1. 今日大纲

1、 了解 Spring 的发展
2、 掌握 Spring 的 java 配置方式
3、 学习 Spring Boot
4、 使用 Spring Boot 来改造购物车系统

2. Spring 的发展

2.1. Spring1.x 时代

在 Spring1.x 时代，都是通过 xml 文件配置 bean，随着项目的不断扩大，需要将 xml 配置分放到不同的配置文件中，需要频繁的在 java 类和 xml 配置文件中切换。

2.2. Spring2.x 时代

随着 JDK 1.5 带来的注解支持，Spring2.x 可以使用注解对 Bean 进行申明和注入，大大的减少了 xml 配置文件，同时也大大简化了项目的开发。

那么，问题来了，究竟是应该使用 xml 还是注解呢？

最佳实践：
1、 应用的基本配置用 xml，比如：数据源、资源文件等；
2、 业务开发用注解，比如：Service 中注入 bean 等；

2.3. Spring3.x 到 Spring4.x

从 Spring3.x 开始提供了 Java 配置方式，使用 Java 配置方式可以更好的理解你配置的 Bean，现在我们就处于这个时代，并且 Spring4.x 和 Spring boot 都推荐使用 java 配置的方式。

3. Spring 的 Java 配置方式

Java 配置是 Spring4.x 推荐的配置方式，可以完全替代 xml 配置。

3.1. @Configuration 和 @Bean

Spring 的 Java 配置方式是通过 @Configuration 和 @Bean 这两个注解实现的：
1、@Configuration 作用于类上，相当于一个 xml 配置文件；
2、@Bean 作用于方法上，相当于 xml 配置中的<bean>；

3.2. 示例

该示例演示了通过 Java 配置的方式进行配置 Spring，并且实现了 Spring IOC 功能。

3.2.1. 创建工程以及导入依赖

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>cn.itcast.springboot</groupId>
   <artifactId>itcast-springboot</artifactId>
   <version>1.0.0-SNAPSHOT</version>
   <packaging>war</packaging>

   <dependencies>
      <dependency>
         <groupId>org.springframework</groupId>
         <artifactId>spring-webmvc</artifactId>
         <version>4.3.7.RELEASE</version>
      </dependency>
      <!-- 连接池 -->
      <dependency>
         <groupId>com.jolbox</groupId>
         <artifactId>bonecp-spring</artifactId>
         <version>0.8.0.RELEASE</version>
      </dependency>
   </dependencies>
   <build>
      <finalName>${project.artifactId}</finalName>
      <plugins>
         <!-- 资源文件拷贝插件 -->
         <plugin>
```

```
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-resources-
plugin</artifactId>
            <configuration>
               <encoding>UTF-8</encoding>
            </configuration>
         </plugin>
         <!-- java编译插件 -->
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
               <source>1.7</source>
               <target>1.7</target>
               <encoding>UTF-8</encoding>
            </configuration>
         </plugin>
      </plugins>
      <pluginManagement>
         <plugins>
            <!-- 配置Tomcat插件 -->
            <plugin>
               <groupId>org.apache.tomcat.maven</groupId>
               <artifactId>tomcat7-maven-
plugin</artifactId>
               <version>2.2</version>
            </plugin>
         </plugins>
      </pluginManagement>
   </build>
</project>
```

3.2.2. 编写 User 对象

```
public class User {

   private String username;

   private String password;

   private Integer age;
```

```
    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

}
```

3.2.3. 编写 UserDAO 用于模拟与数据库的交互

```
public class UserDAO {

    public List<User> queryUserList(){
        List<User> result = new ArrayList<User>();
        // 模拟数据库的查询
        for (int i = 0; i < 10; i++) {
            User user = new User();
            user.setUsername("username_" + i);
            user.setPassword("password_" + i);
            user.setAge(i + 1);
            result.add(user);
        }
        return result;
    }
}
```

3.2.4. 编写 UserService 用于实现 User 数据操作业务逻辑

```
@Service
public class UserService {

    @Autowired // 注入Spring容器中的bean对象
    private UserDAO userDAO;

    public List<User> queryUserList() {
        // 调用userDAO中的方法进行查询
        return this.userDAO.queryUserList();
    }

}
```

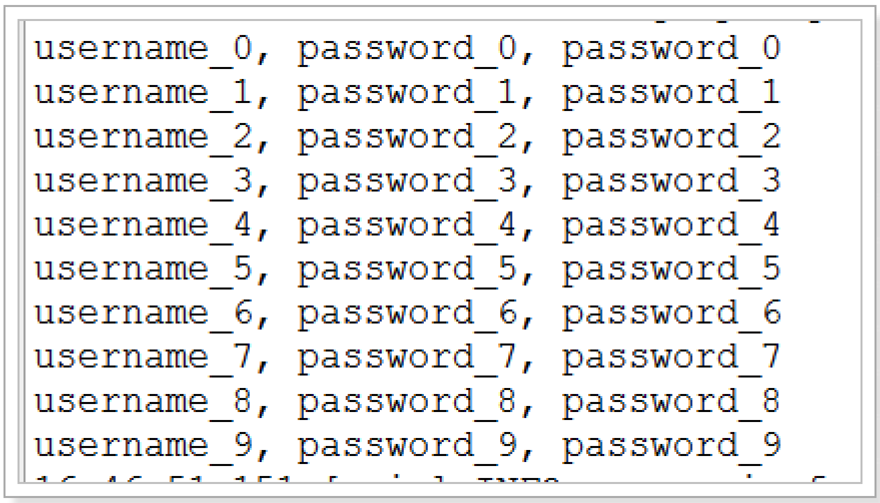
3.2.5. 编写 SpringConfig 用于实例化 Spring 容器

```
@Configuration //通过该注解来表明该类是一个Spring的配置，相当于
一个xml文件
@ComponentScan(basePackages =
"cn.itcast.springboot.javaconfig") //配置扫描包
public class SpringConfig {

    @Bean // 通过该注解来表明是一个Bean对象，相当于xml中的<bean>
    public UserDAO getUserDAO(){
        return new UserDAO(); // 直接new对象做演示
    }

}
```

3.2.6. 编写测试方法 用于启动 Spring 容器

```
public class Main {

    public static void main(String[] args) {
        // 通过Java配置来实例化Spring容器
        AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(SpringConfig.class);

        // 在Spring容器中获取Bean对象
        UserService userService =
context.getBean(UserService.class);

        // 调用对象中的方法
        List<User> list = userService.queryUserList();
        for (User user : list) {
            System.out.println(user.getUsername() + ", " +
user.getPassword() + ", " + user.getPassword());
        }

        // 销毁该容器
        context.destroy();
    }

}
```

3.2.7. 测试效果



```
username_0, password_0, password_0
username_1, password_1, password_1
username_2, password_2, password_2
username_3, password_3, password_3
username_4, password_4, password_4
username_5, password_5, password_5
username_6, password_6, password_6
username_7, password_7, password_7
username_8, password_8, password_8
username_9, password_9, password_9
```

3.2.8. 小结

从以上的示例中可以看出，使用 Java 代码就完美的替代 xml 配置文件，并且结构更加的清晰。

3.3. 实战

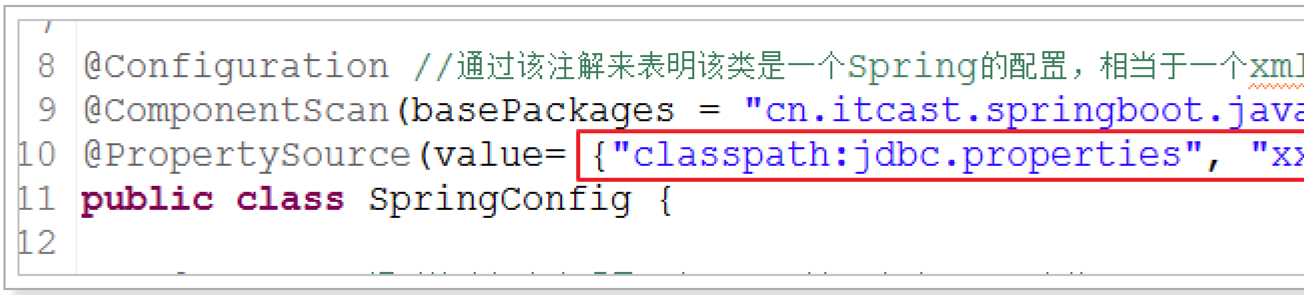
3.3.1. 读取外部的资源配置文件

通过@PropertySource 可以指定读取的配置文件，通过@Value 注解获取值，具体用法：

```
@Configuration //通过该注解来表明该类是一个Spring的配置，相当于一个xml文件
@ComponentScan(basePackages = "cn.itcast.springboot.javaconfig") //配置扫描包
@PropertySource(value= {"classpath:jdbc.properties"})
public class SpringConfig {

    @Value("${jdbc.url}")
    private String jdbcUrl;

    @Bean // 通过该注解来表明是一个Bean对象，相当于xml中的<bean>
    public UserDAO getUserDAO(){

        return new UserDAO(); // 直接new对象做演示

    }

}
```

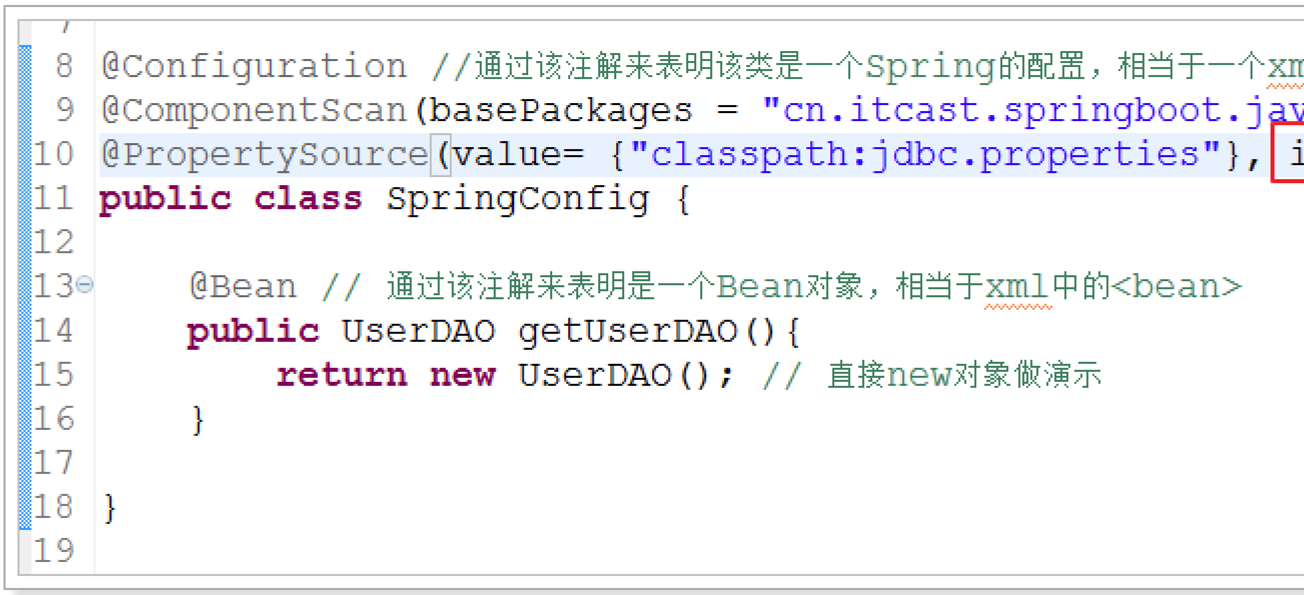
思考：

1、 如何配置多个配置文件？



```
@Configuration //通过该注解来表明该类是一个Spring的配置，相当于一个xml
@ComponentScan(basePackages = "cn.itcast.springboot.java
@PropertySource(value= {"classpath:jdbc.properties", "xx
public class SpringConfig {
```

2、 如果配置的配置文件不存在会怎么样？



