Table of Contents

关于关书

　　ＳｐｒｉｎgＢｏｏｔ的目标在于简化Ｓｐｒｉｎｇ应用程序的开发。正因如此，Ｓｐｒｉｎｇ Boot涉及到了Ｓｐｒｉｎｇ所涉及的方方面面。虽然本书不能覆盖每一个Ｓｐｒｉｎｇ Boot可以应用的地方，不过我们会涉及到Ｓｐｒｉｎｇ所支持的每种技术。另外，Ｓｐｒｉｎｇ Booｔ实战旨在讨论ＳｐｒｉｎｇＢｏｏｔ中提炼出的四个主题：自动配置，启动依赖，命令行接口以及执行器。 沿着这几个主题，我们可能会涉及到Ｓｐｒｉｎｇ的一些特性，但最主要的研究依然还是Spring Ｂｏｏｔ。

　　Ｓｐｒｉｎｇ Boot实战面向所有ｊａｖａ开发者，所以需要有一些了解Ｓｐｒｉｎｇ相关的知识作为前提。Spring Boot使得Ｓｐｒｉｎｇ的核心变得更好更适合使用，这对使用Ｓｐｒｉｎｇ来说是种全新的体验。 尽管如此，本书还是主要聚焦于Ｓｐｒｉｎｇ Boot，不会对Ｓｐｒｉｎｇ做过多的深入研究，如果需要你可以参考Manning实战系列中的Spring实战来获取更多和Spring相关的知识。

本书导航

Spring实战分为了以下8章:

- 在第一章中，我们会对SpringBoot有一个整体认识，以介绍为主，介绍的要点有自动配置，启动依赖，命令行接口以及执行器。
- 在第二章中，主要是对SpringBoot的自动配置和启动依赖进行更深的研究，你将会使用很少的显式配置来构建一个完整的Spring应用程序。
- 在第三章中，主要承接了第二章中的提到的部分，展示了如何通过设置应用程序的属性影响自动配置或者完全的重写自动化的配置，当其不满足你的需求的时候。
- 在第四章中，我们会研究如何为Spring Boot应用程序写自动化集成测试。
- 在第五章中，你将会看到Spring Boot CLI如何为传统Java开发提供吸引人可供选择的方式。即通过编写一些Groovy的脚本文件，并通过命令行运行。
- 在我们研究Groovy这个主题期间，第六章我们会了解下Grails 3，一起看看这个基于SpringRoot的框架最近的版本。
- 在第七章中，你会看到Spring Boot执行器的影响力，挖掘出是什么在执行应用程序的时候运作。你将会看到如何通过WebEndpoint即是Remote Shell，也是JMX MBeans，从而窥视到一个应用才程序的内部。
- 在第八章中，讨论了各种部署你的Spring Boot应用程序的选项，其中包括了传统的应用服务器部署和云部署。

代码约定和下载

整本书有许多示例代码，这些代码会食用固定的宽度的代码字体。本书正文中的类名，方法名或者XML代码也都使用代码字体。

许多Ｓｐｒｉｎｇ的类或者包名有着格外的长度但是富有意义的名字。鉴于此，我们有时候会用到换行符(➥)。

书中的示例代码并不都是完整的。为了关注某个主题，我们有时候只会展示类的一个或者两个方法。

本书构建的应用程序完整代码可以在Ｍａｎｎｉｎｇ出版社站点下载。地址是：www.manning.com/books/spring-boot-in-action

作者在线

购买了ＳｐｒｉｎｇＢｏｏｔ实战的读者可以免费访问Ｍａｎｎｉｎｇ出版社的在线论坛，并可以对本书做出评价，提出技术问题，接受到作者的或者其他用户的帮助等等。要进入这个论坛或者订阅它，你可以在浏览中访问www.manning.com/books/spring-boot-in-action 这个页面会告诉你怎样注册后进入论坛，能够得到什么帮助以及论坛的规则。

Manning出版社为读者提供了一个交流平台，在这里读者之间以及读者和作者之间可以进行有意义的交流。对于作者来说，对论坛进行多少次访问不是强制的，他们对本书论坛的贡献是资源和免费的。我们建议你尽量向作者问一些有挑战性的问题，以保持他们的兴趣！

只要本书还在印刷，读者就可以访问作者在线论坛以及以前讨论的归档信息。

关于封面插图

ＳｐｒｉｎｇBoot实战的封皮图片叫做：“鞑靼人在喀山的习惯”，而喀山正是俄罗斯鞑靼斯坦共和国的首都。这个插图是取自Thomas Jefferys所编著的不同国家服饰的古代现代图集（共四卷），此书于１７５７－１７７２伦敦出版。The title page states that these are hand-colored copperplate engravings, heightened with gum ara- bic. Thomas Jefferys (1719–1771) was called “Geographer to King George III.” He was an English cartographer who was the leading map supplier of his day. He engraved and printed maps for government and other official bodies and produced a wide range of commercial maps and atlases, especially of North America. His work as a mapmaker sparked an interest in local dress customs of the lands he surveyed and mapped, which are brilliantly displayed in this collection.

Fascination with faraway lands and travel for pleasure were relatively new phenom- ena in the late eighteenth century, and collections such as this one were popular, introducing both the tourist as well as the armchair traveler to the inhabitants of other countries. The diversity of the drawings in Jefferys’ volumes speaks vividly of the uniqueness and individuality of the world’s nations some 200 years ago. Dress codes have changed since then, and the diversity by region and country, so rich at the time, has faded away. It is now often hard to tell the inhabitant of one continent from another. Perhaps, trying to view it optimistically, we have traded a cultural and visual diversity for a more varied personal life.

Or a more varied and interesting intellectual and technical life.

此时你很难分辨出这是一本有关计算机的书，Manning出版社通过Jeffreys的图片反映了两个世纪前各个地区生活的多样性，并借此来赞美计算机行业的创意，进取和乐趣。

第一章: 从Spring谈起

本章内容

- Spring Boot如何简化Spring应用程序的开发
- Spring Boot的必要特征
- 配置一个Spring Boot工作空间

Spring框架已经存在了十多年了，事实上，作为java应用程序开发中的标准框架它已经建立其一席之地了。在这样长且传奇的历史上，有些人可能觉得Spring已经尘埃落定了或者是在吃老本，它不会再有什么任何新的令人激动的地方了。有些也可能会说Spring已经成为了遗产了，是时候寻找其他框架来调用了。

然而，这些人错了。

有许多新的和令人振奋的东西诞生在了Spring生态系统中，包括了云计算，大数据，无模式的数据持久化（NoSQL），响应式编程和客户端应用开发等等。

或许最令人激动，最引人注目的，最划时代的新事物是去年左右在Spring生态系统中诞生的Spring Boot。Spring Boot为可以以最小的冲突开发Spring应用程序提供了一种新的模式。通过使用Spring Boot，你可以使得开发Spring应用程序更加的灵活，敏捷，更可以以最小化甚至于无的思想配置Spring本身来处理应用程序的功能需求。事实上，Spring最主要的事情之一就是使得Spring的使用不在成为阻挡你的路线，从而使得事情得以很好的解决。

纵观整本书的所有章节，我们将探讨使用Spring Boot开发的各个方面。但是，首先让我们俯视下Spring Boot到底提供了什么。

１・１　重启Ｓｐｒｉｎｇ之旅

Ｓpring 起初作为Ｊ２ＥＥ可选的一种轻量级框架,而不是和ＥＪＢ这种重量级框架一样开发组件，Ｓｐｒｉｎｇ提供了更加简单的方法实现ｊａｖａ企业级的开发，它使用依赖注入，面向切面编程等来实现和ＥJB的ＰＯＪＯ对象一样的能力。

但是，尽管Ｓｐｒｉｎｇ在组件代码上是轻量级的，可在配置方面确实重量级的。起初，Ｓｐｒｉｎｇ的配置采用了大量的ＸＭＬ，Spring2.5提出了基于注解的组件扫描的方式，这种方式消除了在配置应用程序组件的许多显式的ＸＭＬ配置。Ｓｐｒｉｎｇ３・０更是以基于Ｊａｖａ的类的配置方式，这种方式相对于ＸＭＬ类型安全而且可以重构。

尽管如此，还是逃离不了配置。尤其体现在启用Ｓｐｒｉｎｇ的一些功能上，比如事务管理，ＳｐｒｉｎｇＭＶＣ需要的显式配置，还有就是启用一些第三方的库比如Thymeleaf-模板引擎的配置，不管是基于ＸＭＬ还是Ｊａｖａ类，这些都逃离不了。另外，配置Ｓｅｒｖｌｅｔ或者Ｆｉｌｔｅｒ一样需要在Ｗｅｂ.xml显式配置或者初始化好。组件扫描的方式的确减少了一些配置，Ｊａｖａ类的配置也使得配置少些尴尬了，但Ｓｐｒｉｎｇ仍需要大量的配置。

所有的这些配置为开发带来了冲突，我们花费了更多时间在配置上而不是编写程序的业务逻辑上。要想实现所想的转变需要考虑如何将配置Spring转移到解决业务问题上。像大多框架一样，Ｓｐｒｉｎｇ为我们做了很多，但反过来这却也需要你为之做很多。

此外,项目的依赖管理是一个吃力不讨好的任务。在确定什么库可以作为项目构建的部分变得越发艰难，甚至要知道那些版本和那些版本可以很好的配合等等。这些一样很重要，这使得项目的依赖管理也为开发带来了冲突。当你添加依赖去构建项目，我不再是写程序的代码，任何选择这些错误的依赖版本所产生的不兼容都会变成项目的杀手。

然而，Ｓｐｒｉｎｇ　Ｂｏｏｔ改变了这一切。

１．１．１ 崭新的眼光看待Ｓｐｒｉｎｇ

假定你有一个任务，使用Ｓｐｒｉｎｇ开发一个非常简单的ＨｅｌｌｏＷｏｒｌｄ的ｗｅｂ应用程序。你需要做什么呢？我能想到的东西屈指可数，你会 需要最低限度：

- 一个项目结构，使用Ｍａｖｅｎ或者Ｇｒａｄｌｅ来构建依赖，至少，你需要ＳｐｒｉｎｇＭＶＣ和ＳｅｒｖｌｅｔＡＰＩ来作为依赖
- 一个ｗｅｂ.xml文件，或者WebApplicationInitializer的实现，并且生命Ｓｐｒｉｎｇ的DispatcherServlet
- 一个Ｓｐｒｉｎｇ配置，使得ＳｐｉｎｇＭＶＣ生效
- 一个Ｃｏｎｔｒｏｌｌｅｒ类，可以响应“ＨｅｌｌｏＷｏｒｌｄ”的ＨＴＴＰ请求
- 一个ｗｅｂ应用程序服务器，比如Ｔｏｍｃａｔ，来部署应用到上面。

以上列表中最引人瞩目的一个其实只有一个是明确用来开发ＨｅｌｌｏＷｏｒｌｄ功能的：ｔｈｅ　Ｃｏｎｔｒｏｌｌｅｒ。其余的都是样板化的，在任何一个使用Ｓｐｉｎｇ开发的ｗｅｂ应用程序中都会需要，你为什么还总要提供呢？

假定有这么一种情况，只有controller是你需要的。最终证明，代码清单１－１中的基于Ｇｒｏｏｖｙ的Ｃｏｎｃｏｌｌｅｒ就是一个Ｓｐｒｉｎｇ应用程序完整的例子。

代码清单１－１

```
@RestController
class HelloController {
  @RequestMapping("/")
  def hello() {
    return "Hello World"
  }
}
```

这里并没有过多的配置，没有Weｂ.xml，不需要构建规范，甚至不需要应用程序服务器。这便是整个应用程序。Ｓｐｒｉｎｇ　Ｂｏｏｔ会处理执行应用程序的组织工作。你只需要准备你的程序代码即可了。

假定你已经安装了Ｓｐｒｉｎｇ　Ｂｏｏｔ　ＣＬＩ，你可以在命令行执行ＨｅｌｌｏＣｏｎｔｒｏｌｌｅｒ，如下所示：

```
$ spring run HelloController.groovy
```

你可能也注意到，甚至不需要编译代码，Ｓｐｒｉｎｇ Ｂｏｏｔ ＣＬＩ可以以未编译的形式运行。

我选择使用Ｇｒｏｏｖｙ写这个例子是因为使用Ｇｒｏｏｖｙ语言可以很好的展现Ｓｐ

ｒｉｎｇＢｏｏｔ的简单性。当然，Ｓｐｒｉｎｇ Ｂｏｏｔ不一定非得需要你使用Ｇｒｏｏｖｙ。事实上，本书中大多数写的代码我将会使用Ｊａｖａ语言，不过可能会有一些Ｇｒｏｏｖｙ语言穿插其中，这样更合适一些。

你可以提前看下１．２１，安装下Ｓｐｒｉｎｇ Ｂｏｏｔ CLI，试着运行下这个小ｗｅｂ应用程序，这样你可以感觉到Ｓｐｒｉｎｇ Boot为Spring应用程序的开发带提供了那些关键部分。

1.1.2 审视Spring Boot要点

Spring Boot为Spring应用程序开发带来了伟大的魔法。它的执行具备了以下4个核心手段：

```
* 自动配置——Ｓｐｒｉｎｇ Boot可以按应用程序功能性自动进行配置，这普遍适用于多数Ｓｐｒｉｎｇ应用程序。
* 启动依赖——你可告诉Ｓｐｒｉｎｇ Boot你需要什么功能，它将会确保将需要的库添加进来以构建项目。
* 命令行接口——这一个特点可以让你只需写完整的应用程序代码，而无需关心传统项目的构建。
* 执行器——让你深入了解执行Ｓｐｒｉｎｇ Boot应用程序中发生了什么。
```

每个功能都以其自己的方式简化Spring应用程序的开发。我们将在整本书中逐步运用他们。 但现在，让我们快速浏览一下他们各提供了什么。

自动配置

In any given Spring application’s source code, you’ll find either Java configuration or XML configuration (or both) that enables certain supporting features and functionality for the application. For example, if you’ve ever written an application that accesses a relational database with JDBC , you’ve probably configured Spring’s JdbcTemplate as a bean in the Spring application context. I’ll bet the configuration looked a lot like this:

```
@Bean
public JdbcTemplate jdbcTemplate(DataSource dataSource) {
       return new JdbcTemplate(dataSource);
  }
```

This very simple bean declaration creates an instance of JdbcTemplate , injecting it with its one dependency, a DataSource . Of course, that means that you’ll also need to configure a DataSource bean so that the dependency will be met. To complete this configuration scenario, suppose that you were to configure an embedded H2 database as the DataSource bean:

```
@Bean
public DataSource dataSource() {
  return new EmbeddedDatabaseBuilder()
            .setType(EmbeddedDatabaseType.H2)
            .addScripts('schema.sql', 'data.sql')
            .build();
}
```

This bean configuration method creates an embedded database, specifying two SQL scripts to execute on the embedded database. The build() method returns a Data- Source that references the embedded database.   Neither of these two bean configuration methods is terribly complex or lengthy. But they represent just a fraction of the configuration in a typical Spring application. Moreover, there are countless Spring applications that will have these exact same methods. Any application that needs an embedded database and a JdbcTemplate will need those methods. In short, it’s boilerplate configuration.

