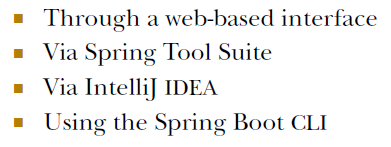
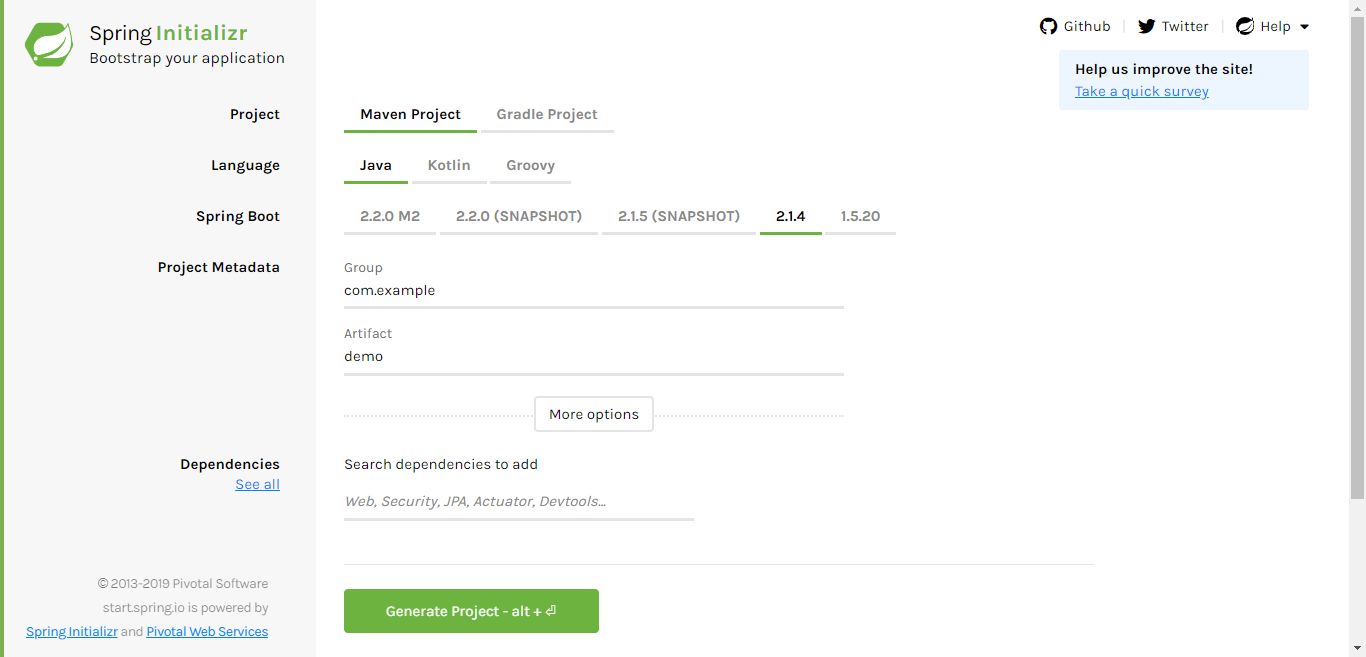
***Initializing a Spring Boot project with Spring Initializr***

* Sometimes the hardest part of a project is getting started. You need to set up a directory structure for various project artifacts, create a build file, and populate the build file with dependencies.
* The Spring Initializr is ultimately a web application that can generate a Spring Boot project structure for you. It doesn’t generate any application code, but it will give you a basic project structure and either a Maven or a Gradle build specification to build your code with. All you need to do is write the application code.
* Spring Initializr can be used in several ways:

****

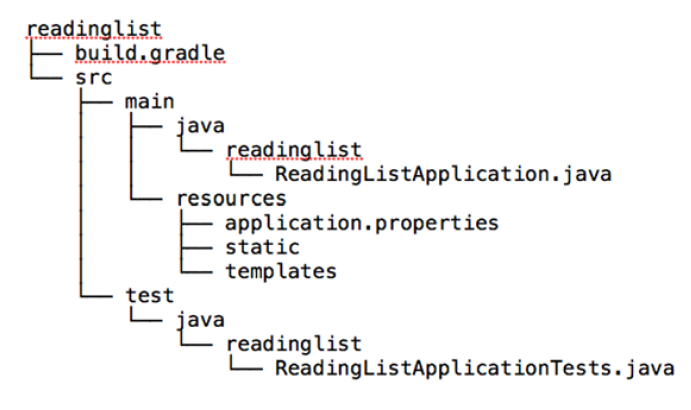
**USING SPRING INITIALIZR’S WEB INTERFACE**

The most straightforward way to use the Spring Initializr is to point your web browser to <http://start.spring.io>.