```
@Configuration //通过该注解来表明该类是一个Spring的配置，相当于一个xm
@ComponentScan(basePackages = "cn.itcast.springboot.jav
@PropertySource(value= {"classpath:jdbc.properties"}, i
public class SpringConfig {

    @Bean // 通过该注解来表明是一个Bean对象，相当于xml中的<bean>
    public UserDAO getUserDAO(){
        return new UserDAO(); // 直接new对象做演示
    }

}
```

3.3.2. 配置数据库连接池

导入依赖：

```
<!-- 连接池 -->
      <dependency>
         <groupId>com.jolbox</groupId>
         <artifactId>bonecp-spring</artifactId>
         <version>0.8.0.RELEASE</version>
      </dependency>
```

之前的 Spring xml 配置：

```
    <!-- 定义数据源 -->
   <bean id="dataSource"
class="com.jolbox.bonecp.BoneCPDataSource"
      destroy-method="close">
      <!-- 数据库驱动 -->
      <property name="driverClass"
value="${jdbc.driverClassName}" />
      <!-- 相应驱动的jdbcUrl -->
      <property name="jdbcUrl" value="${jdbc.url}" />
      <!-- 数据库的用户名 -->
      <property name="username" value="${jdbc.username}"
/>
      <!-- 数据库的密码 -->
      <property name="password" value="${jdbc.password}"
/>
      <!-- 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：
240，如果要取消则设置为0 -->
      <property name="idleConnectionTestPeriod"
value="60" />
      <!-- 连接池中未使用的链接最大存活时间，单位是分，默认值：60，
如果要永远存活设置为0 -->
      <property name="idleMaxAge" value="30" />
      <!-- 每个分区最大的连接数 -->
      <!--
         判断依据：请求并发数
       -->
      <property name="maxConnectionsPerPartition"
value="100" />
      <!-- 每个分区最小的连接数 -->
      <property name="minConnectionsPerPartition"
value="5" />
```

```
    </bean>
```

参考 xml 配置改造成 java 配置方式：

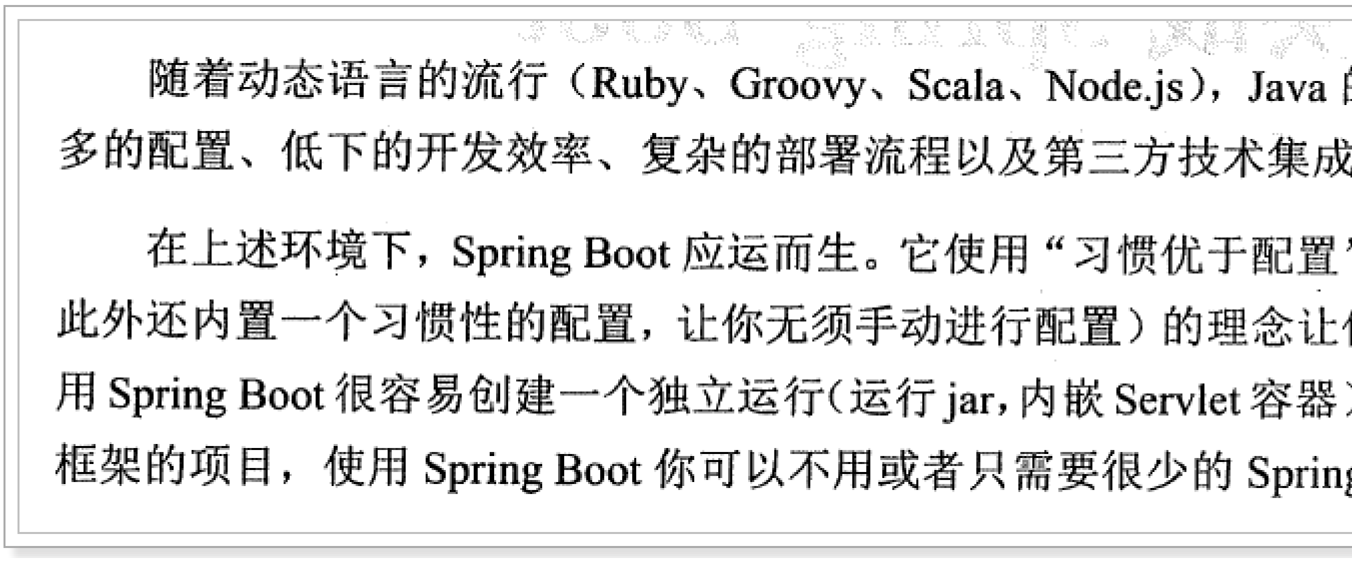
```
    @Value("${jdbc.url}")
    private String jdbcUrl;

    @Value("${jdbc.driverClassName}")
    private String jdbcDriverClassName;

    @Value("${jdbc.username}")
    private String jdbcUsername;

    @Value("${jdbc.password}")
    private String jdbcPassword;

    @Bean(destroyMethod = "close")
    public DataSource dataSource() {
        BoneCPDataSource boneCPDataSource = new
BoneCPDataSource();
        // 数据库驱动

boneCPDataSource.setDriverClass(jdbcDriverClassName);
        // 相应驱动的jdbcUrl
        boneCPDataSource.setJdbcUrl(jdbcUrl);
        // 数据库的用户名
        boneCPDataSource.setUsername(jdbcUsername);
        // 数据库的密码
        boneCPDataSource.setPassword(jdbcUsername);
        // 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：
240，如果要取消则设置为0

boneCPDataSource.setIdleConnectionTestPeriodInMinutes(60)
;
        // 连接池中未使用的链接最大存活时间，单位是分，默认值：60，
如果要永远存活设置为0
```

```
        boneCPDataSource.setIdleMaxAgeInMinutes(30);
        // 每个分区最大的连接数

boneCPDataSource.setMaxConnectionsPerPartition(100);
        // 每个分区最小的连接数
        boneCPDataSource.setMinConnectionsPerPartition(5);
        return boneCPDataSource;
    }
```

思考： 如何使用该 DataSource 对象？

4. Spring Boot

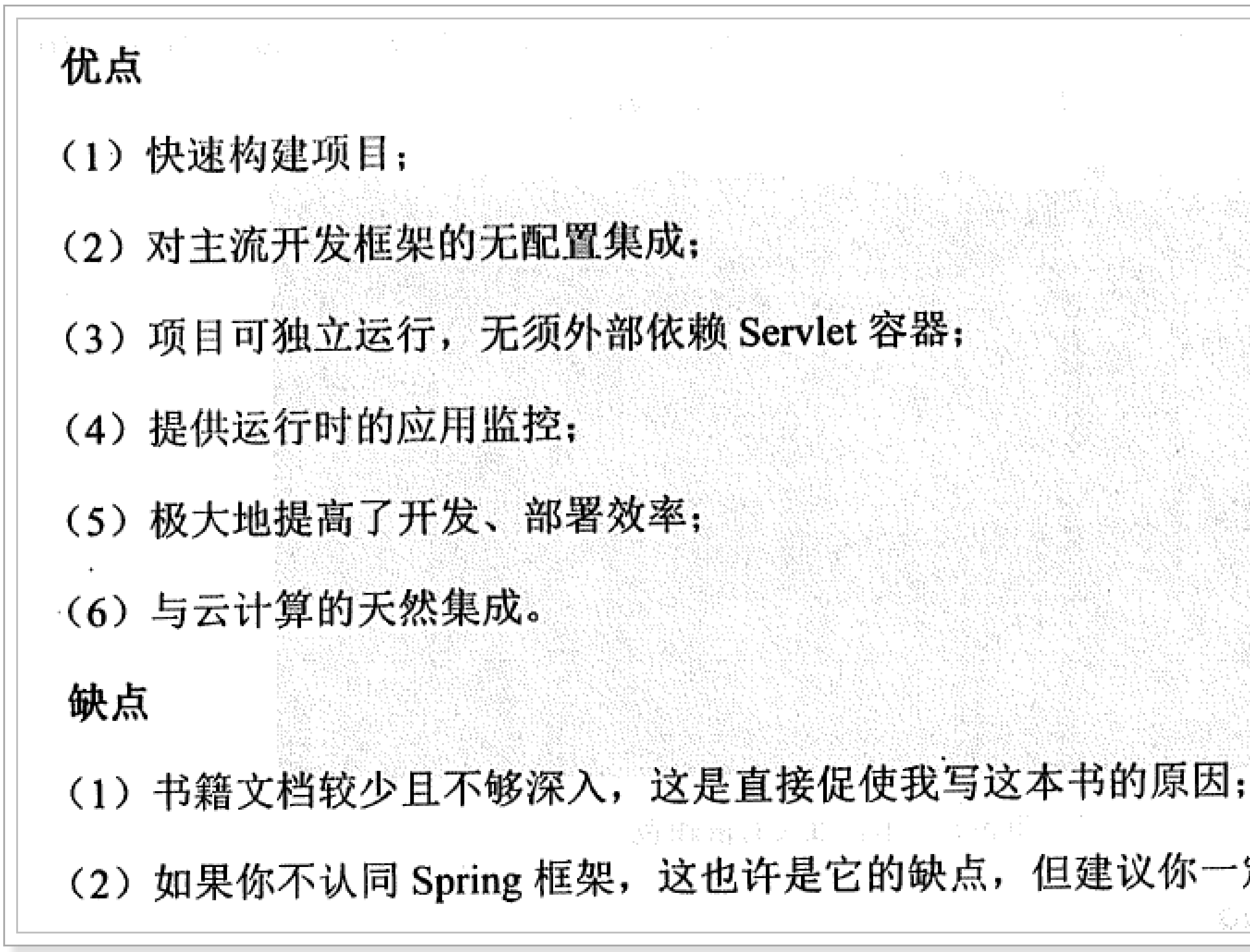
4.1. 什么是 Spring Boot



随着动态语言的流行（Ruby、Groovy、Scala、Node.js），Java
多的配置、低下的开发效率、复杂的部署流程以及第三方技术集成

在上述环境下，Spring Boot 应运而生。它使用“习惯优于配置
此外还内置一个习惯性的配置，让你无须手动进行配置）的理念让
用 Spring Boot 很容易创建一个独立运行（运行 jar，内嵌 Servlet 容器
框架的项目，使用 Spring Boot 你可以不用或者只需要很少的 Sprin

4.2. Spring Boot 的优缺点



优点

（1）快速构建项目；

（2）对主流开发框架的无配置集成；

（3）项目可独立运行，无须外部依赖 Servlet 容器；

（4）提供运行时的应用监控；

（5）极大地提高了开发、部署效率；

（6）与云计算的天然集成。

缺点

（1）书籍文档较少且不够深入，这是直接促使我写这本书的原因；

（2）如果你不认同 Spring 框架，这也许是它的缺点，但建议你一

4.3. 快速入门

4.3.1. 设置 spring boot 的 parent

```
<parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>1.5.2.RELEASE</version>
   </parent>
```

说明：Spring boot 的项目必须要将 parent 设置为 spring boot 的 parent，该 parent 包含了大量默认的配置，大大简化了我们的开发。

4.3.2. 导入 spring boot 的 web 支持

```
     <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
```

4.3.3. 添加 Spring boot 的插件

```
         <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-
plugin</artifactId>
         </plugin>
```

4.3.4. 编写第一个 Spring Boot 的应用

```
@Controller
@SpringBootApplication
@Configuration
public class HelloApplication {

   @RequestMapping("hello")
   @ResponseBody
   public String hello(){

      return "hello world! ";

   }

   public static void main(String[] args) {
      SpringApplication.run(HelloApplication.class,
args);
   }