If it’s so common, then why should you have to write it? Spring Boot can automatically configure these common configuration scenarios. If Spring Boot detects that you have the H2 database library in your application’s class- path, it will automatically configure an embedded H2 database. If JdbcTemplate is in the classpath, then it will also configure a JdbcTemplate bean for you. There’s no need for you to worry about configuring those beans. They’ll be configured for you, ready to inject into any of the beans you write.

There’s a lot more to Spring Boot auto-configuration than embedded databases and JdbcTemplate . There are several dozen ways that Spring Boot can take the bur- den of configuration off your hands, including auto-configuration for the Java Persis- tence API (JPA), Thymeleaf templates, security, and Spring MVC . We’ll dive into auto- configuration starting in chapter 2.

启动依赖

It can be challenging to add dependencies to a project’s build. What library do you need? What are its group and artifact? Which version do you need? Will that version play well with other dependencies in the same project?

Spring Boot offers help with project dependency management by way of starter dependencies. Starter dependencies are really just special Maven (and Gradle) depen- dencies that take advantage of transitive dependency resolution to aggregate com- monly used libraries under a handful of feature-defined dependencies.

For example, suppose that you’re going to build a REST API with Spring MVC that works with JSON resource representations. Additionally, you want to apply declarative validation per the JSR-303 specification and serve the application using an embedded Tomcat server. To accomplish all of this, you’ll need (at minimum) the following eight dependencies in your Maven or Gradle build:

```
* org.springframework:spring-core
* org.springframework:spring-web
* org.springframework:spring-webmvc
* com.fasterxml.jackson.core:jackson-databind
* org.hibernate:hibernate-validator
* org.apache.tomcat.embed:tomcat-embed-core
* org.apache.tomcat.embed:tomcat-embed-el
* org.apache.tomcat.embed:tomcat-embed-logging-juli
```

On the other hand, if you were to take advantage of Spring Boot starter dependencies, you could simply add the Spring Boot “web” starter (org.springframework.boot:spring-boot-starter-web) as a build dependency. This single dependency will transitively pull in all of those other dependencies so you don’t have to ask for them all.

But there’s something more subtle about starter dependencies than simply reduc- ing build dependency count. Notice that by adding the “web” starter to your build, you’re specifying a type of functionality that your application needs. Your app is a web application, so you add the “web” starter. Likewise, if your application will use JPA per- sistence, then you can add the “jpa” starter. If it needs security, you can add the “secu- rity” starter. In short, you no longer need to think about what libraries you’ll need to support certain functionality; you simply ask for that functionality by way of the perti- nent starter dependency.

Also note that Spring Boot’s starter dependencies free you from worrying about which versions of these libraries you need. The versions of the libraries that the start- ers pull in have been tested together so that you can be confident that there will be no incompatibilities between them.

Along with auto-configuration, we’ll begin using starter dependencies right away, starting in chapter 2.

命令行接口

In addition to auto-configuration and starter dependencies, Spring Boot also offers an intriguing new way to quickly write Spring applications. As you saw earlier in section 1.1, the Spring Boot CLI makes it possible to write applications by doing more than writing the application code.

Spring Boot’s CLI leverages starter dependencies and auto-configuration to let you focus on writing code. Not only that, did you notice that there are no import lines in list- ing 1.1? How did the CLI know what packages RequestMapping and RestController come from? For that matter, how did those classes end up in the classpath?

The short answer is that the CLI detected that those types are being used, and it knows which starter dependencies to add to the classpath to make it work. Once those dependencies are in the classpath, a series of auto-configuration kicks in and ensures that DispatcherServlet and Spring MVC are enabled so that the controller can respond to HTTP requests.

Spring Boot’s CLI is an optional piece of Spring Boot’s power. Although it provides tremendous power and simplicity for Spring development, it also introduces a rather unconventional development model. If this development model is too extreme for your taste, then no problem. You can still take advantage of everything else that Spring Boot has to offer even if you don’t use the CLI . But if you like what the CLI pro- vides, you’ll definitely want to look at chapter 5 where we’ll dig deeper into Spring Boot’s CLI .

执行器

The final piece of the Spring Boot puzzle is the Actuator. Where the other parts of Spring Boot simplify Spring development, the Actuator instead offers the ability to inspect the internals of your application at runtime. With the Actuator installed, you can inspect the inner workings of your application, including details such as

- What beans have been configured in the Spring application context
- What decisions were made by Spring Boot’s auto-configuration
- What environment variables, system properties, configuration properties, and
- command-line arguments are available to your application
- The current state of the threads in and supporting your application A trace of recent HTTP requests handled by your application
- Various metrics pertaining to memory usage, garbage collection, web requests,and data source usage

The Actuator exposes this information in two ways: via web endpoints or via a shell interface. In the latter case, you can actually open a secure shell (SSH) into your application and issue commands to inspect your application as it runs.

We’ll explore the Actuator’s capabilities in detail when we get to chapter 7.

１．１．３　Ｓｐｒｉｎｇ Boot不能做什么

由于使用ＳｐｒｉｎｇＢｏｏｔ有许多令人惊奇的地方，在过去一年左右有很多关于其的讨论。在阅读这本书之前，不管你听到或者认为了什么，你可能对ＳｐｒｉｎｇBoot有些许误解，现在你需要清除这些误解，之后再继续接着ＳｐｒｉｎgBoot之旅。

首先，ＳｐｒｉｎｇBoot不是一个应用服务器。这个误解来自于它可以创建Ｗｅｂ应用程序以自执行的方式通过命令行运行起来，不需要部署到传统的Ｊａｖａ应用服务器上。ＳｐｒｉｎｇＢｏｏｔ完成这些应用程序是使用了内嵌的Ｓｅｒｖｌｅｔ容器，比如Ｔｏｍｃａｔ，Ｊｅｔｔｙ或者Ｕｎｄｅｒｔｏｗ等。但是这些内嵌的ｓｅｒｖｌｅｔ容器提供了应用服务器的功能，可这并不是ＳｐｒｉｎｇＢｏｏｔ本身。　　同样，ＳｐｒｉｎｇＢｏｏｔ也没有实现任何Ｊａｖａ企业级规范，比如ＪＰＡ，ＪＭＳ等。只是说它具有可以自动配置Ｂｅａｎｓ到Ｓｐｒｉｎｇ中来支持多种企业级规范的特点。举个例子，ＳｐｒｉｎｇＢｏｏｔ并没有实现ＪＰＡ规范，但是它支持为ＪＰＡ的实现如Hibernate实现自动配置合适的Ｂｅａｎｓ。

最后，SpringBoot并不是通过任何形式的代码生成来完成它的魔法的。而是，通过Ｓｐｒｉｎｇ４配置来影响它的，并通过Maven或者Ｇｒａｄｌｅ解决传递依赖和以及通过自动配置Ｂｅａｎ在Ｓｐｒｉｎｇ应用程序环境中来实现它的魔法的。

简言之，ＳｐｒｉｎｇＢｏｏｔ的心脏其实就是Ｓｐｒｉｎｇ。在内部，ＳｐｒｉｎｇＢｏｏｔ做着和Ｓｐｒｉｎｇ配置Ｂｅａｎ一样的事情，就算ＳｐｒｉｎｇＢｏｏｔ不存在，你也可以在Ｓｐｒｉｎｇ中继续执行你自己的东西。幸亏，ＳｐｒｉｎｇＢｏｏｔ是存在的，不然你又将会陷入显式样板化的配置中去，使用ＳｐｒｉｎｇＢｏｏｔ使得你只要关心逻辑即可了，这让你的应用程序变得更独特了。

目前为止，你应该对ＳｐｒｉｎｇＢｏｏｔ有了大概的映像了，它也该上台面了。构建你的第一个ＳｐｒｉｎｇＢｏｏｔ项目只是时间问题了，那么接下来让我们迈出ＳｐｒｉｎｇＢｏｏｔ的第一步吧。

1.2 Ｓｐｒｉｎｇ Boot入门

总之，一个ＳｐｒｉｎｇＢoot项目其实就是一个常规的Ｓｐｒｉｎｇ项目，只不过是发生在自动配置和启动依赖影响下的Ｓｐｒｉｎｇ项目了。因此，任何你熟悉的应用在Ｓｐｒｉｎｇ项目的工具或技术都可以应用在Ｓｐｒｉｎｇ Boot项目中。并且还有一些十分便利的地方促使你创建你的Ｓｐｒｉｎｇ　Ｂｏｏｔ项目。

最快捷的方式开始一个SpringBoot项目就是安装ＳｐｒｉｎｇBoot CLI,以至于你可以开始编写代码清单１－１的代码，然后通过ＣＬＩ运行它。

１·２·１ 安装Ｓｐｒｉｎｇ Boot CLI

正如我们之前讨论过的，Spring Boot CLI为我们提供了一个有趣，又不传统的方式来进行Spring应用的开发。我们会在第五章详细分析CLI到底为我们提供了什么。但是现在，让我们先来看看如何安装Spring Boot CLI，从而可以运行我们在代码清单1-1中所看到的代码。

安装Spring Boot CLI有多种方式：

- 使用distribution版本安装
- 使用SDKManager安装
- 使用ＨｏｍｅBrew进行安装
- 使用ＭａｃＰｏｒｔｓ安装

这些安装方式我们一个个来看。此外，我们也会手动的彻底安装Spring Boot CLI,通过Bash或zsh等shell脚本。（这里要对Windows使用者说抱歉了，因为BashShell只有Linux系统支持。）那么，首先让我们来看看如何通过Spring以发布的版本来进行安装。

Spring Boot CLI 安装手册(使用distribution版本安装)

或许安装Spring Boot CLI最直接的办法就是下载其压缩包，然后解压，将其bin目录添加到PATH中（环境变量中）。你可以从以下链接下载已发布的文件:

```
* http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/
1.3.0.RELEASE /spring-boot-cli- 1.3.0.RELEASE -bin.zip
* http://repo.spring.io/release/org/springframework/boot/spring-boot-cli/
1.3.0.RELEASE /spring-boot-cli- 1.3.0.RELEASE -bin.tar.gz
```

只要你下载已经发布的版本，将其解压到你的文件系统中，在当中你可以找到包含spring命令脚本的bin目录。bin目录既包含windows下的脚本，也包含Linux系统下的脚本。然后将bin目录添加到系统的PATH中（环境变量中），即可以准备运行Spring Boot CLI了。

- 创建Spring Boot的软链接　如果你使用SpringBoot在类Unix系统上，你可以创建一个为解压缩的文件链接，并把链接的目录添加到系统的PATH配置中，而不使用直接目录。通过修改软连接，这样就可以很好的更新SpringBooter的版本，甚至可以灵活的切换版本，你可以在CLI下通过试试如下命令来验证版本。

  ```
  $ spring --version
  ```

  如果一切工作正常，你将会看到显式出已安装的Spring Boot CLI版本提示。

  即使这是一次手工安装，它也是一种很好的选择，它也不需要你额外在安装什么。如果你是Window用户，这也是唯一提供给你的安装方式了。但如果你是Linux机器，就可以做更多的自动化配置了，要想这样，SDKManager可能会很有帮助。

安装 SDKManager（使用SDKManager安装）
The Software Development Kit Manager (SDKMAN ; formerly known as GVM) can be used to install and manage multiple versions of Spring Boot CLI installations. In order to use SDKMAN , you'll need to get and install the SDKMAN tool from http://sdkman .io. The easiest way to install SDKMAN is at the command line:

```
$ curl -s get.sdkman.io | bash
```

Follow the instructions given in the output to complete the SDKMAN installation. For my machine, I had to perform the following command at the command line:

```
$ source "/Users/habuma/.sdkman/bin/sdkman-init.sh"
```

Note that this command will be different for different users. In my case, my home directory is at /Users/habuma , so that's the root of the shell script's path. You'll want to adjust accordingly to fit your situation. Once SDKMAN is installed, you can install Spring Boot's CLI like this:

```
  $ sdk install springboot
  $ spring --version
```

Assuming all goes well, you'll be shown the current version of Spring Boot.
If you want to upgrade to a newer version of Spring Boot CLI , you just need to install it and start using it. To find out which versions of Spring Boot CLI are available, use SDKMAN 's list command:

```
$ sdk list springboot
```

The list command shows all available versions, including which versions are installed and which is currently in use. From this list you can choose to install a version and then use it. For example, to install Spring Boot CLI version 1.3.0.RELEASE , you'd use the install command, specifying the version:

```
$ sdk install springboot 1.3.0.RELEASE
```

If you'd like that version to be the default for all shells, use the default command:

```
$ sdk default springboot 1.3.0.RELEASE
```

The nice thing about using SDKMAN to manage your Spring Boot CLI installation is that it allows you to easily switch between different versions of Spring Boot. This will enable you to try out snapshot, milestone, and release candidate builds before they're formally released, but still switch back to a stable release for other work.