****

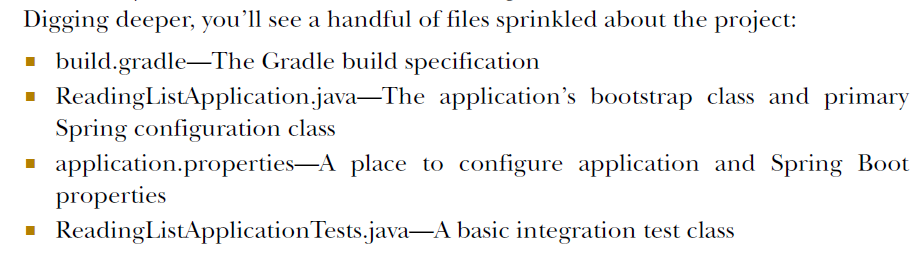
**Note:**

**Spring Initializr is a web application that generates empty Spring projects as starting points for development.**

****

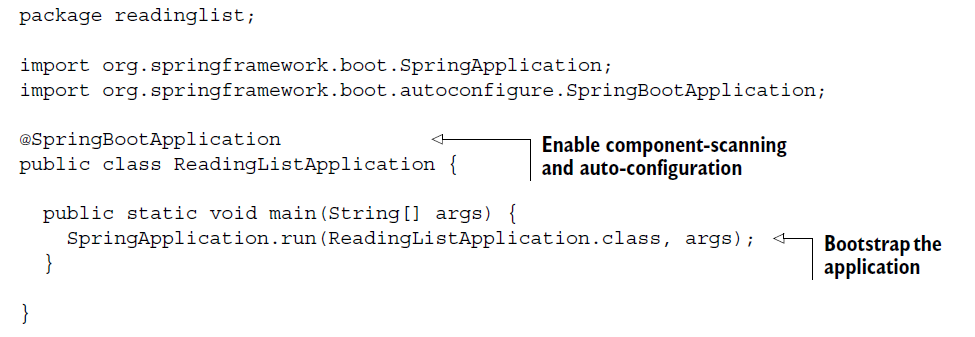
***Examining a newly initialized Spring Boot project***

The main application code is placed in the **src/main/java** branch of the directory tree, resources are placed in the **src/main/resources** branch, and test code is placed in the **src/test/java** branch. At this point we don’t have any test resources, but if we did we’d put them in **src/test/resources**.

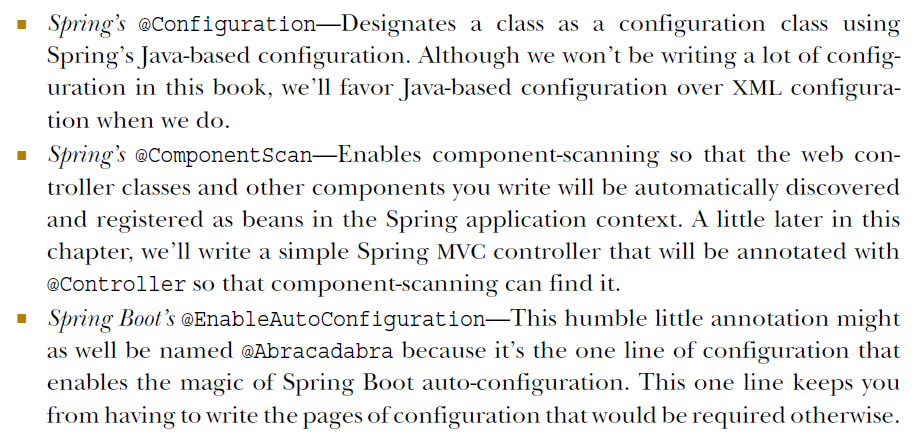


**BOOTSTRAPPING SPRING**

The **ReadingListApplication** class serves two purposes in a Spring Boot application: configuration and bootstrapping. First, it’s the central Spring configuration class. Even though Spring Boot auto-configuration eliminates the need for a lot of Spring configuration, you’ll need at least a small amount of Spring configuration to enable auto-configuration.

****

The **@SpringBootApplication** enables Spring component-scanning and Spring Boot auto-configuration. In fact, **@SpringBootApplication** combines three other useful annotations:

****

In older versions of Spring Boot, you’d have annotated the **ReadingListApplication** class with all three of these annotations. But since Spring Boot 1.2.0, **@SpringBootApplication** is all you need.

**ReadingListApplication** is also a bootstrap class. There are several ways to run Spring Boot applications, including traditional WAR file deployment. But for now the **main**() method here will enable you to run your application as an executable JAR file from the command line. It passes a reference to the **ReadingListApplication** class to **SpringApplication**.**run**(), along with the command-line arguments, to kick off the application.

****

The application should start up fine and enable a Tomcat server listening on port 8080. You can point your browser at http://localhost:8080 if you want, but because you haven’t written a controller class yet, you’ll be met with an HTTP 404 (Not Found) error and an error page.

You’ll almost never need to change ReadingListApplication.java. 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.)

**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.

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 finegrained configuration of the stuff that Spring Boot automatically configures. But you

can also use it to specify properties used by application code.

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.

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.