}
```

代码说明：

1、@SpringBootApplication：Spring Boot 项目的核心注解，主要目的是开启自动配置。;

2、@Configuration：这是一个配置 Spring 的配置类;

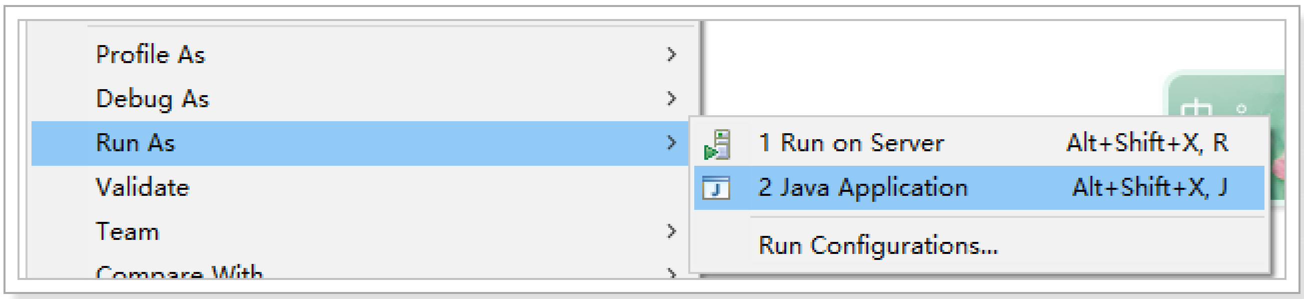
3、@Controller：标明这是一个 SpringMVC 的 Controller 控制器;

4、main 方法：在 main 方法中启动一个应用，即：这个应用的入口;

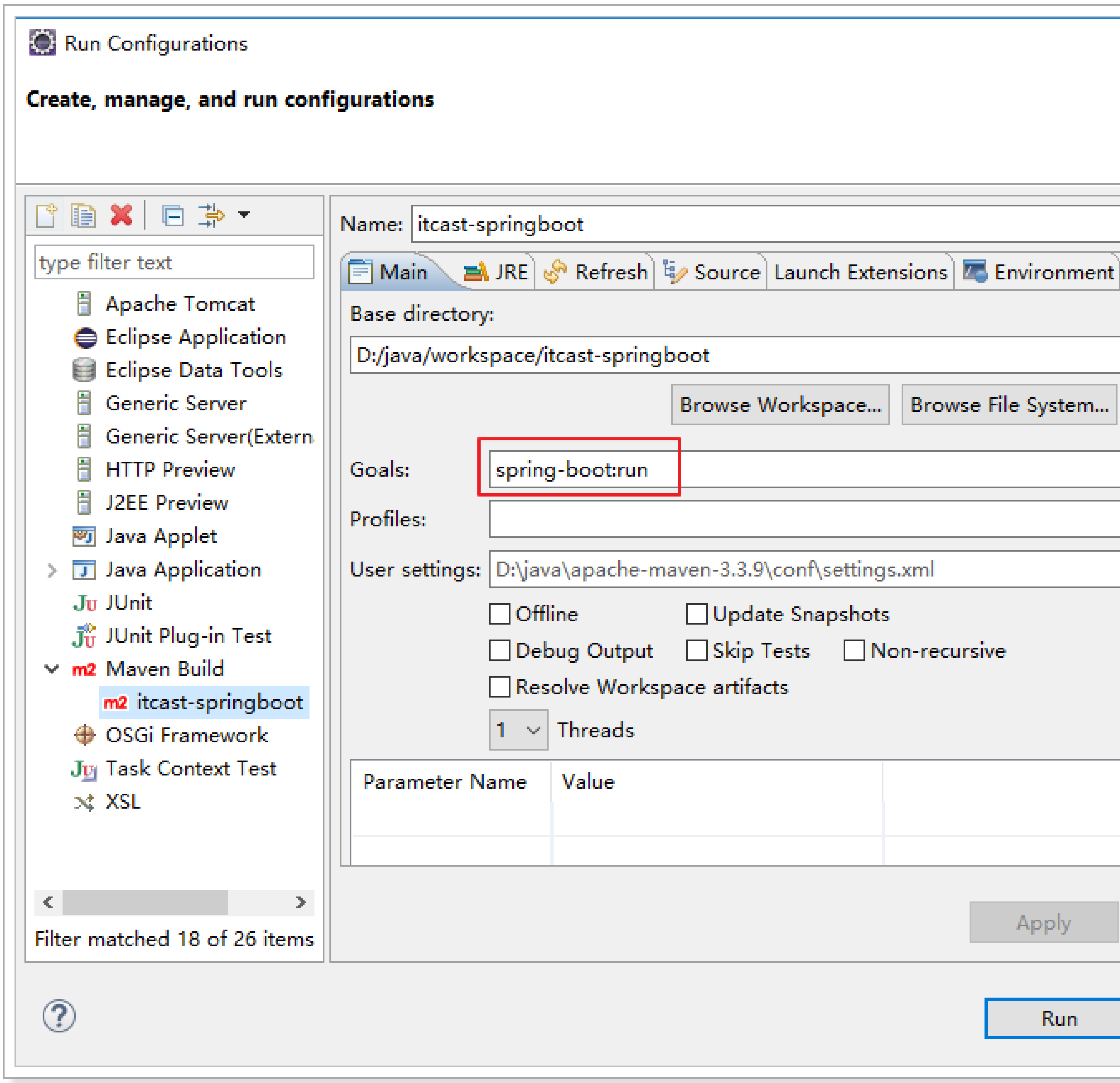
4.3.5. 启动应用

在 Spring Boot 项目中，启动的方式有两种，一种是直接 run Java Application 另外一种是通过 Spring Boot 的 Maven 插件运行。

第一种：



第二种：


Run Configurations

Create, manage, and run configurations

type filter text

- Apache Tomcat
- Eclipse Application
- Eclipse Data Tools
- Generic Server
- Generic Server(Extern
- HTTP Preview
- J2EE Preview
- Java Applet
- Java Application
- JUnit
- JUnit Plug-in Test
- Maven Build
  - itcast-springboot
- OSGi Framework
- Task Context Test
- XSL

Filter matched 18 of 26 items

Name: itcast-springboot

Main | JRE | Refresh | Source | Launch Extensions | Environment

Base directory:
D:/java/workspace/itcast-springboot

Browse Workspace... | Browse File System...

Goals: spring-boot:run

Profiles:

User settings: D:\java\apache-maven-3.3.9\conf\settings.xml

☐ Offline ☐ Update Snapshots
☐ Debug Output ☐ Skip Tests ☐ Non-recursive
☐ Resolve Workspace artifacts

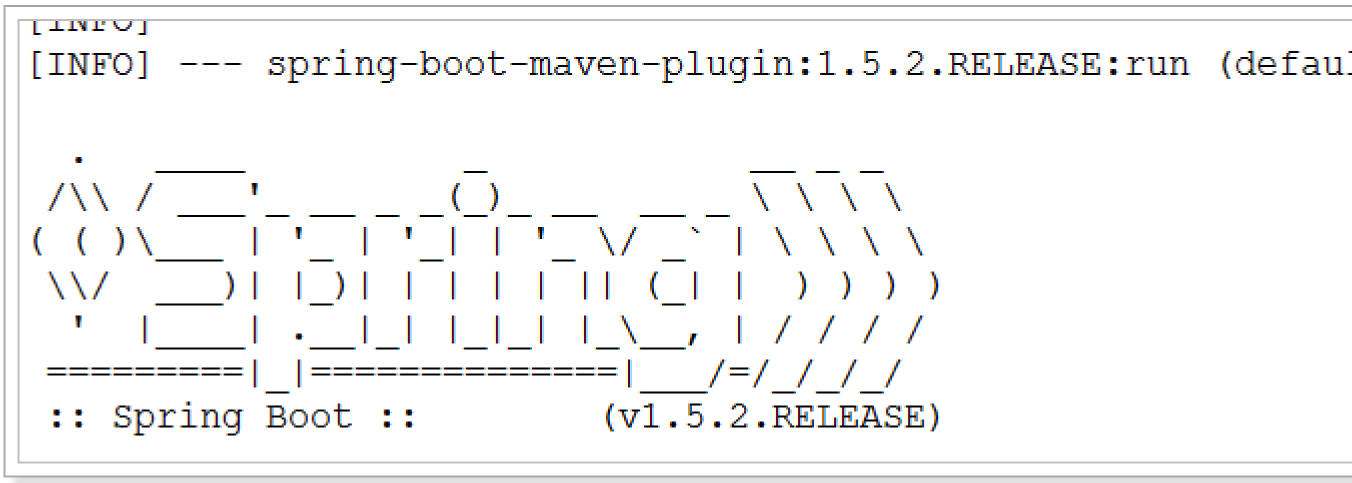
1 Threads

Parameter Name	Value

Apply

Run

启动效果：



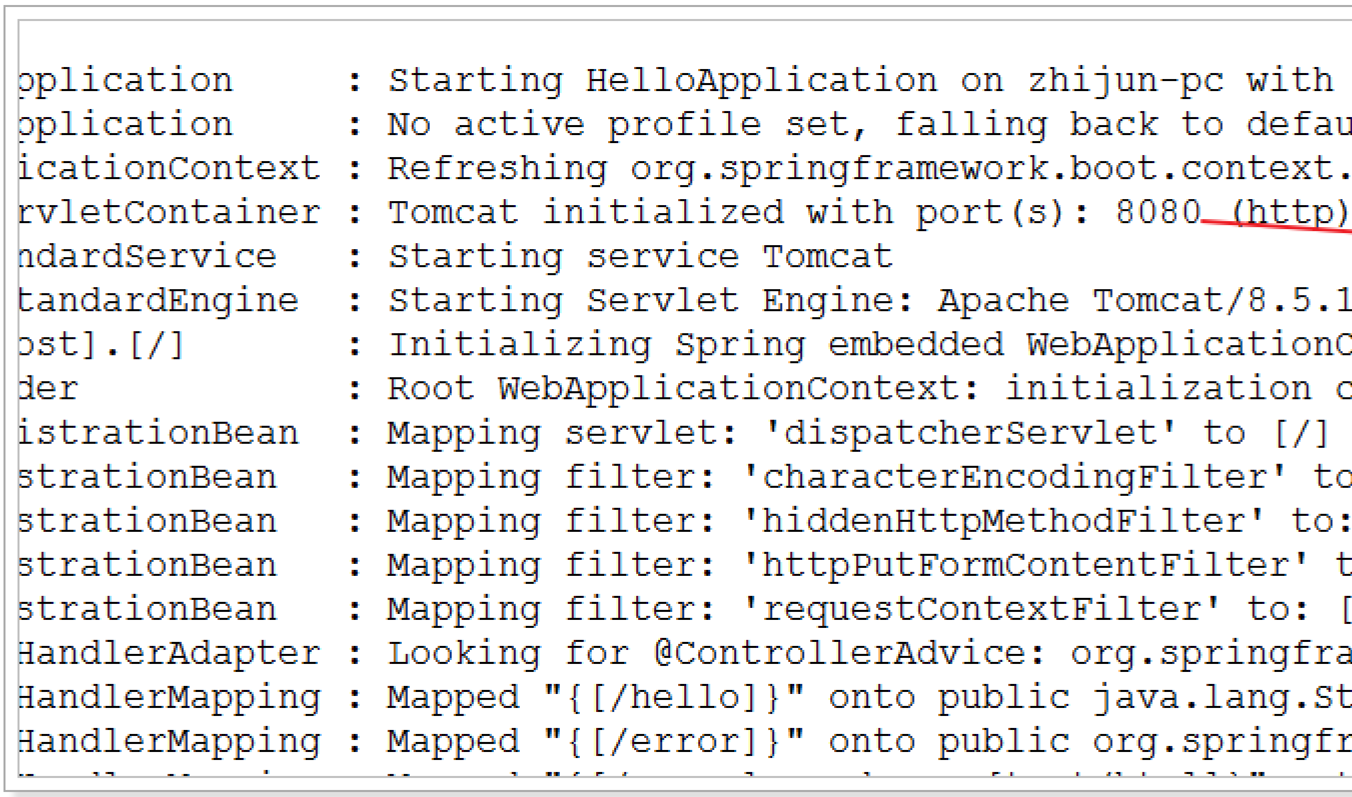
```
[INFO] --- spring-boot-maven-plugin:1.5.2.RELEASE:run (defau