使用ＨｏｍｅBrew进行安装

If you'll be developing on an OS X machine, you have the option of using Homebrew to install the Spring Boot CLI . Homebrew is a package manager for OS X that is used to install many different applications and tools. The easiest way to install Homebrew is by running the installation Ruby script:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
master/install)"
```

You can read more about Homebrew (and find other installation options) at http:// brew.sh. In order to install the Spring Boot CLI using Homebrew, you'll need to "tap" Piv- otal's tap: 1

```
$ brew tap pivotal/tap
```

Now that Homebrew is tapping Pivotal's tap, you can install the Spring Boot CLI like this:

```
$ brew install springboot
```

Homebrew will install the Spring Boot CLI to /usr/local/bin, and it's ready to go. You can verify the installation by checking the version that was installed:

```
$ spring --version
```

It should respond by showing you the version of Spring Boot that was installed. You can also try running the code in listing 1.1.

1 Tapping is a way to add additional repositories to those that Homebrew works from. Pivotal, the company behind Spring and Spring Boot, has made the Spring Boot CLI available through its tap.

使用ＭａｃＰｏｒｔｓ安装

Another Spring Boot CLI installation option for OS X users is to use MacPorts, another popular installer for Mac OS X . In order to use MacPorts to install the Spring Boot CLI , you must first install MacPorts, which itself requires that you have Xcode installed. Furthermore, the steps for installing MacPorts vary depending on which ver- sion of OS X you're using. Therefore, I refer you to https://www.macports.org/ install.php for instructions on installing

MacPorts.

Once you have MacPorts installed, you can install the Spring Boot CLI at the command line like this:

```
$ sudo port install spring-boot-cli
```

MacPorts will install the Spring Boot CLI to /opt/local/share/java/spring-boot-cli and put a symbolic link to the binary in /opt/local/bin, which should already be in your system path from installing MacPorts. You can verify the installation by checking the version that was installed:

```
$ spring --version
```

It should respond by showing you the version of Spring Boot that was installed. You can also try running the code in listing 1.1.

ENABLING COMMAND-LINE COMPLETION Spring Boot's CLI offers a handful of commands for running, packaging, and testing your CLI -based application. Moreover, each of those commands has several options. It can be difficult to remember all that the CLI offers. Command-line completion can help you recall how to use the Spring Boot CLI .

If you've installed the Spring Boot CLI with Homebrew, you already have command- line completion installed. But if you installed Spring Boot manually or with SDKMAN , you'll need to source the scripts or install the completion scripts manually. (Command- line completion isn't an option if you've installed the Spring Boot CLI via MacPorts.)

The completion scripts are found in the Spring Boot CLI installation directory under the shell-completion subdirectory. There are two different scripts, one for BASH and one for zsh. To source the completion script for BASH , you can enter the following at the command line (assuming a SDKMAN installation):

```
$ . ~/.sdkman/springboot/current/shell-completion/bash/spring
```

This will give you Spring Boot CLI completion for the current shell, but you'll have to source this script again each time you start a new shell to keep that feature. Option- ally, you can copy the script to your personal or system script directory. The location of the script directory varies for different Unix installations, so consult your system docu- mentation (or Google) for details.

With command completion enabled, you should be able to type spring at the command line and then hit the Tab key to be offered options for what to type next. Once you've chosen a command, type -- (double-hyphen) and then hit Tab again to be shown a list of options for that command.

If you’re developing on Windows or aren’t using BASH or zsh, you can’t use these command-line completion scripts. Even so, you can get command completion if you run the Spring Boot CLI shell:

```
$ spring shell
```

Unlike the command-completion scripts for BASH and zsh (which operate within the BASH /zsh shell), the Spring Boot CLI shell opens a new Spring Boot–specific shell. From this shell, you can execute any of the CLI ’s commands and get command com- pletion with the Tab key.

The Spring Boot CLI offers an easy way to get started with Spring Boot and to prototype simple applications. As we’ll discuss later in chapter 8, it can also be used for production-ready applications, given the right production runtime environment.

Even so, Spring Boot CLI ’s process is rather unconventional in contrast to how most Java projects are developed. Typically, Java projects use tools like Gradle or Maven to build WAR files that are deployed to an application server. If the CLI model feels a little uncomfortable, you can still take advantage of most of the features of Spring Boot in the context of a traditionally built Java project. 2 And the Spring Initializr can help you get started.

1.2.2 使用Ｓｐｒｉｎｇ Initializr初始化ＳｐｒｉｎｇBoot项目

有时候最困难的部分是项目的初始阶段。你需要为各种项目文件创建目录结构，创建项目的构建文件以及根据构建文件确定依赖等。ＳｐｒｉｎｇＢｏｏｔＣＬＩ可以帮你去除这么多的初始工作，但如果你仍喜欢更传统的Ｊａｖａ项目结构，那你一定会看上Ｓｐｒｉｎｇ Initializr的。

Ｓｐｒｉｎｇ Initializr本质上是一个Ｗｅｂ应用程序，它可以为你生成一个ＳｐｒｉｎｇＢｏｏｔ项目结构。它不会生成任何应用程序代码，但是它会给你提供基本的项目结构，也会使用Ｍａｖｅｎ或者Ｇｒａｄｌｅ构建你的项目代码。你的全部就是只需要写应用程序代码即可。

Spring Initializr可以通过多种方式使用：

- 通过Ｗｅｂ端接口的界面
- 通过Spring Tool Suite
- 通过IntelliJ IDEA
- 使用Spring Boot CLI

接下来我们会一一使用这些接口来初始化，首先让我们来看看通过ｗｅｂ端如何来实现。

使用**SPRING INITIALIZR’S** Ｗｅｂ端接口界面

最直接的方式使用Spring Initializr是通过Ｗｅｂ浏览器访问http://start.spring.io, 你可以通过图１－１看到熟悉的界面。

首先要做的两件事是选择你的项目构建使用Ｍａｖｅｎ还是Gradle，还有就是选择ＳｐｒｉｎｇＢｏｏｔ的版本。默认使用的构建项目方式是Ｍａｖｅｎ，会使用最新的发行版本，不会选择里程碑版或者快照版的ＳｐｒｉｎｇＢｏｏｔ，当然也欢迎你选择不同的版本。

在左边的表单中，要求你明确项目的一些元数据。至少，你必须提供项目的ｇｒｏｕｐ和artifact。但是如果点“Switch to　the full version”的链接，你可以明确项目其他的一些元数据，比如版本，基本包名等。这些元数据的输入被用来生成Ｍａｖｅｎ的pom.xml文件或者Ｇｒａｄｌｅ的build.gradle文件。

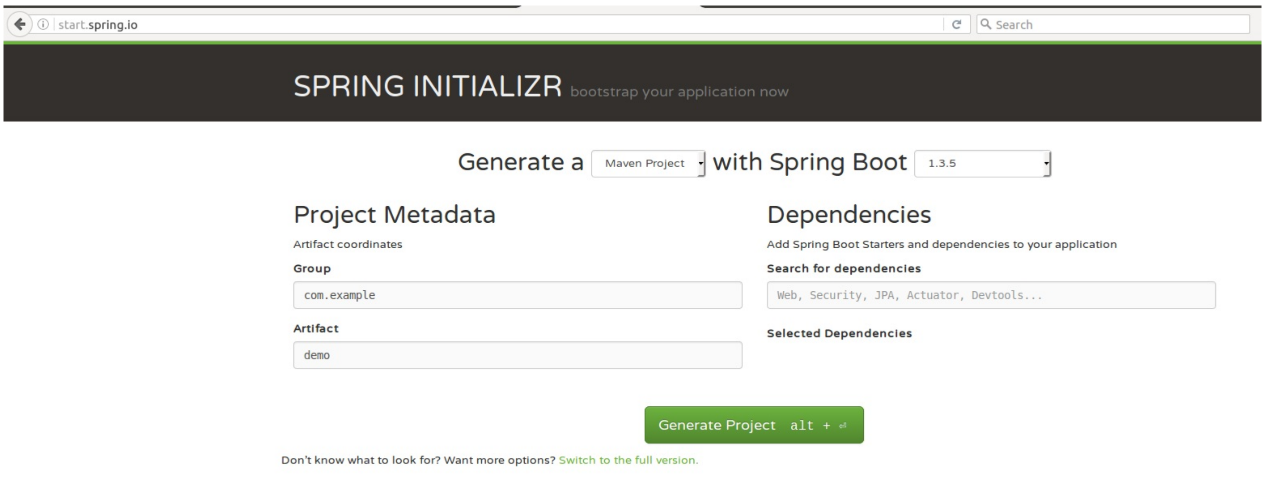

start.spring.io
Search
SPRING INITIALIZR bootstrap your application now
Generate a Maven Project with Spring Boot 1.3.5
Project Metadata
Artifact coordinates
Group
com.example
Artifact
demo
Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies
Web, Security, JPA, Actuator, Devtools...
Selected Dependencies
Generate Project alt + ⏎
Don't know what to look for? Want more options? Switch to the full version.

图１－１　Spring Initializr是一个ｗｅｂ应用程序用来生成空的Ｓｐｒｉｎｇ项目作为最初开发时使用

在右边的表单中，要求你填入项目的依赖。最简单的方式就是在ｔｅｘｔ表单中输入依赖的类型。根据输入的类型，会出现一个与之相匹配的列表，选择你所需要的，它将会被添加到项目中去。如果你没有找到你所需要的，点击“Switch to the　full version”的链接，你可以得到完整的所提供的依赖列表。

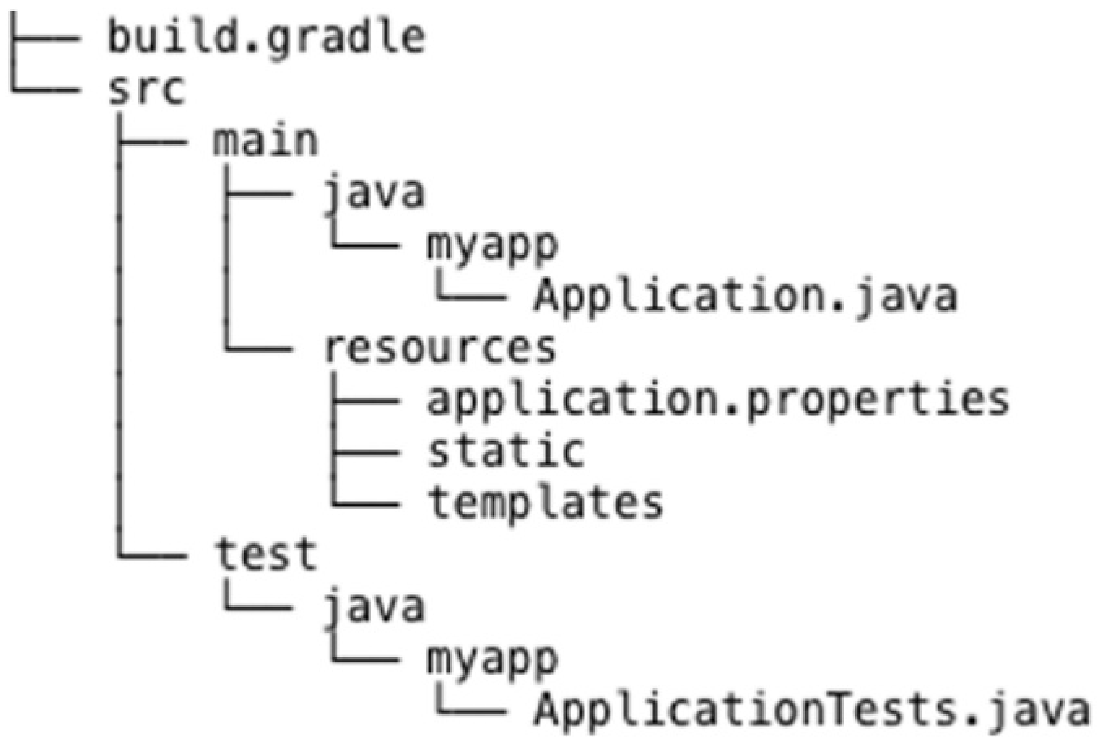
如果你看过附录Ｂ,你可以识别出相应的ＳｐｒｉｎｇＢｏｏｔ所提供的依赖。事实上，通过选择任意的这些依赖，会告诉Initializr添加依赖到构建文件中作为启动使用。（我们会在第二章更多的讨论启动依赖）。

只要你填写好表单里的内容并且选择好了依赖，点击生成项目按钮，Spring Initializr会为你生成一个项目。生成的项目会打成ｚｉｐ包，名字便是你填写的Ａｒｔｉｆａｃｔ，通过浏览器可以将其下载下来。ｚｉｐ包的内容可能会有写不同，这取决于你之前生成项目的选择。无论何种方式，ｚｉｐ包都包含了最基本的项目，之后你就可以使用ＳｐｒｉｎｇBoot为你创建的项目进行开发了。

举个例子，假设你通过Spring Initializr指定了如下初始化信息：

