc-0.0.1-SNAPSHOT.jar，另外一个是普通的 JAR 文

件，名为 masterSpringMvc-0.0.1-SNAPSHOT. jar.original，这个文件不包含任何的依赖。

1.5.2关注代码

需要注意的是@SpringBootApplication 注解，如果看一下这个注解的代码的话，就会发

现它实际上组合了 3 个其他的注解，也就是@Configuration、@EnableAutoConfiguration 和

@ComponentScan：

@ComponentScan 也是一个比较经典的注解，它会告诉 Spring 去哪里查找 Spring 组

件（服务、控制器等）。在默认情况下，这个注解将会扫描当前包以及该包下面的所有

子包。

在这里，比较新颖的是@EnableAutoConfiguration 注解，它会指导 Spring Boot 发挥其

魔力。如果你将其移除掉的话，就无法从 Spring Boot 的自动配置中收益了。

使用 Spring Boot 来编写 MVC 应用的第一步通常是在代码中添加控制器。将控制器放

到 controller 子包中，

HelloWorld例子

代码：

package masterSpringMvc.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.ResponseBody;

@Controller

public class HelloController {

@RequestMapping("/")

@ResponseBody

public String hello() {

return "Hello, world!";

}

}

1.6 幕后的 Spring Boot

如果你之前搭建过 Spring MVC 应用，那么可能已经习惯于编写相关的 XML 文件或 Java

注解配置类。

一般来讲，初始的步骤如下所示：

1．初始化 Spring MVC 的 DispatcherServlet；

2．搭建转码过滤器，保证客户端请求进行正确地转码；

3．搭建视图解析器（view resolver），告诉 Spring 去哪里查找视图，以及它们是使用哪

种方言编写的（JSP、Thymeleaf 模板等）；

4．配置静态资源的位置（CSS、JS）；

5．配置所支持的地域以及资源 bundle；

6．配置 multipart 解析器，保证文件上传能够正常工作；

7．将 Tomcat 或 Jetty 包含进来，从而能够在 Web 服务器上运行我们的应用；

8．建立错误页面（如 404）。

不过，Spring Boot 为我们处理了所有的事情。因为这些配置一般是与应用相关的，所

以你可以无限制地将它们进行组合。

1.6.1 分发器和 multipart 配置

接下来，让我们看一下在幕后到底发生了什么。

我们使用默认生成的Spring Boot 配置文件，并将其设置为debug 模式。在src/main/resources/

application.properties 中添加下面这一行：18 精通 Spring MVC 4

debug=true

现在，如果重新启动应用的话，就能看到 Spring Boot 的自动配置报告。它分为两部分：一

部分是匹配上的（positive matches），列出了应用中，所有的自动配置，另一部分是没有匹配

上的（negative matches），这部分是应用在启动的时候，需求没有满足的 Spring Boot 自动配置： 重新启动后日志。

现在，如果重新启动应用的话，就能看到 Spring Boot 的自动配置报告。它分为两部分：一

部分是匹配上的（positive matches），列出了应用中，所有的自动配置，另一部分是没有匹配

上的（negative matches），这部分是应用在启动的时候，需求没有满足的 Spring Boot 自动配置



DispatcherServletAutoConfiguration：

@Order(Ordered.HIGHEST\_PRECEDENCE)

@Configuration

@ConditionalOnWebApplication

@ConditionalOnClass(DispatcherServlet.class)

@AutoConfigureAfter(EmbeddedServletContainerAutoConfiguration.class)

这个文件中还包含了 Spring MVC 分发器 Servlet 和 multipart 解析器的典型配置。整个

Spring MVC 配置被拆分到了多个文件之中。

另外，值得一提的是，这些 bean 会遵循特定的规则，以此来检查是否处于激活状态。在

@Conditional(DefaultDispatcherServletCondition.class)条件满足的情况下，ServletRegistrationBean 函

数才会启用，这有些复杂，但是能够检查在你的配置中，是否已经注册了分发器 Servlet。

只有在满足@ConditionalOnMissingBean(name=DispatcherServlet.MULTIPART\_RESOLVER\_

BEAN\_NAME)条件的情况下，MultipartResolver 函数才会处于激活状态，例如，当我们自

己还没有注册的时候。

这意味着 Spring Boot 仅仅是基于常见的使用场景，帮助我们对应用进行配置。不过，

可以在任意的地方覆盖这些默认值，并声明自己的配置。

因此，通过查看 DispatcherServletAutoConfiguration，就了解了为什么我们已经拥有了

分发器 Servlet 和 multipart 解析器。

1.6.2 视图解析器、静态资源以及区域配置