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.2.RELEASE)
```

看到如下信息就说明启动成功了：

INFO 6188 --- [main] c.i.springboot.demo.HelloApplication : Started HelloApplication in 3.281 seconds (JVM running for 3.601)

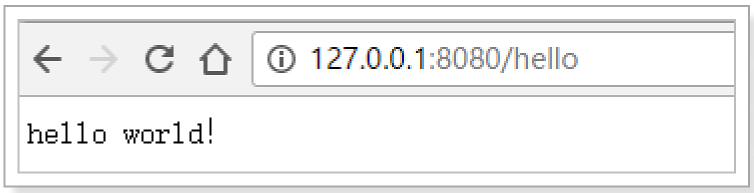
4.3.6. 测试

打开浏览器，输入地址：



```
pplication        : Starting HelloApplication on zhijun-pc with
pplication        : No active profile set, falling back to defau
icationContext    : Refreshing org.springframework.boot.context.
rvletContainer    : Tomcat initialized with port(s): 8080 (http)
ndardService      : Starting service Tomcat
tandardEngine     : Starting Servlet Engine: Apache Tomcat/8.5.1
ost].[/]          : Initializing Spring embedded WebApplicationC
der               : Root WebApplicationContext: initialization c
istrationBean     : Mapping servlet: 'dispatcherServlet' to [/]
strationBean      : Mapping filter: 'characterEncodingFilter' to
strationBean      : Mapping filter: 'hiddenHttpMethodFilter' to:
strationBean      : Mapping filter: 'httpPutFormContentFilter' t
strationBean      : Mapping filter: 'requestContextFilter' to: [
HandlerAdapter    : Looking for @ControllerAdvice: org.springfra
HandlerMapping    : Mapped "{[/hello]}" onto public java.lang.St
HandlerMapping    : Mapped "{[/error]}" onto public org.springfr
```

效果：



```
127.0.0.1:8080/hello

hello world!
```

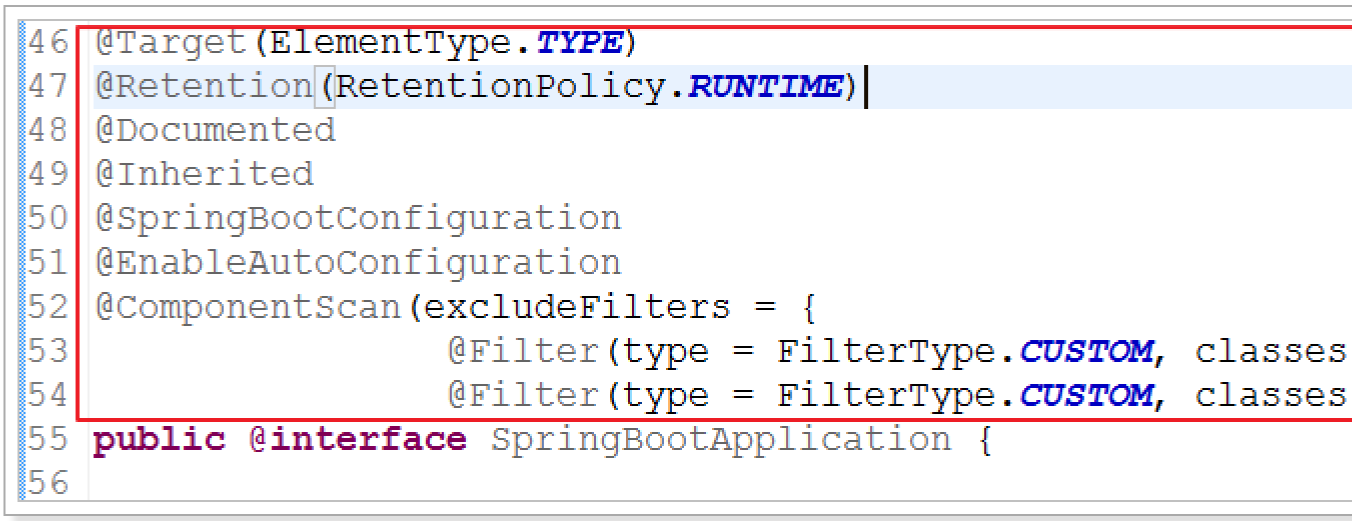
是不是很 Easy?

4.4. Spring Boot 的核心

4.4.1. 入口类和@SpringBootApplication

Spring Boot 的项目一般都会有*Application 的入口类，入口类中会有 main 方法，这是一个标准的 Java 应用程序的入口方法。

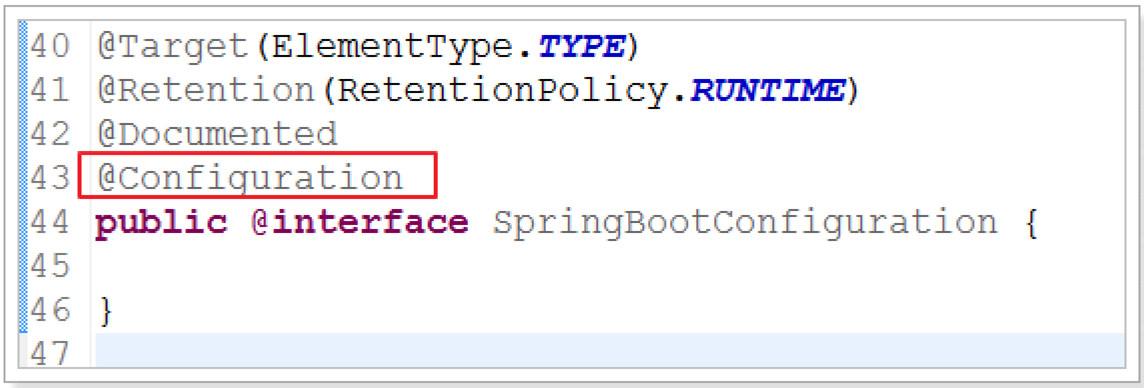
@SpringBootApplication 注解是 Spring Boot 的核心注解，它其实是一个组合注解：



```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
                @Filter(type = FilterType.CUSTOM, classes
                @Filter(type = FilterType.CUSTOM, classes
public @interface SpringBootApplication {
```

该注解主要组合了以下注解：

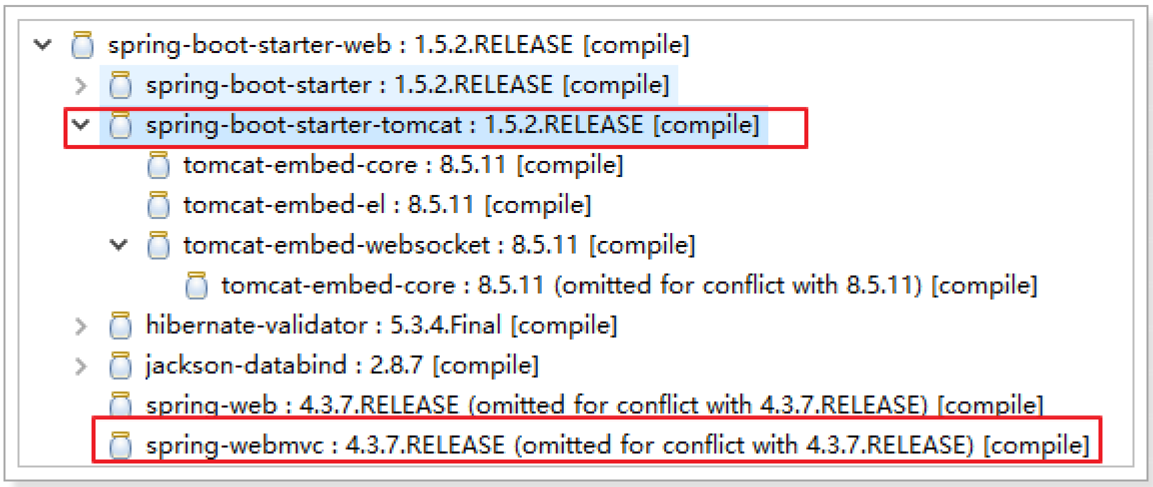
1. @SpringBootConfiguration：这是 Spring Boot 项目的配置注解，这也是一个组合注解：



```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Configuration
public @interface SpringBootConfiguration {

}
```

在 Spring Boot 项目中推荐使用@ SpringBootConfiguration 替代@Configuration

2. @EnableAutoConfiguration：启用自动配置，该注解会使 Spring Boot 根据项目中依赖的 jar 包自动配置项目的配置项：
   a) 如：我们添加了 spring-boot-starter-web 的依赖，项目中也就会引入 SpringMVC 的依赖，Spring Boot 就会自动配置 tomcat 和 SpringMVC

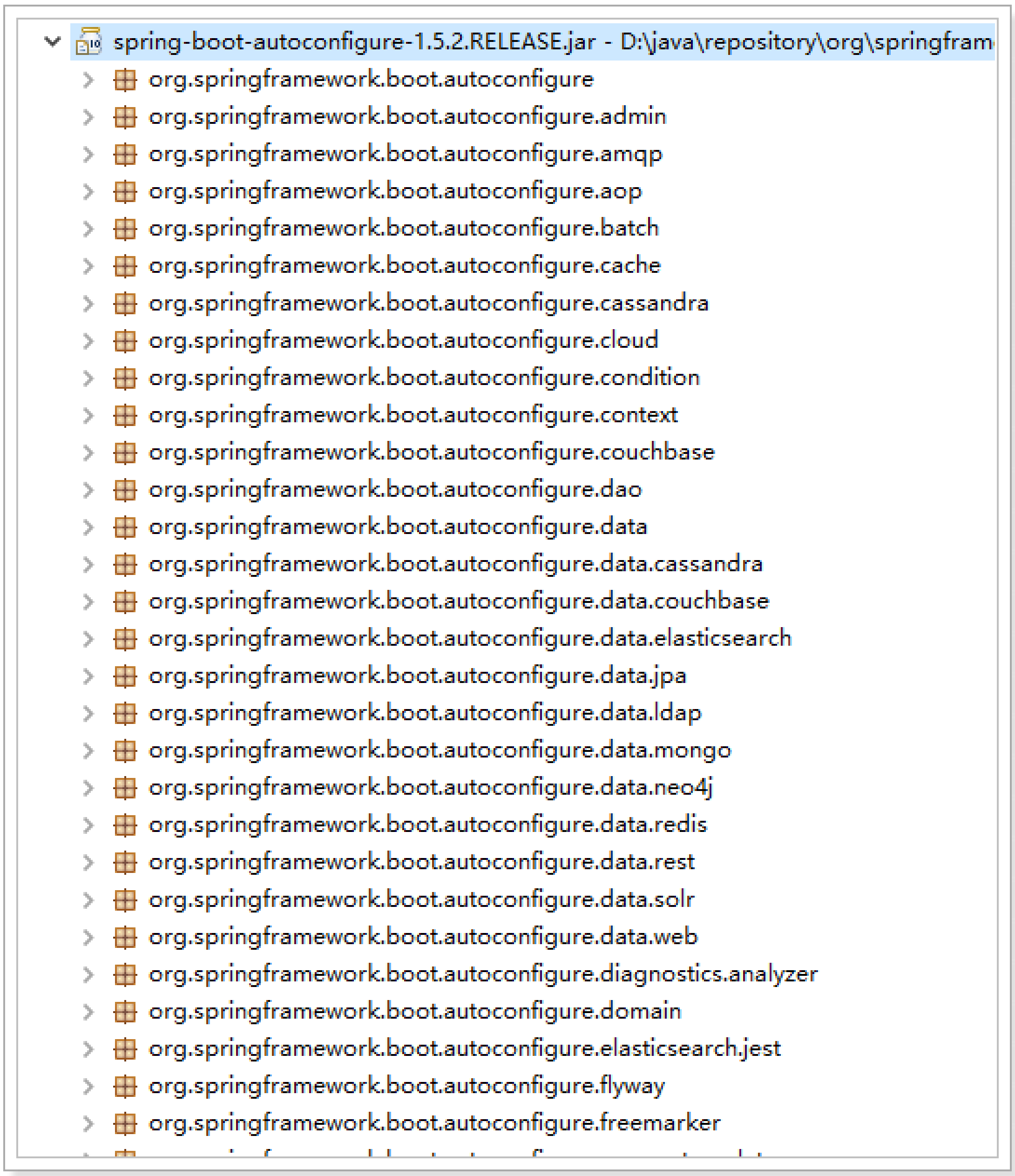