```
* Artifact: myapp
* Package Name: myapp
* Type: Gradle Project
* Dependencies: Web and JPA
```

在点击ＧｅｎｅｒａｔｅＰｒｏｊｅｃｔ后，你会得到一个名字为myapp.zｉｐ的文件，解压它后，你可以看到如图１－２所示的熟悉的项目结构。



```
├── build.gradle
└── src
    ├── main
    │   ├── java
    │   │   └── myapp
    │   │       └── Application.java
    │   └── resources
    │       ├── application.properties
    │       ├── static
    │       └── templates
    └── test
        └── java
            └── myapp
                └── ApplicationTests.java
```

图１.2　Initializr-创建的一个最小化的基础项目可以用来构建SpringＢｏｏｔ的应用程序

正如你所见，项目中只有很少的代码，大多是一堆空的目录，包括了:

- build.gradle/pom.xml—Ｇｒａｄｌｅ构建项目的叙述文件。如果你选择的是Ｍａｖｅｎ则会是一个pom.xml文件。
- Application.java—一个具有ｍａｉｎ函数的程序，可以让从此开始项目的运行。
- ApplicationTests.java—一个空的Ｊｕｎｉｔ测试类，用来测试并加载Ｓｐｒｉｎｇ应用上下文通过ＳｐｒｉｎｇＢｏｏｔ的自动配置来实现。
- application.properties—一个空的配置文件，为你可以添加合适的配置做准备了。

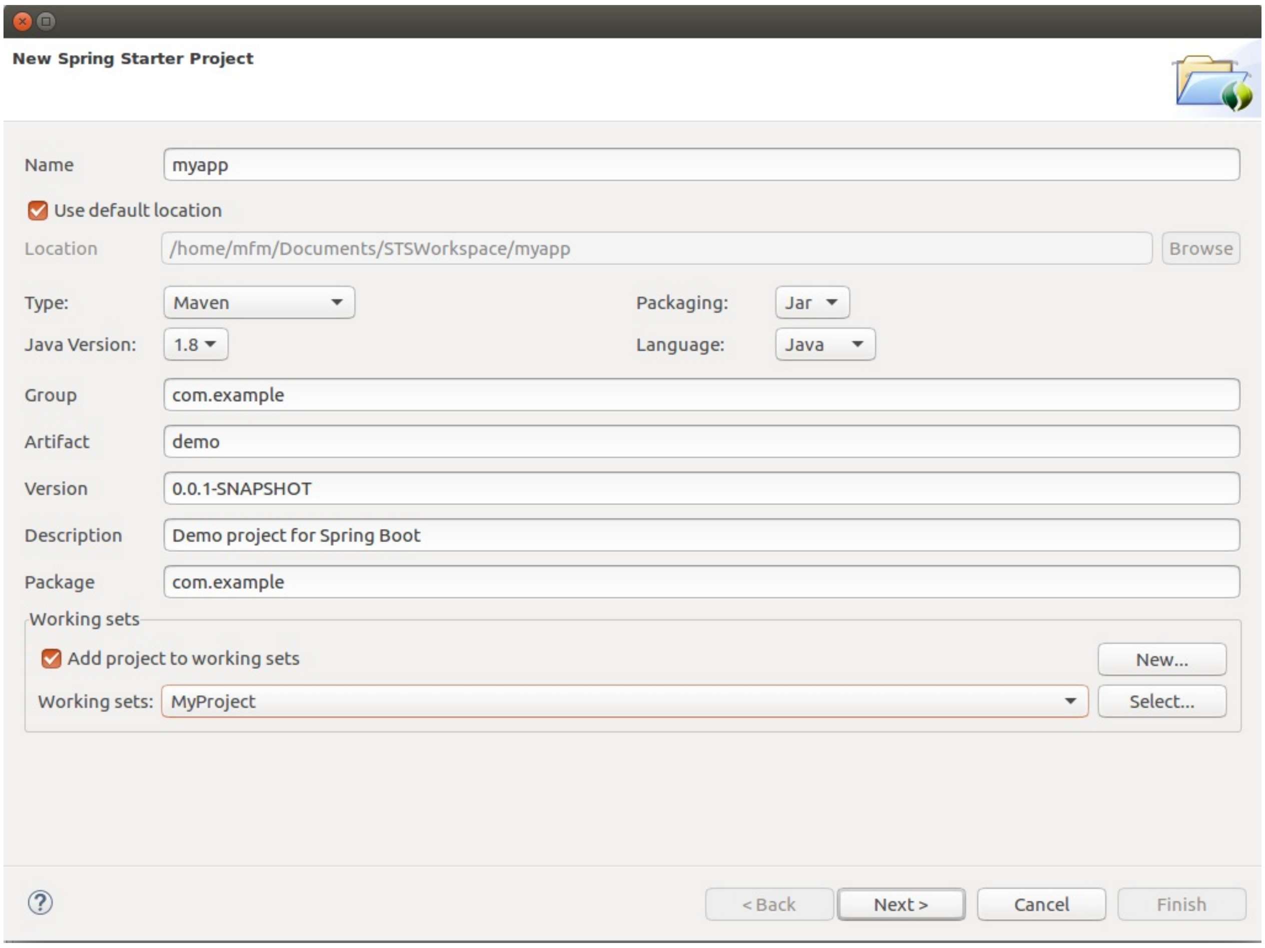
这些空的目录在ＳｐｒｉｎｇＢｏｏｔ应用中有重要的作用。static目录可以将ｗｅｂ应用程序中的一些静态文件如ＪａｖａＳｃｒｉｐｔ，ｓｔｙｌｅｓｈｅｅｔｓ,ｉｍａｇｅｓ等等放入其中管理，至于ｔｅｍｐｌａｔｅｓ目录，之后你可以看到，你可以讲需要渲染的模型数据放到这个目录下。

你也可能会选择通过ＩＤＥ来Initializr创建你的项目。如果Spring Tool Suite是你选择的ＩＤＥ，则你可以在ＩＤＥ中直接创建项目了。接下来让我们看一看Spring Tool Suite（ＳＴＳ)对创建ＳｐｒｉｎｇＢｏｏｔ的支持吧。

使用**SPRING TOOL SUITE**（ＳＴ**S**）创建Ｓｐｒｉｎｇ Ｂｏｏｔ项目

Spring Tool Suite 3 是开发Ｓｐｒｉｎｇ应用程序非常好的ＩＤＥ。自从３・４・０版本之后，ＳＴＳ已经集成了Spring Initializr的功能，使得开始一个ＳｐｒｉｎｇＢｏｏｔ项目变得十分简单了。

要创建ＳｐｒｉｎｇＢｏｏｔ应用程序，首先右键点击选择the New >,之后点击Spring Starter Project的选项，这样之后你会开到如图１－３所示的对话框出现。


New Spring Starter Project

Name: myapp

☑ Use default location

Location: /home/mfm/Documents/STSWorkspace/myapp

Type: Maven

Packaging: Jar

Java Version: 1.8

Language: Java

Group: com.example

Artifact: demo

Version: 0.0.1-SNAPSHOT

Description: Demo project for Spring Boot

Package: com.example

Working sets

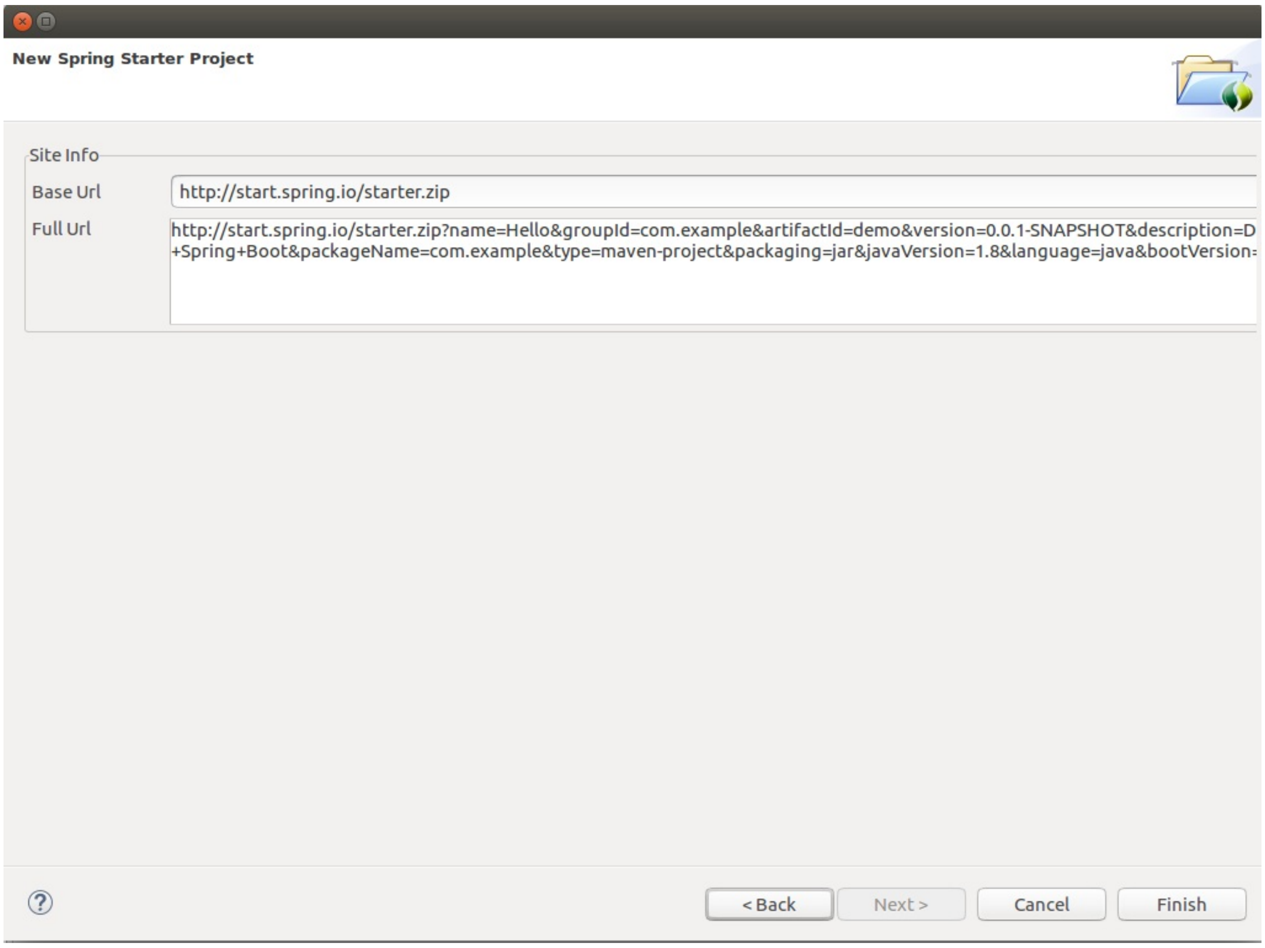
☑ Add project to working sets

Working sets: MyProject

图１－３　Spring　Tool Suite　ＩＤＥ通过与Spring Initializr集成来创建Ｓｐｒｉｎｇ Ｂｏｏｔ项目

正如你所看到的，这个对话框中具有和ｗｅｂ端页面一样的信息。事实上，这些填入的数据会和在ｗｅｂ端页面一样提供给Spring Initializr来创建项目。

你可以指定你的项目创建在文件系统的何处，也可以将项目添加到制定的工作集中。点击下一步之后，你会看到如图１－４所示的对话框提示。



Base Url: http://start.spring.io/starter.zip

Full Url: http://start.spring.io/starter.zip?name=Hello&groupId=com.example&artifactId=demo&version=0.0.1-SNAPSHOT&description=D
+Spring+Boot&packageName=com.example&type=maven-project&packaging=jar&javaVersion=1.8&language=java&bootVersion=

图１－４　 the Spring　Starter Project创建的对话花开，你可以明确指定的项目的创建

Ｌｏｃａｌｔｉｏｎ会指定你项目存放的位置，如果你需要ｅｃｌｉｐｓｅ的工作集来组织管理你项目，通过ｓｅｌｅｃｔ选择框，你可以添加项目到制定的工作集中。