```
spring-boot-starter-web : 1.5.2.RELEASE [compile]
  spring-boot-starter : 1.5.2.RELEASE [compile]
  spring-boot-starter-tomcat : 1.5.2.RELEASE [compile]
    tomcat-embed-core : 8.5.11 [compile]
    tomcat-embed-el : 8.5.11 [compile]
    tomcat-embed-websocket : 8.5.11 [compile]
      tomcat-embed-core : 8.5.11 (omitted for conflict with 8.5.11) [compile]
  hibernate-validator : 5.3.4.Final [compile]
  jackson-databind : 2.8.7 [compile]
  spring-web : 4.3.7.RELEASE (omitted for conflict with 4.3.7.RELEASE) [compile]
  spring-webmvc : 4.3.7.RELEASE (omitted for conflict with 4.3.7.RELEASE) [compile]
```

3. @ComponentScan：默认扫描@SpringBootApplication 所在类的同级目录以及它的子目录。

4.4.2. 关闭自动配置

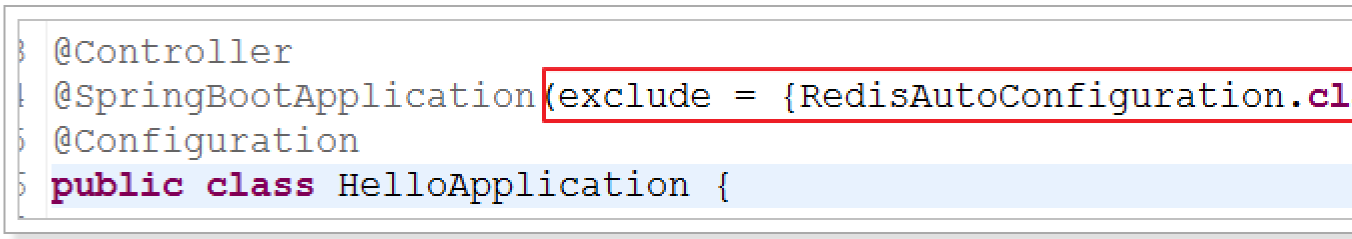
通过上述，我们得知，Spring Boot 会根据项目中的 jar 包依赖，自动做出配置，Spring Boot 支持的自动配置如下（非常多）：



```
spring-boot-autoconfigure-1.5.2.RELEASE.jar - D:\java\repository\org\springfram
    org.springframework.boot.autoconfigure
    org.springframework.boot.autoconfigure.admin
    org.springframework.boot.autoconfigure.amqp
    org.springframework.boot.autoconfigure.aop
    org.springframework.boot.autoconfigure.batch
    org.springframework.boot.autoconfigure.cache
    org.springframework.boot.autoconfigure.cassandra
    org.springframework.boot.autoconfigure.cloud
    org.springframework.boot.autoconfigure.condition
    org.springframework.boot.autoconfigure.context
    org.springframework.boot.autoconfigure.couchbase
    org.springframework.boot.autoconfigure.dao
    org.springframework.boot.autoconfigure.data
    org.springframework.boot.autoconfigure.data.cassandra
    org.springframework.boot.autoconfigure.data.couchbase
    org.springframework.boot.autoconfigure.data.elasticsearch
    org.springframework.boot.autoconfigure.data.jpa
    org.springframework.boot.autoconfigure.data.ldap
    org.springframework.boot.autoconfigure.data.mongo
    org.springframework.boot.autoconfigure.data.neo4j
    org.springframework.boot.autoconfigure.data.redis
    org.springframework.boot.autoconfigure.data.rest
    org.springframework.boot.autoconfigure.data.solr
    org.springframework.boot.autoconfigure.data.web
    org.springframework.boot.autoconfigure.diagnostics.analyzer
    org.springframework.boot.autoconfigure.domain
    org.springframework.boot.autoconfigure.elasticsearch.jest
    org.springframework.boot.autoconfigure.flyway
    org.springframework.boot.autoconfigure.freemarker
```

如果我们不需要 Spring Boot 自动配置，想关闭某一项的自动配置，该如何设置呢？

比如：我们不想自动配置 Redis，想手动配置。

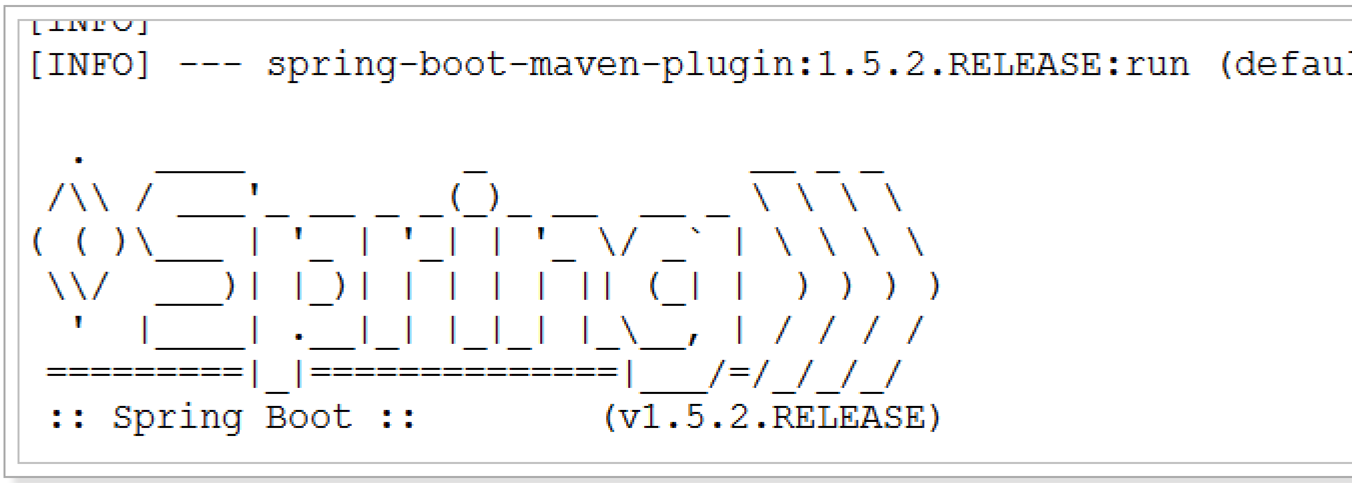


```java
@Controller
@SpringBootApplication(exclude = {RedisAutoConfiguration.cl
@Configuration
public class HelloApplication {
```

当然了，其他的配置就类似了。

4.4.3. 自定义 Banner

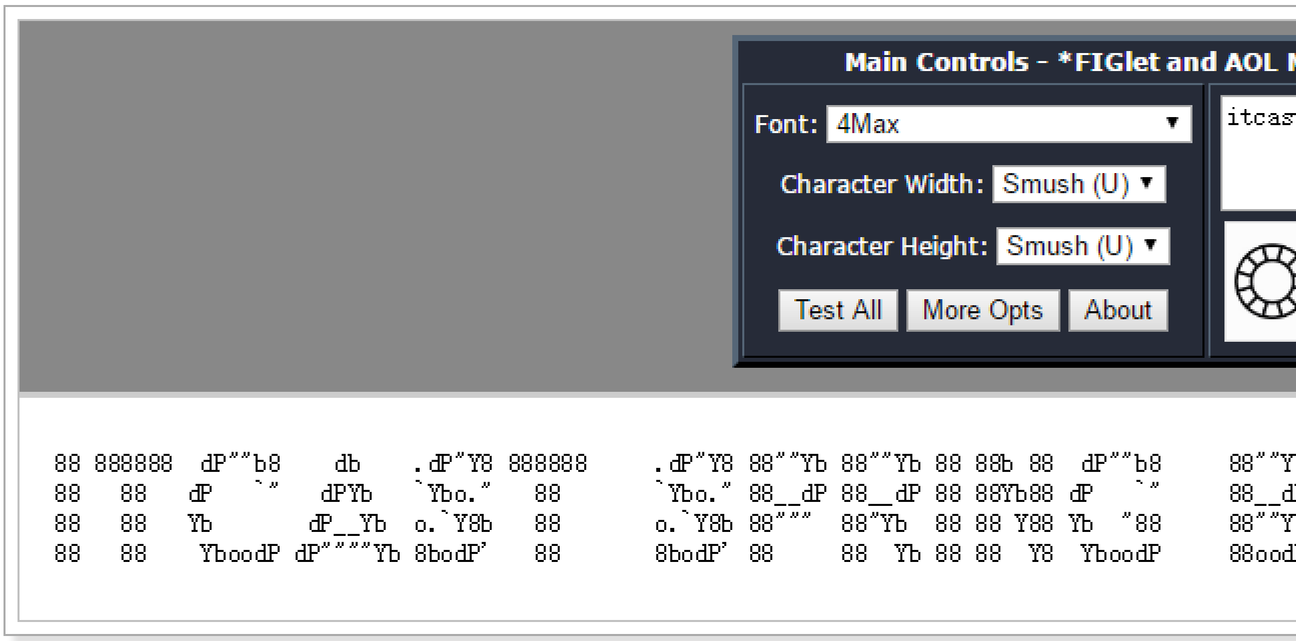
启动 Spring Boot 项目后会看到这样的图案：



```
[INFO] --- spring-boot-maven-plugin:1.5.2.RELEASE:run (defau

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.2.RELEASE)
```

这个图片其实是可以自定义的：

1. 打开网站：http://patorjk.com/software/taag/#p=display&h=3&v=3&f=4Max&t=itcast%20Spring%20Boot

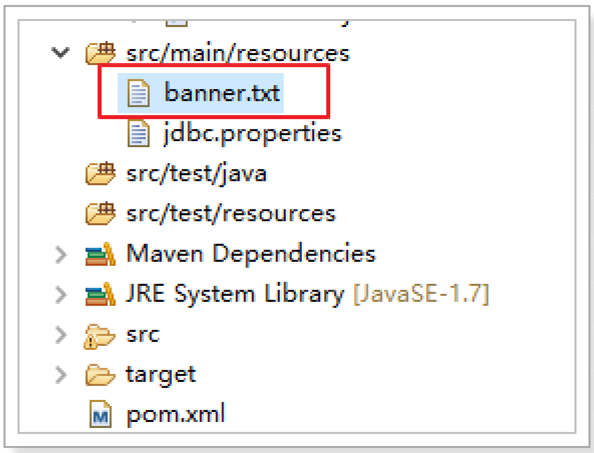


Font: 4Max
Character Width: Smush (U)
Character Height: Smush (U)

itcas

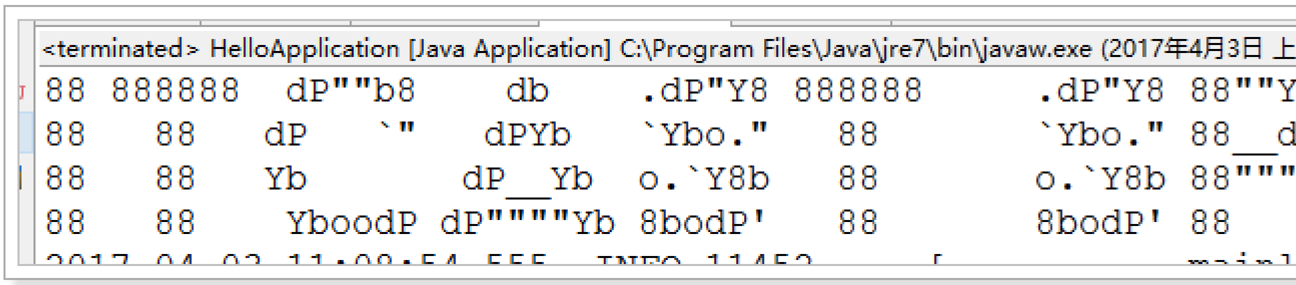
```
88 888888  dP""b8    db    .dP"Y8 888888    .dP"Y8 88""Yb 88""Yb 88 88b 88  dP""b8    88""Y
88   88   dP   `"   dPYb   `Ybo."   88      `Ybo." 88__dP 88__dP 88 88Yb88 dP   `"    88__d
88   88   Yb       dP__Yb  o.`Y8b   88      o.`Y8b 88"""  88"Yb  88 88 Y88 Yb  "88    88""Y
88   88    YboodP dP""""Yb 8bodP'   88      8bodP' 88     88  Yb 88 88  Y8  YboodP    88ood
```

2. 拷贝生成的字符到一个文本文件中，并且将该文件命名为 banner.txt
3. 将 banner.txt 拷贝到项目的 resources 目录中：



```
src/main/resources
    banner.txt
    jdbc.properties
src/test/java
src/test/resources
Maven Dependencies
JRE System Library [JavaSE-1.7]
src
target
pom.xml
```

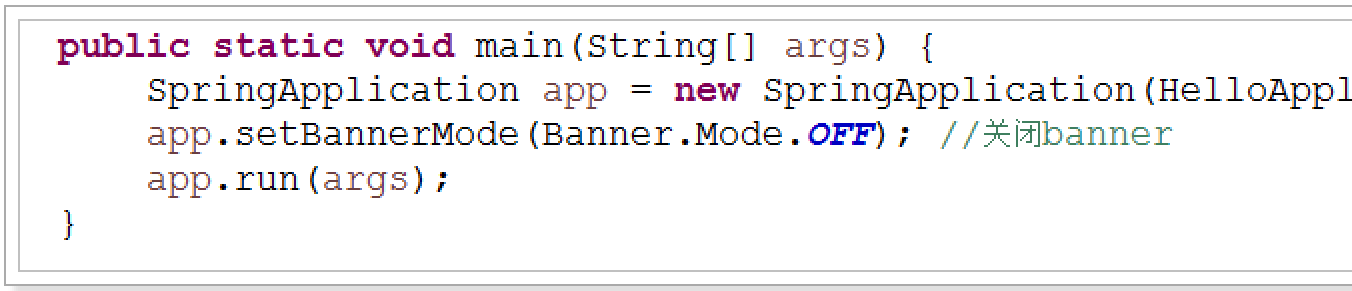
4. 重新启动程序，查看效果：



```
<terminated> HelloApplication [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (2017年4月3日 上
88 888888  dP""b8    db    .dP"Y8 888888     .dP"Y8 88""Y
88    88   dP   `"   dPYb   `Ybo."   88        `Ybo." 88__d
88    88   Yb       dP__Yb  o.`Y8b   88        o.`Y8b 88"""
88    88    YboodP dP""""Yb 8bodP'   88        8bodP' 88
2017-04-03 11:08:54.555  INFO 11452 --- [          main
```

好像没有默认的好看啊！！！

如果不想看到任何的 banner，也是可以将其关闭的：

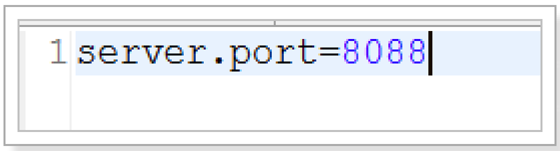


```java
public static void main(String[] args) {
    SpringApplication app = new SpringApplication(HelloAppl
    app.setBannerMode(Banner.Mode.OFF); //关闭banner
    app.run(args);
}
```

4.4.4. 全局配置文件

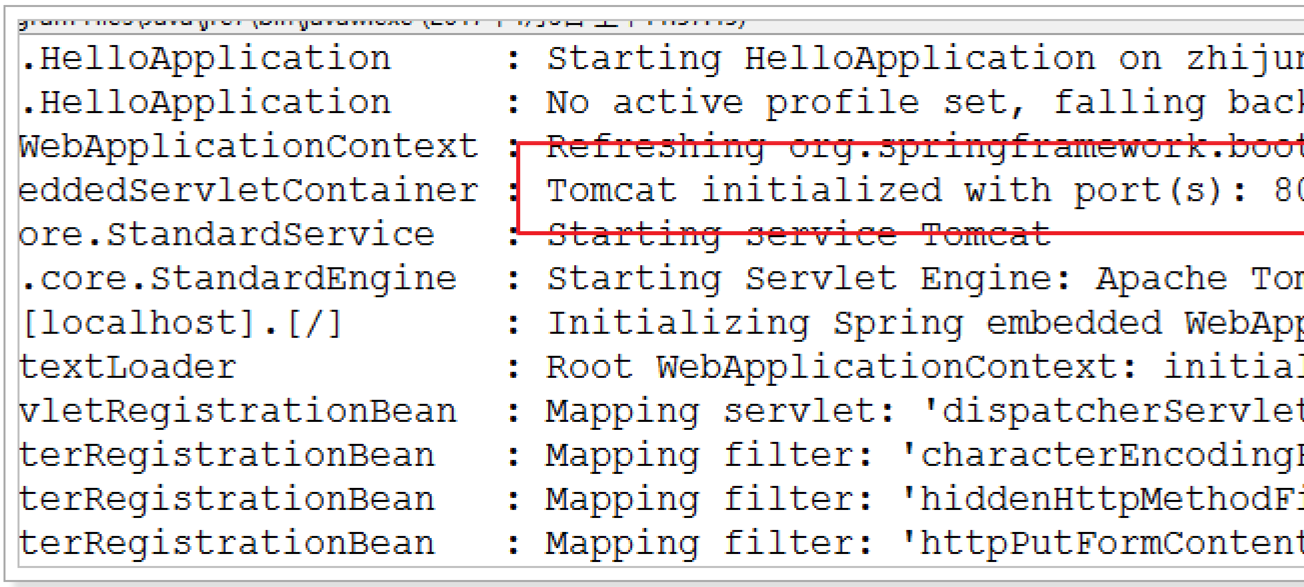
Spring Boot 项目使用一个全局的配置文件 application.properties 或者是 application.yml，在 resources 目录下或者类路径下的/config 下，一般我们放到 resources 下。

1、 修改 tomcat 的端口为 8088



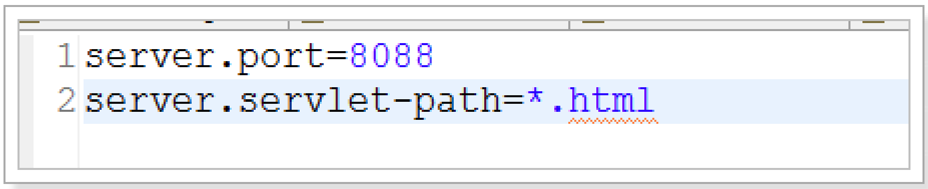
```
server.port=8088
```

重新启动应用，查看效果：



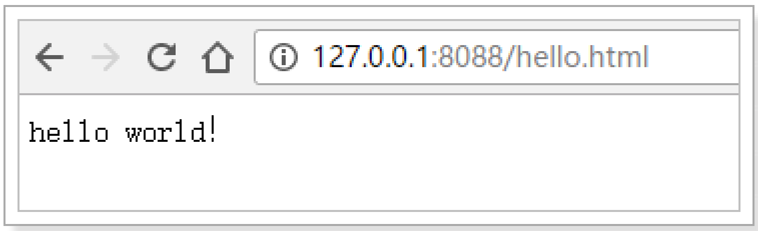
```
.HelloApplication          : Starting HelloApplication on zhijun
.HelloApplication          : No active profile set, falling back
WebApplicationContext      : Refreshing org.springframework.boot
eddedServletContainer      : Tomcat initialized with port(s): 80
ore.StandardService        : Starting service Tomcat
.core.StandardEngine       : Starting Servlet Engine: Apache Tom
[localhost].[/]            : Initializing Spring embedded WebApp
textLoader                 : Root WebApplicationContext: initial
vletRegistrationBean       : Mapping servlet: 'dispatcherServlet
terRegistrationBean        : Mapping filter: 'characterEncodingF
terRegistrationBean        : Mapping filter: 'hiddenHttpMethodFi
terRegistrationBean        : Mapping filter: 'httpPutFormContent
```

2、 修改进入 DispatcherServlet 的规则为：*.html



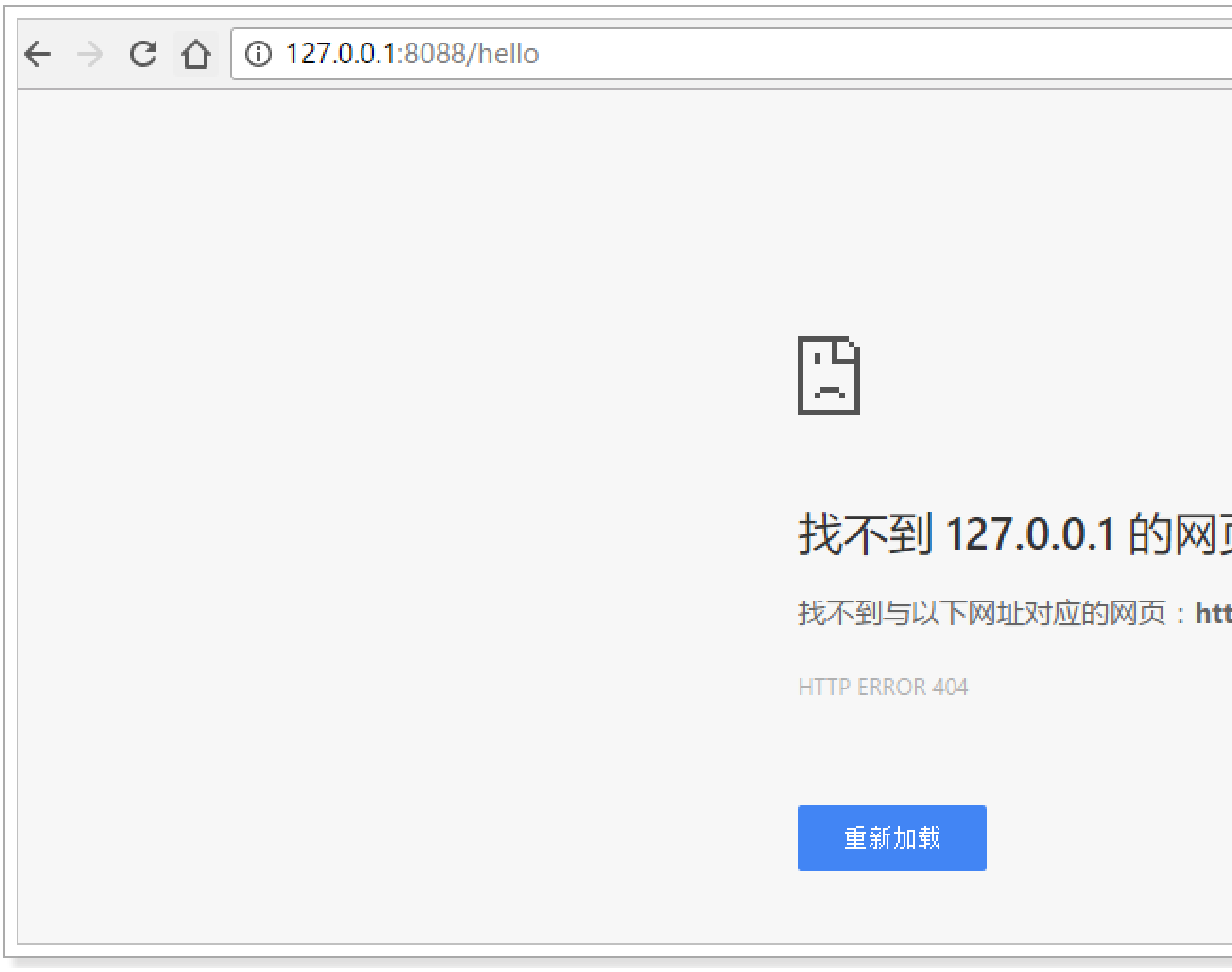
```
server.port=8088
server.servlet-path=*.html
```

测试：



127.0.0.1:8088/hello.html

hello world!


127.0.0.1:8088/hello

找不到 127.0.0.1 的网页

找不到与以下网址对应的网页：htt

HTTP ERROR 404

更多的配置：