这个位置信息简单的描述了将要连接Initializr的ＵＲＬ,大多情况下，你可以忽略这一个地方，当然，如果你要部署你你自定义　的tializr服务（比如使用ｇｉｔｈｕｂ克隆出的https://github.com/spring-io/initializr），你则需要将基本的ＵＲＬ填写在此处。

点击ｆｉｎｉｓｈ按钮后，会进入项目生成的和导入的进程中。重要的是要知道ＳＴＳ的ＳｔａｒｔｅｒＰｒｏｊｅｃｔ的创建项目也就是http://start.spring.io　中的Spring Initializr的创建项目，所以你必须联网才能确保其正确工作。

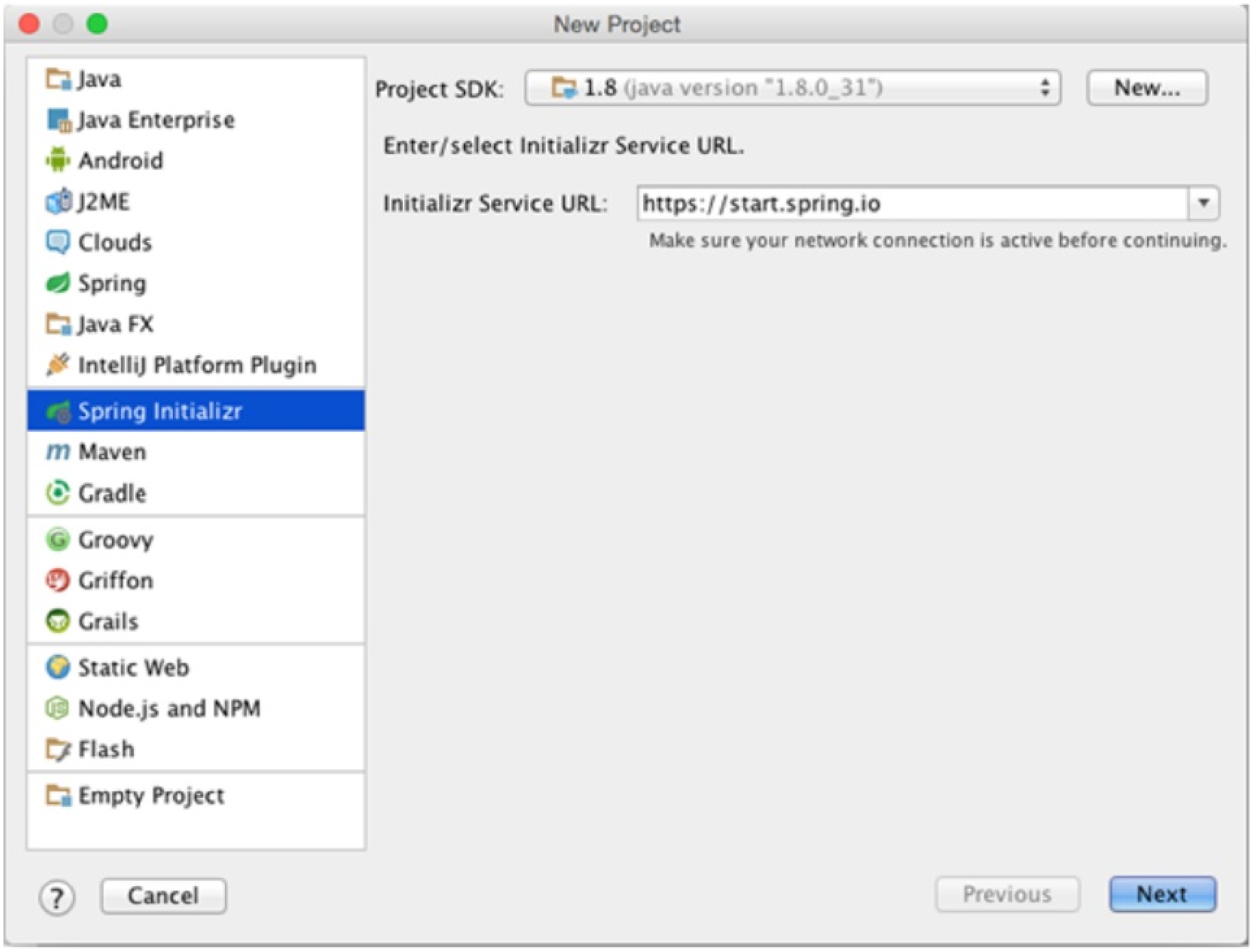
当项目导入到工作空间后，你可以准备启动你的应用程序了。当你开发的你的应用程序的时候，你会发现ＳＴＳ将ＳｐｒｉｎｇＢｏｏｔ一些地方封装起来，举个例子，你可以在项目上点击右键，选择Ｒｕｎ菜单中的Ｒｕｎ　Ａｓ > Spring Boot，你就可以使用内嵌的服务来运行你的应用程序了。

需要重视理解的是ＳＴＳ通过ＲＥＳＴＡＰＩ来实现与Initializr的协调工作。因此，当其工作的时候会连接Initializr，如果你的用来开发机器没有联网，或者Initializr被防火墙挡住，你使用ＳＴＳ创建ＳｐｒｉｎｇＳｔａｒｔｅｒＰｒｏｊｅｃｔ向导也就不会成功了。

使用**INTELLIJ IDEA**创建ＳｐｒｉｎｇＢｏｏｔ项目

IntelliJ IDEA是一个非常受欢迎的ＩＤＥ，我们来看看IntelliJ IDEA 14.1是如何支持ＳｐｒｉｎｇＢｏｏｔ的。

使用IntelliJ IDEA开始一个ＳｐｒｉｎｇＢｏｏｔ项目首先第一步就是选择New > Project，然后在左边的列表中选择Ｓｐｒｉｎｇ　Ｉｎｉｔｉａｌｉｚｒ，如图１－５所示，接下来要求输入的内容就和ＳＴＳ的Initializｒ初始化Web应用程序类似了。



Project SDK: 1.8 (java version "1.8.0_31")

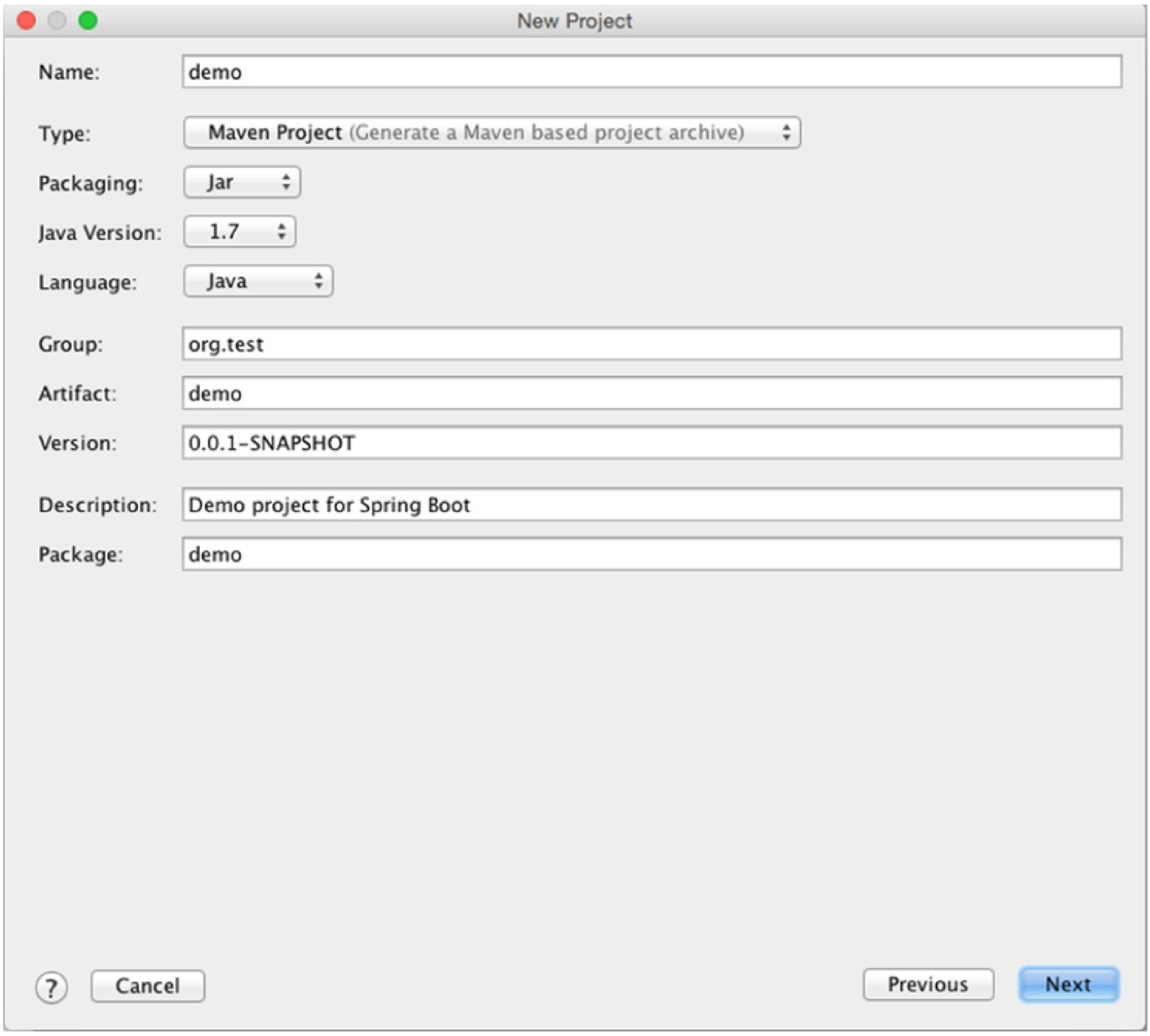
Enter/select Initializr Service URL.

Initializr Service URL: https://start.spring.io

Make sure your network connection is active before continuing.

图１－５　IntelliJ IDEA's Spring Boot initialization 向导第一屏

在初始化的窗口中，选择了左边的Spring Initializr之后，你会选择项目的ＳＤＫ版本，选择Initializr web service的地址，如果你要运行你自己的Initializr实例的化，则需要做这个选择，否则就不用做任何改变，直接下一步。然后，你会看到如图１－６所示的提示内容：



Name: demo
Type: Maven Project (Generate a Maven based project archive)
Packaging: Jar
Java Version: 1.7
Language: Java
Group: org.test
Artifact: demo
Version: 0.0.1-SNAPSHOT
Description: Demo project for Spring Boot
Package: demo

图１－６　IntelliJ IDEA’s Spring Boot initialization 向导之填写项目基本信息

通过IntelliJ IDEA填写ＳｐｒｉｎｇＢｏｏｔ初始化向导要求的项目基本信息，比如项目名称，ｇｒｏｕｐＩｄ，ａｒｔｉｆａｃｔＩｄ，Ｊａｖａ的版本，构建方式使用Ｇｒａｄｌｅ还是Ｍａｖｅｎ等描述后，点击下一步，则会看到如图１－７所示的界面。

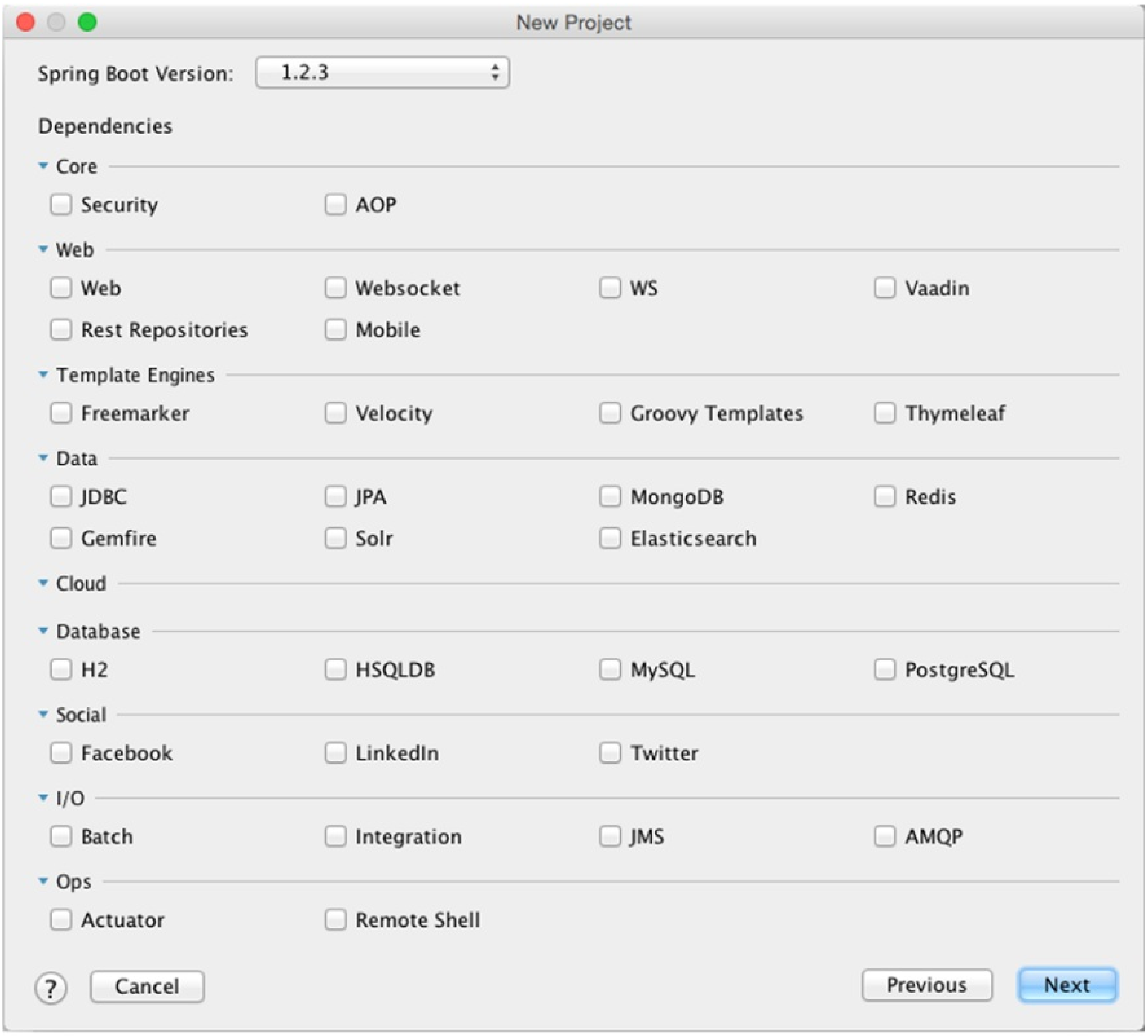
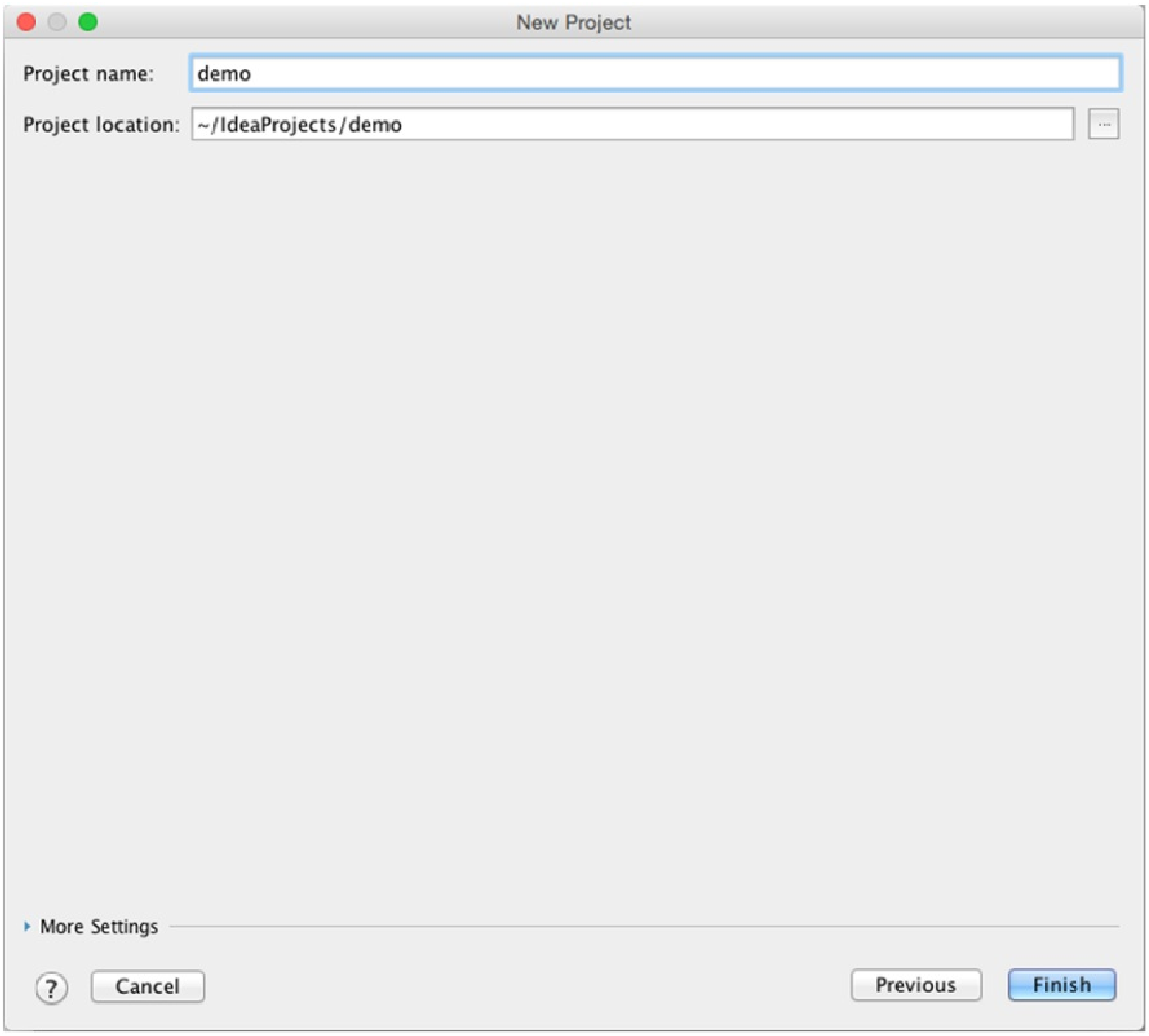


图１－７　 IntelliJ IDEA’s Spring Boot initialization 向导选择项目依赖

之前我们填写了项目的基本信息后,在这里我们要求将项目的依赖关系指定。正如ＳＴＳ创建ＳｐｒｉｎｇＳｔａｒｔｅｒ看到的依赖一样，你做出　选择后，点击下一步　，就会进入了最后的如图１－８所示界面了。


New Project

Project name: demo

Project location: ~/IdeaProjects/demo

图１－８　IntelliJ IDEA’s Spring Boot initialization 向导最后一屏

最后的设置很简单，告诉IntelliJ IDEA的名字以及项目保存的路径，之后点击Ｆｉｎｉｓｈ即可完成创建。这样你就使用这个ＩＤＥ创建了最基础的ＳｐｒｉｎｇＢｏｏｔ项目了。

使用ＳｐｒｉｎｇＢｏｏｔＣＬＩ中的**INITIALZR**创建项目　　正如你之前所看到的，ＳｐｒｉｎｇBoot CLI强大的方式开发Ｓｐｒｉｎｇ应用程序—只需要专注于写代码即可。然而，Ｓｐｒｉｎｇ Boot CLI也有一些命令来帮助你在传统Ｊａｖａ项目中，初始化和快速启动开发。

ＳｐｒｉｎｇＢｏｏｔ ＣＬＩ通过init的命令来实现和Initializr客户端接口一样的动作。以下命令就是最简单使用ｉｎｉｔ的方式创建基础的ＳｐｒｉｎｇBoot项目：