```
#
==================================================================
==
# COMMON SPRING BOOT PROPERTIES
#
# This sample file is provided as a guideline. Do NOT copy it in
its
# entirety to your own application.              ^^^
#
==================================================================
==


# ----------------------------------------
# CORE PROPERTIES
# ----------------------------------------

# BANNER
banner.charset=UTF-8 # Banner file encoding.
banner.location=classpath:banner.txt # Banner file location.
banner.image.location=classpath:banner.gif # Banner image file
location (jpg/png can also be used).
banner.image.width= # Width of the banner image in chars (default
76)
banner.image.height= # Height of the banner image in chars
(default based on image height)
banner.image.margin= # Left hand image margin in chars (default
2)
banner.image.invert= # If images should be inverted for dark
terminal themes (default false)

# LOGGING
logging.config= # Location of the logging configuration file. For
instance `classpath:logback.xml` for Logback
logging.exception-conversion-word=%wEx # Conversion word used
when logging exceptions.
logging.file= # Log file name. For instance `myapp.log`
logging.level.*= # Log levels severity mapping. For instance
`logging.level.org.springframework=DEBUG`
logging.path= # Location of the log file. For instance `/var/log`
logging.pattern.console= # Appender pattern for output to the
console. Only supported with the default logback setup.
logging.pattern.file= # Appender pattern for output to the file.
Only supported with the default logback setup.
logging.pattern.level= # Appender pattern for log level
(default %5p). Only supported with the default logback setup.
logging.register-shutdown-hook=false # Register a shutdown hook
for the logging system when it is initialized.

# AOP
spring.aop.auto=true # Add @EnableAspectJAutoProxy.
spring.aop.proxy-target-class=false # Whether subclass-based
(CGLIB) proxies are to be created (true) as opposed to standard
Java interface-based proxies (false).

# IDENTITY (ContextIdApplicationContextInitializer)
spring.application.index= # Application index.
spring.application.name= # Application name.
```

```
# ADMIN (SpringApplicationAdminJmxAutoConfiguration)
spring.application.admin.enabled=false # Enable admin features
for the application.
spring.application.admin.jmx-
name=org.springframework.boot:type=Admin,name=SpringApplication #
JMX name of the application admin MBean.

# AUTO-CONFIGURATION
spring.autoconfigure.exclude= # Auto-configuration classes to
exclude.

# SPRING CORE
spring.beaninfo.ignore=true # Skip search of BeanInfo classes.

# SPRING CACHE (CacheProperties)
spring.cache.cache-names= # Comma-separated list of cache names
to create if supported by the underlying cache manager.
spring.cache.caffeine.spec= # The spec to use to create caches.
Check CaffeineSpec for more details on the spec format.
spring.cache.couchbase.expiration=0 # Entry expiration in
milliseconds. By default the entries never expire.
spring.cache.ehcache.config= # The location of the configuration
file to use to initialize EhCache.
spring.cache.guava.spec= # The spec to use to create caches.
Check CacheBuilderSpec for more details on the spec format.
spring.cache.infinispan.config= # The location of the
configuration file to use to initialize Infinispan.
spring.cache.jcache.config= # The location of the configuration
file to use to initialize the cache manager.
spring.cache.jcache.provider= # Fully qualified name of the
CachingProvider implementation to use to retrieve the JSR-107
compliant cache manager. Only needed if more than one JSR-107
implementation is available on the classpath.
spring.cache.type= # Cache type, auto-detected according to the
environment by default.

# SPRING CONFIG - using environment property only
(ConfigFileApplicationListener)
spring.config.location= # Config file locations.
spring.config.name=application # Config file name.

# HAZELCAST (HazelcastProperties)
spring.hazelcast.config= # The location of the configuration file
to use to initialize Hazelcast.
```

```
# PROJECT INFORMATION (ProjectInfoProperties)
spring.info.build.location=classpath:META-INF/build-
info.properties # Location of the generated build-info.properties
file.
spring.info.git.location=classpath:git.properties # Location of
the generated git.properties file.

# JMX
spring.jmx.default-domain= # JMX domain name.
spring.jmx.enabled=true # Expose management beans to the JMX
domain.
spring.jmx.server=mbeanServer # MBeanServer bean name.

# Email (MailProperties)
spring.mail.default-encoding=UTF-8 # Default MimeMessage
encoding.
spring.mail.host= # SMTP server host. For instance
`smtp.example.com`
spring.mail.jndi-name= # Session JNDI name. When set, takes
precedence to others mail settings.
spring.mail.password= # Login password of the SMTP server.
spring.mail.port= # SMTP server port.
spring.mail.properties.*= # Additional JavaMail session
properties.
spring.mail.protocol=smtp # Protocol used by the SMTP server.
spring.mail.test-connection=false # Test that the mail server is
available on startup.
spring.mail.username= # Login user of the SMTP server.

# APPLICATION SETTINGS (SpringApplication)
spring.main.banner-mode=console # Mode used to display the banner
when the application runs.
spring.main.sources= # Sources (class name, package name or XML
resource location) to include in the ApplicationContext.
spring.main.web-environment= # Run the application in a web
environment (auto-detected by default).

# FILE ENCODING (FileEncodingApplicationListener)
spring.mandatory-file-encoding= # Expected character encoding the
application must use.

# INTERNATIONALIZATION (MessageSourceAutoConfiguration)
```

```
spring.messages.always-use-message-format=false # Set whether to
always apply the MessageFormat rules, parsing even messages
without arguments.
spring.messages.basename=messages # Comma-separated list of
basenames, each following the ResourceBundle convention.
spring.messages.cache-seconds=-1 # Loaded resource bundle files
cache expiration, in seconds. When set to -1, bundles are cached
forever.
spring.messages.encoding=UTF-8 # Message bundles encoding.
spring.messages.fallback-to-system-locale=true # Set whether to
fall back to the system Locale if no files for a specific Locale
have been found.

# OUTPUT
spring.output.ansi.enabled=detect # Configure the ANSI output.

# PID FILE (ApplicationPidFileWriter)
spring.pid.fail-on-write-error= # Fail if
ApplicationPidFileWriter is used but it cannot write the PID
file.
spring.pid.file= # Location of the PID file to write (if
ApplicationPidFileWriter is used).

# PROFILES
spring.profiles.active= # Comma-separated list (or list if using
YAML) of active profiles.
spring.profiles.include= # Unconditionally activate the specified
comma separated profiles (or list of profiles if using YAML).

# SENDGRID (SendGridAutoConfiguration)
spring.sendgrid.api-key= # SendGrid api key (alternative to
username/password)
spring.sendgrid.username= # SendGrid account username
spring.sendgrid.password= # SendGrid account password
spring.sendgrid.proxy.host= # SendGrid proxy host
spring.sendgrid.proxy.port= # SendGrid proxy port


# ----------------------------------------
# WEB PROPERTIES
# ----------------------------------------

# EMBEDDED SERVER CONFIGURATION (ServerProperties)
```

```
server.address= # Network address to which the server should bind
to.
server.compression.enabled=false # If response compression is
enabled.
server.compression.excluded-user-agents= # List of user-agents to
exclude from compression.
server.compression.mime-types= # Comma-separated list of MIME
types that should be compressed. For instance
`text/html,text/css,application/json`
server.compression.min-response-size= # Minimum response size
that is required for compression to be performed. For instance
2048
server.connection-timeout= # Time in milliseconds that connectors
will wait for another HTTP request before closing the connection.
When not set, the connector's container-specific default will be
used. Use a value of -1 to indicate no (i.e. infinite) timeout.
server.context-parameters.*= # Servlet context init parameters.
For instance `server.context-parameters.a=alpha`
server.context-path= # Context path of the application.
server.display-name=application # Display name of the
application.
server.max-http-header-size=0 # Maximum size in bytes of the HTTP
message header.
server.error.include-stacktrace=never # When to include a
"stacktrace" attribute.
server.error.path=/error # Path of the error controller.
server.error.whitelabel.enabled=true # Enable the default error
page displayed in browsers in case of a server error.
server.jetty.acceptors= # Number of acceptor threads to use.
server.jetty.max-http-post-size=0 # Maximum size in bytes of the
HTTP post or put content.
server.jetty.selectors= # Number of selector threads to use.
server.jsp-servlet.class-
name=org.apache.jasper.servlet.JspServlet # The class name of the
JSP servlet.
server.jsp-servlet.init-parameters.*= # Init parameters used to
configure the JSP servlet
server.jsp-servlet.registered=true # Whether or not the JSP
servlet is registered
server.port=8080 # Server HTTP port.
server.server-header= # Value to use for the Server response
header (no header is sent if empty)
server.servlet-path=/ # Path of the main dispatcher servlet.
server.use-forward-headers= # If X-Forwarded-* headers should be
applied to the HttpRequest.
server.session.cookie.comment= # Comment for the session cookie.
server.session.cookie.domain= # Domain for the session cookie.
server.session.cookie.http-only= # "HttpOnly" flag for the
session cookie.
server.session.cookie.max-age= # Maximum age of the session
cookie in seconds.
server.session.cookie.name= # Session cookie name.
server.session.cookie.path= # Path of the session cookie.
server.session.cookie.secure= # "Secure" flag for the session
cookie.
server.session.persistent=false # Persist session data between
restarts.
server.session.store-dir= # Directory used to store session data.
server.session.timeout= # Session timeout in seconds.
server.session.tracking-modes= # Session tracking modes (one or
more of the following: "cookie", "url", "ssl").
server.ssl.ciphers= # Supported SSL ciphers.
server.ssl.client-auth= # Whether client authentication is wanted
("want") or needed ("need"). Requires a trust store.
server.ssl.enabled= # Enable SSL support.
server.ssl.enabled-protocols= # Enabled SSL protocols.
server.ssl.key-alias= # Alias that identifies the key in the key
store.
server.ssl.key-password= # Password used to access the key in the
key store.
server.ssl.key-store= # Path to the key store that holds the SSL
certificate (typically a jks file).
server.ssl.key-store-password= # Password used to access the key
store.
server.ssl.key-store-provider= # Provider for the key store.
server.ssl.key-store-type= # Type of the key store.
server.ssl.protocol=TLS # SSL protocol to use.
server.ssl.trust-store= # Trust store that holds SSL
certificates.
server.ssl.trust-store-password= # Password used to access the
trust store.
server.ssl.trust-store-provider= # Provider for the trust store.
server.ssl.trust-store-type= # Type of the trust store.
server.tomcat.accept-count= # Maximum queue length for incoming
connection requests when all possible request processing threads
are in use.
```