```
$ spring init
```

通过连接Initializr Ｗｅｂ应用程序，init命令会下载ｄｅｍｏ.zip文件，如果你解压这个项目，你会发现使用Ｍａｖｅｎ构建的典型的项目结构。Ｍａｖｅｎ构建的规格是最小化的，仅仅包括了ＳｐｒｉｎｇBoot启动和测试的依赖，以后你可以适当的逐步添加。

如果你希望构建一个ｗｅｂ应用程序，并依赖于ＪＰＡ和ＳｐｒｉｎｇSecurity，则可以通过－ｄ或则--dependencies来指名依赖。

```
$ spring init -dweb,jpa,security
```

By default, the build specification for both Maven and Gradle builds will produce an executable JAR file. If you'd rather produce a WAR file, you can specify so with the --packaging or -p parameter:

```
$ spring init -dweb,jpa,security --build gradle -p war
```

So far, the ways we've used the init command have resulted in a zip file being down- loaded. If you'd like for the CLI to crack open that zip file for you, you can specify a directory for the project to be extracted to:

```
$ spring init -dweb,jpa,security --build gradle -p war myapp
```

The last parameter given here indicates that you want the project to be extracted to the myapp directory.
Optionally, if you want the CLI to extract the generated project into the current directory, you can use either the --extract or the -x parameter:

```
$ spring init -dweb,jpa,security --build gradle -p jar -x
```

The init command has several other parameters, including parameters for building a Groovy-based project, specifying the Java version to compile with, and selecting a ver- sion of Spring Boot to build against. You can discover all of the parameters by using the help command:

```
$ spring help init
```

You can also find out what choices are available for those parameters by using the --list or -l parameter with the init command:

```
$ spring init -l
```

You'll notice that although spring init -l lists several parameters that are supported by the Initializr, not all of those parameters are directly supported by the Spring Boot CLI 's init command. For instance, you can't specify the root package name when ini- tializing a project

with the CLI ; it will default to “demo”. spring help init can help you discover what parameters are supported by the CLI ’s init command.
Whether you use Initializr’s web-based interface, create your projects from Spring Tool Suite, or use the Spring Boot CLI to initialize a project, projects created using the Spring Boot Initializr have a familiar project layout, not unlike other Java projects you may have developed before.

１·３ 小结

Spring Boot is an exciting new way to develop Spring applications with minimal fric- tion from the framework itself. Auto-configuration eliminates much of the boilerplate configuration that infests traditional Spring applications. Spring Boot starters enable you to specify build dependencies by what they offer rather than use explicit library names and version. The Spring Boot CLI takes Spring Boot’s frictionless development model to a whole new level by enabling quick and easy development with Groovy from the command line. And the Actuator lets you look inside your running application to see what and how Spring Boot has done.

This chapter has given you a quick overview of what Spring Boot has to offer. You’re probably itching to get started on writing a real application with Spring Boot. That’s exactly what we’ll do in the next chapter. With all that Spring Boot does for you, the hardest part will be turning this page to chapter 2.

第二章： 开发你的第一个**SpringBoot**应用程序

本章内容:

- Working with Spring Boot starters
- Automatic Spring configuration

When’s the last time you went to a supermarket or major retail store and actually had to push the door open? Most large stores have automatic doors that sense your presence and open for you. Any door will enable you to enter a building, but auto- matic doors don’t require that you push or pull them open.

Similarly, many public facilities have restrooms with automatic water faucets and towel dispensers. Although not quite as prevalent as automatic supermarket doors, these devices don’t ask much of you and instead are happy to dispense water and towels.

And I honestly don’t remember the last time I even saw an ice tray, much less filled it with water or cracked it to get ice for a glass of water. My refrigerator/freezer some- how magically always has ice for me and is at the ready to fill a glass for me.

I bet you can think of countless ways that modern life is automated with devices that work for you, not the other way around. With all of this automationeverywhere, you’d think that we’d see more of it in our development tasks. Strangely,that hasn’t been so.

Up until recently, creating an application with Spring required you to do a lot of work for the framework. Sure, Spring has long had fantastic features for developing amazing applications. But it was up to you to add all of the library dependencies to the project’s build specification. And it was your job to write configuration to tell Spring what to do.

In this chapter, we’re going to look at two ways that Spring Boot has added a level of automation to Spring development: starter dependencies and automatic configura- tion. You’ll see how these essential Spring Boot features free you from the tedium and distraction of enabling Spring in your projects and let you focus on actually develop- ing your applications. Along the way, you’ll write a small but complete Spring applica- tion that puts Spring Boot to work for you.

2.1应用Ｓｐｒｉｎｇ Ｂｏｏｔ

读到了这里，事实告诉我你一个读者，或者其实你是一个书虫，阅读一切你想读的，有或许是在需要的基础上，接触这本书是为了了解如何用Ｓｐｒｉｎｇ开发你的应用程序。

无论怎样的原因，你其实都是个读者。读者可能倾向于列出一张他们喜欢的书单，根据书单，读完一个去掉一个。那让我们使用ＳｐｒｉｎｇＢｏｏｔ帮助我们开发个可以减少这种仪式的一个程序。

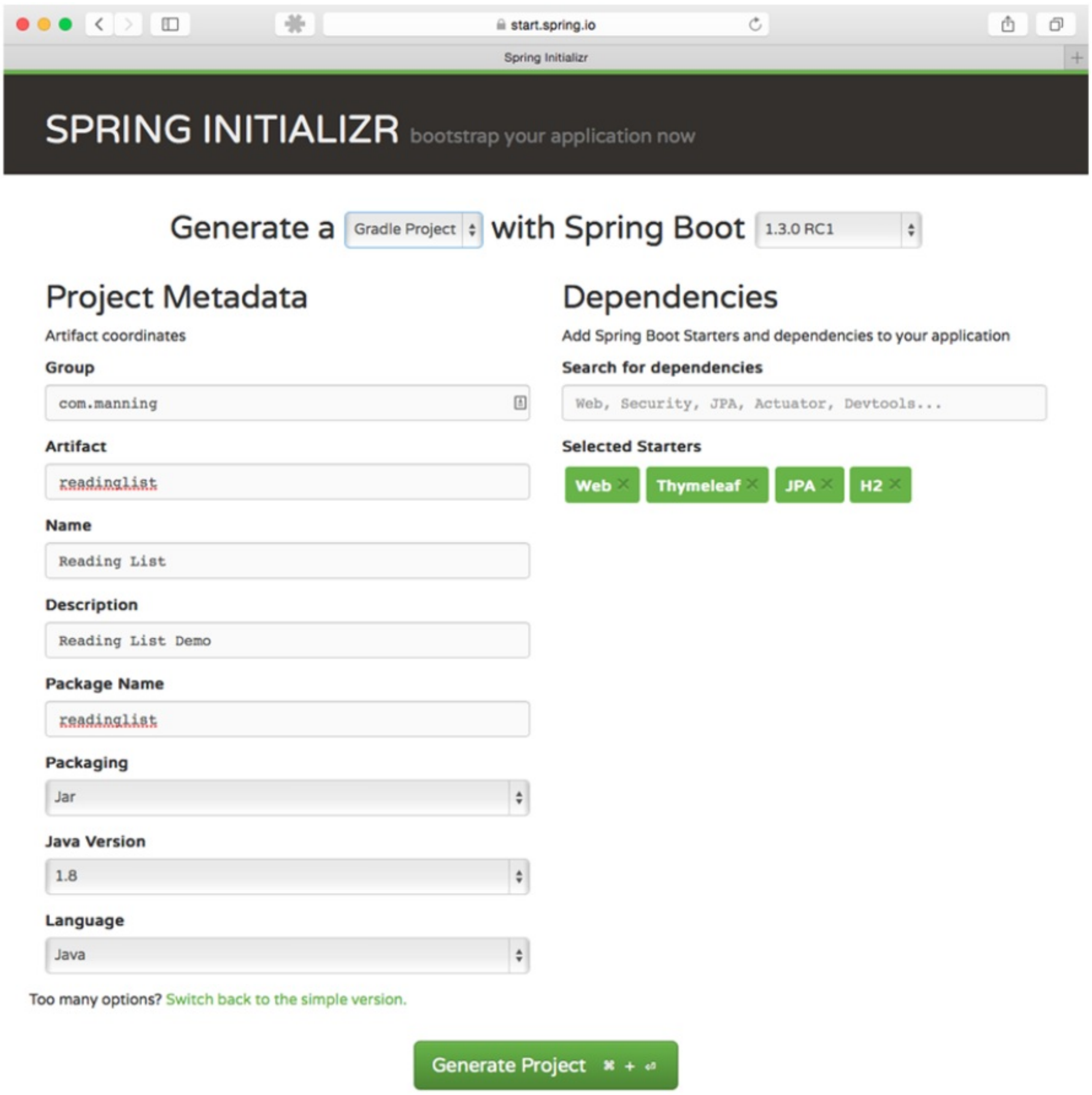
整本书我们打算构建一个阅读列表应用。通过这个应用，你可以输入你想要读的书籍，将其添加到阅读列表中，而读过的书你可以移除它们。

首先，我们需要初始化我们的项目。在第一章中，我们了解了一大把的方式使用Spring Initializ启动创建我们的项目，任何一种选择都是可以的，所以选择一个最适合你的方式即可，让我们来开始应用ＳｐｒｉｎｇＢｏｏｔ吧。

从技术角度出发，我们打算使用Ｓｐｒｉｎｇ MVC来处理ｗｅｂ请求，使用Thymeleaf来定义ｗｅｂ页面，使用ＳｐｒｉｎｇＤａｔａ ＪＰＡ来在数据库持久化和读取数据。另外，我们准备使用Ｈ2这个嵌入式的数据库。虽然我们可以使用Ｇｒｏｏｖｙ语言进行开发，但是我们还是选择使用Ｊａｖａ来进行我们的开发，最后构建工具我们将使用Ｇｒａｄｌｅ。

如果你使用Initializr，无论是ＳＴＳ，Ｗｅｂ，ＩｎｔｅｌｌｉＪ都得确保选择了Ｗｅｂ，ＪＰＡ，Ｔｈｙｍｅｌｅａｆ这些依赖。当然还要记得选上开发用的数据库Ｈ2 Ｄａｔａ Ｂａｓｅ。

至于项目的元数据，你可以选择你自己喜欢的。目的是和ｒｅａｄｉｎｇｌｉｓｔ应用有关即可，我使用了如图２．１所示的信息。



Generate a Gradle Project with Spring Boot 1.3.0 RC1

Project Metadata
Artifact coordinates
Group: com.manning
Artifact: readinglist
Name: Reading List
Description: Reading List Demo
Package Name: readinglist
Packaging: Jar
Java Version: 1.8
Language: Java

Dependencies
Add Spring Boot Starters and dependencies to your application
Search for dependencies: Web, Security, JPA, Actuator, Devtools...
Selected Starters: Web, Thymeleaf, JPA, H2

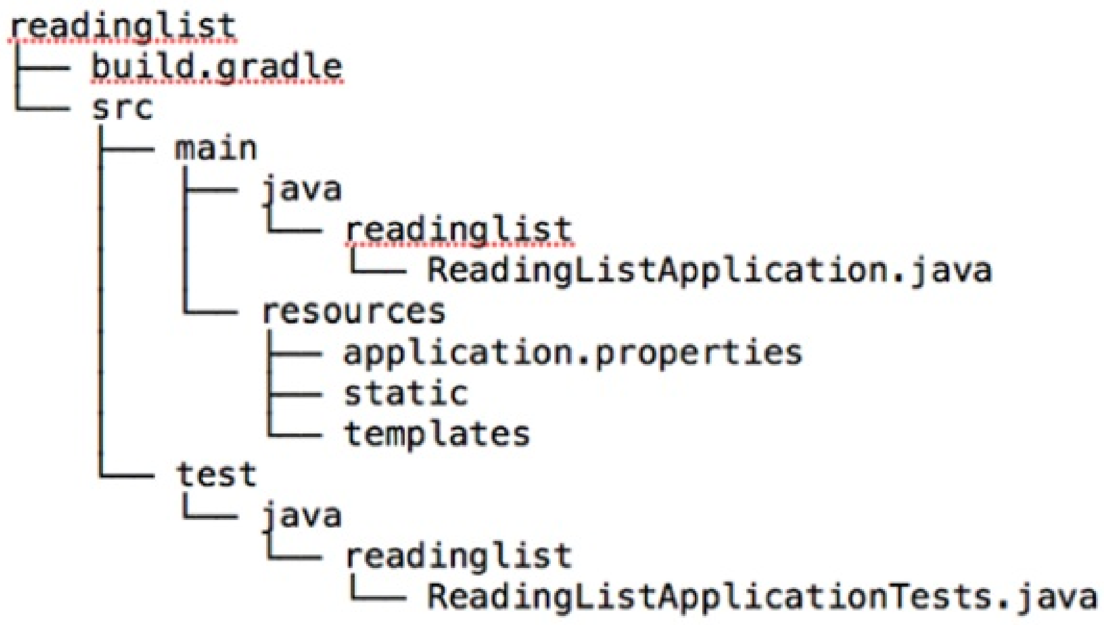
Too many options? Switch back to the simple version.

Generate Project

图 2.1 通过ｗｅｂ端的Initializr来初始化reading list 应用信息

如果你使用ＳＴS或者ＩｎｔｅｌｌｉＪ应该输入了图２．１相匹配的信息。另外如果你使用ＳｐｒｉｎｇＢｏｏｔ CLI来初始化，你可以输入如下命令:

```
$ spring init -dweb,data-jpa,h2,thymeleaf --build gradle readinglist
```



```
readinglist
├── build.gradle
└── src
    ├── main
    │   ├── java
    │   │   └── readinglist
    │   │       └── ReadingListApplication.java
    │   └── resources
    │       ├── application.properties
    │       ├── static
    │       └── templates
    └── test
        └── java
            └── readinglist
                └── ReadingListApplicationTests.java
```

Figure 2.2 The structure of the initialized readinglist project

这个是Initializr为你初始化出的必要的项目结构。现在，你可以真正的开始开发你的应用了。让我们慢慢的进一步来看看初始化的项目包含了些什么。

２・１・１ 检查刚初始化的ＳｐｒｉｎｇＢｏｏt项目

在图２・１中首先注意到的是项目是典型的Ｍａｖｅｎ或Ｇｒａｄｌｅ项目结构。你应用程序的主要代码会放在ｓｒc/main/java下，资源配置文件会放在ｓrc/main/resources下，如果你需要测试的话，测试资源文件会房子src/test/resources下，测试代码则会在ｓｒc/test/java下。

如果深入看的话，你可以看到一把文件在整个项目中：

- build.gradle—Ｇｒａｄｌｅ的构建文件
- ReadingListApplication.java—应用程序初始类，Ｓｐｒｉｎｇ主要的配置类
- application.properties—配置应用程序和ＳｐｒｉｎｇＢｏｏｔ属性的文件
- ReadingListApplicationTests.java—集成了测试的基本类

这样构建的好处有很多，我们后边会一一证明的。那么，让我们从开发ReadingListApplication.ｊａｖａ开始吧。

Ｓｐｒｉｎｇ的初始化 ReadingListApplication类在ＳｐｒｉｎｇBoot应用程序中主要有２个意图：配置和初始化。首先，它是Ｓｐｒｉｎｇ的配置中心，尽管ＳｐｒｉｎｇＢｏｏｔ的自动配置消除了Ｓｐｒｉｎｇ许多配置，但至少你需要配置下Ｓｐｒｉｎｇ使得自动配置生效。正如代码清单２－１中所示，配置只有一行即可：

代码清单**2**－１ **ReadingListApplication.java is both a bootstrap class and a configuration class**

```
package readinglist;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
//Enable component-scanning and auto-configuration
@SpringBootApplication
public class ReadingListApplication {

public static void main(String[] args) {
  //Bootstrap the application
  SpringApplication.run(ReadingListApplication.class, args);
}

}
```

@SpringBootApplication可以使得Ｓｐｒｉｎｇ包扫描以及Ｓｐｒｉｎｇ自动配置生效。事实上，@SpringBootApplication是兼备以下３个注解的功能的：

- Spring’s @Configuration —Designates a class as a configuration class using Spring’s Java-based configuration. Although we won’t be writing a lot of config- uration in this book, we’ll favor Java-based configuration over XML configura- tion when we do.
- Spring’s @ComponentScan —Enables component-scanning so that the web con- troller classes and other components you write will be automatically discovered and registered as beans in the Spring application context. A little later in this chapter, we’ll write a simple Spring MVC controller that will be annotated with @Controller so that component-scanning can find it.
- Spring Boot’s @EnableAutoConfiguration —This humble little annotation might as well be named @Abracadabra because it’s the one line of configuration that enables the magic of Spring Boot auto-configuration. This one line keeps you from having to write the pages of configuration that would be required otherwise.

在旧的ＳｐｒｉｎｇＢｏｏｔ版本中，你需要使用以上这些注解来注释ＲｅａｄＬｉｓApplication,但到了ＳｐｒｉｎｇＢｏｏｔ 1.2.0后，只需要一个@SpringBootApplication即可了。

正如我所说的，ReadingListApplication也是用来初始化的一个类。有许多方法来运行ＳｐｒｉｎｇBoot应用程序，包括传统的war包部署，而现在我们使用ｍａｉｎ()方法也可以运行你的应用程序，之后也可以通过在命令行执行ｊａｒ包来运行，通过命令行参数来停止应用程序。

事实上，你甚至不需要写任何的代码，你仍然可以构建的应用程序并试着运行它。最简单的方式构建并运行你的应用程序的化，你可以使用Ｇｒａｄｌｅ的ｂｏｏtRun任务：

```
$ gradle bootRun
```

bootRun任务来自于Gradle的插件，我们会在２·１２中在讨论。另外可以选择的就是通过在命令行执行ｇｒａｄｌｅ build来构建，并使用ｊａｖａ命令来运行：

```
$ gradle build
...
$ java -jar build/libs/readinglist-0.0.1-SNAPSHOT.jar
```

应用程序应该运行在ｔｏｍｃａｔ，并使用８０８０端口监听。你可以在浏览器上输入htｔｐ://localhost:8080访问，不过由于还没有编写Ｃｏｎｔｒｏｌｌｅｒ，所以你会得到４０４的错误页面。按照本章完成后，这个ｕｒｌ则会出现ｒｅａｄｉｎｇ－ｌｉｓｔ的页面了。

你几乎不用修改ReadingListApplication.java。如果你的应用程序需要额外的Ｓｐｒｉｎ ｇ配置，ＳｐｒｉｎｇＢｏｏｔ的自动配置会为你提供的。这在以往一般会使用@Configuration将配置配置到一些类中。If your application requires any additional Spring configuration beyond what Spring Boot auto-configuration provides, it’s usually best to write it into separate @Configuration - configured classes. (They’ll be picked up and used by component-scanning.) In exceptionally simple cases, though, you could add custom configuration to ReadingListApplication.java.

TESTING SPRING BOOT APPLICATIONS Initializr也为你提供了测试类的骨架来帮助你开始编写测试。ReadingListApplicationTests（代码清单２－２）不是一个像占位符一样的测试类，它可以为如何编写ＳｐｒｉｎｇＢｏｏ测试的例子。

代码清单**2－2 @SpringApplicationConfiguration loads a Spring application context**

```
package readinglist;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.SpringApplicationConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfiguration;

import readinglist.ReadingListApplication;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(
classes = ReadingListApplication.class) //Load context via Spring Boot
@WebAppConfiguration
public class ReadingListApplicationTests {
@Test
public void contextLoads() { //Test that the context loads
}

}
```

In a typical Spring integration test, you’d annotate the test class with @Context-Configuration to specify how the test should load the Spring application context. But in order to take full advantage of Spring Boot magic, the @SpringApplication- Configuration annotation should be used instead. As you can see from listing 2.2, ReadingListApplicationTests is annotated with @SpringApplicationConfiguration to load the Spring application context from the ReadingListApplication configura- tion class.

ReadingListApplicationTests also includes one simple test method, context- Loads() . It’s so simple, in fact, that it’s an empty method. But it’s sufficient for the purpose of verifying that the application context loads without any problems. If the configuration defined in

ReadingListApplication is good, the test will pass. If there are any problems, the test will fail.

Of course, you'll add some of your own tests as we flesh out the application. But the contextLoads() method is a fine start and verifies every bit of functionality pro- vided by the application at this point. We'll look more at how to test Spring Boot appli- cations in chapter 4.

CONFIGURING APPLICATION PROPERTIES The application.properties file given to you by the Initializr is initially empty. In fact, this file is completely optional, so you could remove it completely without impacting the application. But there's also no harm in leaving it in place.

We'll definitely find opportunity to add entries to application.properties later. For now, however, if you want to poke around with application.properties, try adding the following line:

```
server.port=8000
```

With this line, you're configuring the embedded Tomcat server to listen on port 8000 instead of the default port 8080. You can confirm this by running the application again. This demonstrates that the application.properties file comes in handy for fine- grained configuration of the stuff that Spring Boot automatically configures. But you can also use it to specify properties used by application code. We'll look at several examples of both uses of application.properties in chapter 3. The main thing to notice is that at no point do you explicitly ask Spring Boot to load application.properties for you. By virtue of the fact that application.properties exists, it will be loaded and its properties made available for configuring both Spring and application code.

We're almost finished reviewing the contents of the initialized project. But we have one last artifact to look at. Let's see how a Spring Boot application is built.

2.1.2 仔细研究ＳｐｒｉｎｇＢｏｏｔ 项目的构建

For the most part, a Spring Boot application isn’t much different from any Spring application, which isn’t much different from any Java application. Therefore, building a Spring Boot application is much like building any Java application. You have your choice of Gradle or Maven as the build tool, and you express build specifics much the same as you would in an application that doesn’t employ Spring Boot. But there are a few small details about working with Spring Boot that benefit from a little extra help in the build.

Spring Boot provides build plugins for both Gradle and Maven to assist in building Spring Boot projects. Listing 2.3 shows the build.gradle file created by Initializr, which applies the Spring Boot Gradle plugin.

２・２ 使用启动依赖

2.2.1 指定每个方面的依赖

2.2.2 覆写启动依赖中的传递依赖

2.3 使用自动配置

简言之，SpringBoot自动配置是指在运行时(准确的说是应用程序启动时)通过考虑多种因素来决定Spring该应用那些配置进行的处理动作。举例说明下，以下这些事是SpringBoot自动配置可能要考虑的:

- ClassPath中是否提供了Spring的JdbcTemplate?如果是的，如果有一个DataSource的Bean,之后自动配置一个JdbcTemplate的Bean。
- ClassPath中是否提供了Thymeleaf?如果是，之后会配置Thymeleaf的解析器，视图解析器以及模板引擎。
- ClassPath中是否有Spring Security?如果是，之后会配置一个最基本的Web Security的配置。

应用程序每次启动时关于自动配置会有将近200多个决策，包括了如安全，集成，持久化，web开发等等。所有的这些自动配置让你不至于一直写显式的配置，除非十分必要的时候。

有趣的是自动配置很难用书本描述出来。如果没有配置要写，我们又以什么来讨论呢?

2.3.1 聚焦应用的功能

有一种方式可以让我们了解SpringBoot自动配置。在接下来的几页的叙述中，会像你展示配置的时候如果不使用SpringBoot是怎样的。可是已经有很多不错关于Spring的书涉及这一点，所以再将这些配置展现一边不会是我们的readlist应用开发的更快。

不在浪费时间来讨论Spring的配置了，我们知道SpringBoot会帮我们很好的管好这一切。那就让我们看看使用SpringBoot的自动配置能让我们专注于readlist应用代码编写的优点是什么吧。

定义领域模型**(domain)**

我们的应用中主要的领域模型是书籍信息，将其使用在读者阅读列表上。因此，我们需要定义一个实体类来表述书籍信息。正如代码清单2-5所示代码一样:

代码清单 **2.5 The Book class represents a book in the reading list**

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class Book {
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
      private Long id;
      private String reader;
      private String isbn;
      private String title;
      private String author;
      private String description;
      public Long getId() {
          return id;
      }
      public void setId(Long id) {
          this.id = id;
      }
      public String getReader() {
          return reader;
      }
      public void setReader(String reader) {
          this.reader = reader;
      }
      public String getIsbn() {
          return isbn;
      }
      public void setIsbn(String isbn) {
          this.isbn = isbn;
      }
      public String getTitle() {
          return title;
      }
      public void setTitle(String title) {
          this.title = title;
      }
      public String getAuthor() {
          return author;
      }
      public void setAuthor(String author) {
          this.author = author;
      }
      public String getDescription() {
          return description;
      }
      public void setDescription(String description) {
          this.description = description;
      }
```

正如你所见,Book类是一个简单Java类，又一堆描述book的属性和必要的访问方法。使用@Entity注解指名是一个JPA的实体。id属性上注解@Id和@GeneratedValue来表示了这个域是实体的唯一标识并且它的值会自动递增。

定义**repository**的接口
Next up, we need to define the repository through which the ReadingList objects will be persisted to the database. Because we're using Spring Data JPA , that task is a simple matter of creating an interface that extends Spring Data JPA 's JpaRepository interface:

```
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
public interface ReadingListRepository extends JpaRepository<Book, Long> {
List<Book> findByReader(String reader);
}
```

By extending JpaRepository , ReadingListRepository inherits 18 methods for per- forming common persistence operations. The JpaRepository interface is parameter- ized with two parameters: the domain type that the repository will work with, and the type of its ID property. In addition, I've added a findByReader() method through which a reading list can be looked up given a reader's username.

If you're wondering about who will implement ReadingListRepository and the 18 methods it inherits, don't worry too much about it. Spring Data provides a special magic of its own, making it possible to define a repository with just an interface. The interface will be implemented automatically at runtime when the application is started.

CREATING THE WEB INTERFACE Now that we have the application's domain defined and a repository for persisting objects from that domain to the database, all that's left is to create the web front-end. A Spring MVC controller like the one in listing 2.6 will handle HTTP requests for the application.

2.3.2 运行应用

2.3.3 刚才到底发生了什么?

正如我所说过的，当没有具体配置的时候,很难描述自动配置。所以没必要花时间讨论你不需要做的，这一节我们主要关注你所需要的是什么，或则说是编写应程序代码。

但是这里仍然有些配置存在,不是吗？配置是Spring框架的主要元素，你必须有一些配置告诉Spring如何运行你的应用。　　当你添加SpringBoot到你的应用程序时，会有一个名为spring-boot-autoconfigure的jar文件，里面包含了多个配置类。每一个配置类会在你的应用的ClassPath下，会在适当的机会下为你的应用做配置。里边有Thymeleaf的配置,SpringDataJPA的配置，SpringMVC的配置还有许多其他的你可能或不可能添加到你的应用的配置中。

是什么让这一切配置特殊的，问题在于，它利用Spring有条件的配置支持，Spring4.0的对条件配置支持做了介绍，允许配置在可用的应用程序中，但除去有些条件满足需要被忽略的配置。

编写你的Spring条件配置变得容易了,因为只要你实现Condition接口中的matches()方法就可以了。比如，下面就一个简单的条件配置类，会仅当JdbcTemplate出现在Classpath中的时候会被创建对应的Bean。

```
import org.springframework.context.annotation.Condition;
import org.springframework.context.annotation.ConditionContext;
import org.springframework.core.type.AnnotatedTypeMetadata;
public class JdbcTemplateCondition implements Condition {
@Override
public boolean matches(ConditionContext context
                        ,AnnotatedTypeMetadata metadata) {
          try {
          context.getClassLoader().loadClass(
          "org.springframework.jdbc.core.JdbcTemplate");
          return true;
          } catch (Exception e) {
          return false;
          }
      }
}
```

You can use this custom condition class when you declare beans in Java:

```
@Conditional(JdbcTemplateCondition.class)
public MyService myService() {
...
}
```

在这种情况下,MyService的Bean会被创建如果JdbcTemplateCondition通过的化。也就是说，MyserviceBean只有在JdbcTemplate存在在Classpath时会被创建。否则，bean的声明将会被忽略。

尽管条件配置在这里看起来已很容易配置了，SpringBoot还定义了许多有趣的注解可以应用他们到条件配置类中，来控制SpringBoot的自动配置。即SpringBoot会应用使用这些注解了条件配置类。表2-1就是SpringBoot主要提供的条件注解:

Table 2.1 Conditional annotations used in auto-configuration

Conditional annotation	Configuration applied if...?
@ConditionalOnBean	...the specified bean has been configured
@ConditionalOnMissingBean	...the specified bean has not already been configured
@ConditionalOnClass	...the specified class is available on the classpath
@ConditionalOnMissingClass	...the specified class is not available on the classpath
@ConditionalOnExpression	...the given Spring Expression Language (SpEL) expression evaluates to true
@ConditionalOnJava	...the version of Java matches a specific value or range of versions
@ConditionalOnJndi	...there is a JNDI InitialContext available and optionally given JNDI locations exist
@ConditionalOnProperty	...the specified configuration property has a specific value
@ConditionalOnResource	...the specified resource is available on the classpath
@ConditionalOnWebApplication	...the application is a web application
@ConditionalOnNotWebApplication	...the application is not a web application

一般你不需要查看SpringBoot自动配置的相关类。但是为了介绍表2.1中的一些条件注解被使用，以下有些片段从Spring Boot的自动配置的库中摘抄出来(有关DataSourceAutoConfiguration的配置)。

```
@Configuration
@ConditionalOnClass({ DataSource.class, EmbeddedDatabaseType.class })
@EnableConfigurationProperties(DataSourceProperties.class)
@Import({ Registrar.class, DataSourcePoolMetadataProvidersConfiguration.class
})
public class DataSourceAutoConfiguration {
...
}
```

正如你所见的，DataSourceAutoConfiguration是一个基于@Configuration注解的类，导入了额外的其他配置，也定义了自己的一些Bean。最重要的是DataSourceAutoConfiguration使用了@ConditionalOnClass指名需要DataSource和EmbeddedDatabaseType在classpath中提供。如果条件失败，DataSourceAutoConfiguration的其他任何配置都会被忽略。

在DataSourceAutoConfiguration有一个内部类用来提供JdbcTemplate的自动配置：

```
@Configuration
@Conditional(DataSourceAutoConfiguration.DataSourceAvailableCondition.class)
protected static class JdbcTemplateConfiguration {
@Autowired(required = false)
private DataSource dataSource;
@Bean
@ConditionalOnMissingBean(JdbcOperations.class)
public JdbcTemplate jdbcTemplate() {
return new JdbcTemplate(this.dataSource);
}
...
}
```

JdbcTemplateConfiguration有一个最低条件@Conditional，就是首先需要有DataSourceAvailableCondition的提供才行，也就是需要一个DataSource的bean被提供和自动配置。@Bean注解在jdbcTemplate()上，以便提供jdbcTemplate的bean。不过由于@ConditionalOnMissingBean也配置在jdbcTemplate()上，仅当JdbcOperations(Jdbctemplate接口的实现)不存在的时候才会配置jdbcTemplate。

当然以上代码还说明了很多DataSourceAutoConfiguration的配置，如需要提供SpringBoot其他的配置类。以上知识让你简单了解下SpringBoot如何使用条件配置实现自动配置的。

以下配置的决定就是通过自动配置实现的，由于这些直接涉及到了我们的例子中，让我们来一起看看：

- Because H2 is on the classpath, an embedded H2 database bean will be created. This bean is of type javax.sql.DataSource , which the JPA implementation (Hibernate) will need to access the database.
- Because Hibernate Entity Manager is on the classpath (transitively via Spring Data JPA), auto-configuration will configure beans needed to support working with Hibernate, including Spring's LocalContainerEntityManagerFactory- Bean and JpaVendorAdapter .
- Because Spring Data JPA is on the classpath, Spring Data JPA will be configured to automatically create repository implementations from repository interfaces.
- Because Thymeleaf is on the classpath, Thymeleaf will be configured as a view option for Spring MVC , including a Thymeleaf template resolver, template engine, and view resolver. The template resolver is configured to resolve tem- plates from /templates relative to the root of the classpath.
- Because Spring MVC is on the classpath (thanks to the web starter depen- dency), Spring's DispatcherServlet will be configured and Spring MVC will be enabled.

- Because this is a Spring MVC web application, a resource handler will be regis- tered to serve static content from /static relative to the root of the classpath. (The resource handler will also serve static content from /public, /resources, and / META-INF /resources).
- Because Tomcat is on the classpath (transitively referred to by the web starter dependency), an embedded Tomcat container will be started to listen on port 8080.

最后，你从这里学到最主要就是，SpringBoot的自动配置可以帮你去除Spring配置的负担，让你可以专注于编写你的应用程序代码。

2.4 小结

通过SpringBoot启动依赖和自动配置，你可以快速简单的开发Spring应用程序。启动依赖帮助你只要关注应用功能而不是功能所具体需要提供的库及其版本。同时，自动配置让你从样板化的配置解放出来，可以让你使用SpringBoot开发Spring应用变得常用了。

尽管自动配置是使Spring工作的一种很方便的方式，这也为Spring开发提供了可选的方法。要是你想要或需要自定义的配置Spring，在下一章，我们会探讨如何根据应用的目标需求来覆写SpringBoot自动配置。你也可以了解到如何应用一些相同的技术来配置到你的应用程序组件上。