```
server.tomcat.accesslog.buffered=true # Buffer output such that
it is only flushed periodically.
server.tomcat.accesslog.directory=logs # Directory in which log
files are created. Can be relative to the tomcat base dir or
absolute.
server.tomcat.accesslog.enabled=false # Enable access log.
server.tomcat.accesslog.pattern=common # Format pattern for
access logs.
server.tomcat.accesslog.prefix=access_log # Log file name prefix.
server.tomcat.accesslog.rename-on-rotate=false # Defer inclusion
of the date stamp in the file name until rotate time.
server.tomcat.accesslog.request-attributes-enabled=false # Set
request attributes for IP address, Hostname, protocol and port
used for the request.
server.tomcat.accesslog.rotate=true # Enable access log rotation.
server.tomcat.accesslog.suffix=.log # Log file name suffix.
server.tomcat.additional-tld-skip-patterns= # Comma-separated
list of additional patterns that match jars to ignore for TLD
scanning.
server.tomcat.background-processor-delay=30 # Delay in seconds
between the invocation of backgroundProcess methods.
server.tomcat.basedir= # Tomcat base directory. If not specified
a temporary directory will be used.
server.tomcat.internal-
proxies=10\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}|\\
        192\\.168\\.\\d{1,3}\\.\\d{1,3}|\\
        169\\.254\\.\\d{1,3}\\.\\d{1,3}|\\
        127\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}|\\
        172\\.1[6-9]{1}\\.\\d{1,3}\\.\\d{1,3}|\\
        172\\.2[0-9]{1}\\.\\d{1,3}\\.\\d{1,3}|\\
        172\\.3[0-1]{1}\\.\\d{1,3}\\.\\d{1,3} # regular expression
matching trusted IP addresses.
server.tomcat.max-connections= # Maximum number of connections
that the server will accept and process at any given time.
server.tomcat.max-http-post-size=0 # Maximum size in bytes of the
HTTP post content.
server.tomcat.max-threads=0 # Maximum amount of worker threads.
server.tomcat.min-spare-threads=0 # Minimum amount of worker
threads.
server.tomcat.port-header=X-Forwarded-Port # Name of the HTTP
header used to override the original port value.
server.tomcat.protocol-header= # Header that holds the incoming
protocol, usually named "X-Forwarded-Proto".
```

```
server.tomcat.protocol-header-https-value=https # Value of the
protocol header that indicates that the incoming request uses
SSL.
server.tomcat.redirect-context-root= # Whether requests to the
context root should be redirected by appending a / to the path.
server.tomcat.remote-ip-header= # Name of the http header from
which the remote ip is extracted. For instance `X-FORWARDED-FOR`
server.tomcat.uri-encoding=UTF-8 # Character encoding to use to
decode the URI.
server.undertow.accesslog.dir= # Undertow access log directory.
server.undertow.accesslog.enabled=false # Enable access log.
server.undertow.accesslog.pattern=common # Format pattern for
access logs.
server.undertow.accesslog.prefix=access_log. # Log file name
prefix.
server.undertow.accesslog.rotate=true # Enable access log
rotation.
server.undertow.accesslog.suffix=log # Log file name suffix.
server.undertow.buffer-size= # Size of each buffer in bytes.
server.undertow.buffers-per-region= # Number of buffer per
region.
server.undertow.direct-buffers= # Allocate buffers outside the
Java heap.
server.undertow.io-threads= # Number of I/O threads to create for
the worker.
server.undertow.max-http-post-size=0 # Maximum size in bytes of
the HTTP post content.
server.undertow.worker-threads= # Number of worker threads.

# FREEMARKER (FreeMarkerAutoConfiguration)
spring.freemarker.allow-request-override=false # Set whether
HttpServletRequest attributes are allowed to override (hide)
controller generated model attributes of the same name.
spring.freemarker.allow-session-override=false # Set whether
HttpSession attributes are allowed to override (hide) controller
generated model attributes of the same name.
spring.freemarker.cache=false # Enable template caching.
spring.freemarker.charset=UTF-8 # Template encoding.
spring.freemarker.check-template-location=true # Check that the
templates location exists.
spring.freemarker.content-type=text/html # Content-Type value.
spring.freemarker.enabled=true # Enable MVC view resolution for
this technology.
```

```
spring.freemarker.expose-request-attributes=false # Set whether
all request attributes should be added to the model prior to
merging with the template.
spring.freemarker.expose-session-attributes=false # Set whether
all HttpSession attributes should be added to the model prior to
merging with the template.
spring.freemarker.expose-spring-macro-helpers=true # Set whether
to expose a RequestContext for use by Spring's macro library,
under the name "springMacroRequestContext".
spring.freemarker.prefer-file-system-access=true # Prefer file
system access for template loading. File system access enables
hot detection of template changes.
spring.freemarker.prefix= # Prefix that gets prepended to view
names when building a URL.
spring.freemarker.request-context-attribute= # Name of the
RequestContext attribute for all views.
spring.freemarker.settings.*= # Well-known FreeMarker keys which
will be passed to FreeMarker's Configuration.
spring.freemarker.suffix= # Suffix that gets appended to view
names when building a URL.
spring.freemarker.template-loader-path=classpath:/templates/ #
Comma-separated list of template paths.
spring.freemarker.view-names= # White list of view names that can
be resolved.

# GROOVY TEMPLATES (GroovyTemplateAutoConfiguration)
spring.groovy.template.allow-request-override=false # Set whether
HttpServletRequest attributes are allowed to override (hide)
controller generated model attributes of the same name.
spring.groovy.template.allow-session-override=false # Set whether
HttpSession attributes are allowed to override (hide) controller
generated model attributes of the same name.
spring.groovy.template.cache= # Enable template caching.
spring.groovy.template.charset=UTF-8 # Template encoding.
spring.groovy.template.check-template-location=true # Check that
the templates location exists.
spring.groovy.template.configuration.*= # See
GroovyMarkupConfigurer
spring.groovy.template.content-type=test/html # Content-Type
value.
spring.groovy.template.enabled=true # Enable MVC view resolution
for this technology.
```

```
spring.groovy.template.expose-request-attributes=false # Set
whether all request attributes should be added to the model prior
to merging with the template.
spring.groovy.template.expose-session-attributes=false # Set
whether all HttpSession attributes should be added to the model
prior to merging with the template.
spring.groovy.template.expose-spring-macro-helpers=true # Set
whether to expose a RequestContext for use by Spring's macro
library, under the name "springMacroRequestContext".
spring.groovy.template.prefix= # Prefix that gets prepended to
view names when building a URL.
spring.groovy.template.request-context-attribute= # Name of the
RequestContext attribute for all views.
spring.groovy.template.resource-loader-path=classpath:/templates/
# Template path.
spring.groovy.template.suffix=.tpl # Suffix that gets appended to
view names when building a URL.
spring.groovy.template.view-names= # White list of view names
that can be resolved.

# SPRING HATEOAS (HateoasProperties)
spring.hateoas.use-hal-as-default-json-media-type=true # Specify
if application/hal+json responses should be sent to requests that
accept application/json.

# HTTP message conversion
spring.http.converters.preferred-json-mapper=jackson # Preferred
JSON mapper to use for HTTP message conversion. Set to "gson" to
force the use of Gson when both it and Jackson are on the
classpath.

# HTTP encoding (HttpEncodingProperties)
spring.http.encoding.charset=UTF-8 # Charset of HTTP requests and
responses. Added to the "Content-Type" header if not set
explicitly.
spring.http.encoding.enabled=true # Enable http encoding support.
spring.http.encoding.force= # Force the encoding to the
configured charset on HTTP requests and responses.
spring.http.encoding.force-request= # Force the encoding to the
configured charset on HTTP requests. Defaults to true when
"force" has not been specified.
spring.http.encoding.force-response= # Force the encoding to the
configured charset on HTTP responses.
spring.http.encoding.mapping= # Locale to Encoding mapping.
```

```
# MULTIPART (MultipartProperties)
spring.http.multipart.enabled=true # Enable support of multi-part
uploads.
spring.http.multipart.file-size-threshold=0 # Threshold after
which files will be written to disk. Values can use the suffixed
"MB" or "KB" to indicate a Megabyte or Kilobyte size.
spring.http.multipart.location= # Intermediate location of
uploaded files.
spring.http.multipart.max-file-size=1MB # Max file size. Values
can use the suffixed "MB" or "KB" to indicate a Megabyte or
Kilobyte size.
spring.http.multipart.max-request-size=10MB # Max request size.
Values can use the suffixed "MB" or "KB" to indicate a Megabyte
or Kilobyte size.
spring.http.multipart.resolve-lazily=false # Whether to resolve
the multipart request lazily at the time of file or parameter
access.

# JACKSON (JacksonProperties)
spring.jackson.date-format= # Date format string or a fully-
qualified date format class name. For instance `yyyy-MM-dd
HH:mm:ss`.
spring.jackson.default-property-inclusion= # Controls the
inclusion of properties during serialization.
spring.jackson.deserialization.*= # Jackson on/off features that
affect the way Java objects are deserialized.
spring.jackson.generator.*= # Jackson on/off features for
generators.
spring.jackson.joda-date-time-format= # Joda date time format
string. If not configured, "date-format" will be used as a
fallback if it is configured with a format string.
spring.jackson.locale= # Locale used for formatting.
spring.jackson.mapper.*= # Jackson general purpose on/off
features.
spring.jackson.parser.*= # Jackson on/off features for parsers.
spring.jackson.property-naming-strategy= # One of the constants
on Jackson's PropertyNamingStrategy. Can also be a fully-
qualified class name of a PropertyNamingStrategy subclass.
spring.jackson.serialization.*= # Jackson on/off features that
affect the way Java objects are serialized.
spring.jackson.time-zone= # Time zone used when formatting dates.
For instance `America/Los_Angeles`
```

```
# JERSEY (JerseyProperties)
spring.jersey.application-path= # Path that serves as the base
URI for the application. Overrides the value of
"@ApplicationPath" if specified.
spring.jersey.filter.order=0 # Jersey filter chain order.
spring.jersey.init.*= # Init parameters to pass to Jersey via the
servlet or filter.
spring.jersey.servlet.load-on-startup=-1 # Load on startup
priority of the Jersey servlet.
spring.jersey.type=servlet # Jersey integration type.

# SPRING LDAP (LdapProperties)
spring.ldap.urls= # LDAP URLs of the server.
spring.ldap.base= # Base suffix from which all operations should
originate.
spring.ldap.username= # Login user of the server.
spring.ldap.password= # Login password of the server.
spring.ldap.base-environment.*= # LDAP specification settings.

# EMBEDDED LDAP (EmbeddedLdapProperties)
spring.ldap.embedded.base-dn= # The base DN
spring.ldap.embedded.credential.username= # Embedded LDAP
username.
spring.ldap.embedded.credential.password= # Embedded LDAP
password.
spring.ldap.embedded.ldif=classpath:schema.ldif # Schema (LDIF)
script resource reference.
spring.ldap.embedded.port= # Embedded LDAP port.
spring.ldap.embedded.validation.enabled=true # Enable LDAP schema
validation.
spring.ldap.embedded.validation.schema= # Path to the custom
schema.

# SPRING MOBILE DEVICE VIEWS
(DeviceDelegatingViewResolverAutoConfiguration)
spring.mobile.devicedelegatingviewresolver.enable-fallback=false
# Enable support for fallback resolution.
spring.mobile.devicedelegatingviewresolver.enabled=false # Enable
device view resolver.
spring.mobile.devicedelegatingviewresolver.mobile-prefix=mobile/
# Prefix that gets prepended to view names for mobile devices.
spring.mobile.devicedelegatingviewresolver.mobile-suffix= #
Suffix that gets appended to view names for mobile devices.
```

```
spring.mobile.devicedelegatingviewresolver.normal-prefix= #
Prefix that gets prepended to view names for normal devices.
spring.mobile.devicedelegatingviewresolver.normal-suffix= #
Suffix that gets appended to view names for normal devices.
spring.mobile.devicedelegatingviewresolver.tablet-prefix=tablet/
# Prefix that gets prepended to view names for tablet devices.
spring.mobile.devicedelegatingviewresolver.tablet-suffix= #
Suffix that gets appended to view names for tablet devices.

# SPRING MOBILE SITE PREFERENCE (SitePreferenceAutoConfiguration)
spring.mobile.sitepreference.enabled=true # Enable
SitePreferenceHandler.

# MUSTACHE TEMPLATES (MustacheAutoConfiguration)
spring.mustache.allow-request-override= # Set whether
HttpServletRequest attributes are allowed to override (hide)
controller generated model attributes of the same name.
spring.mustache.allow-session-override= # Set whether HttpSession
attributes are allowed to override (hide) controller generated
model attributes of the same name.
spring.mustache.cache= # Enable template caching.
spring.mustache.charset= # Template encoding.
spring.mustache.check-template-location= # Check that the
templates location exists.
spring.mustache.content-type= # Content-Type value.
spring.mustache.enabled= # Enable MVC view resolution for this
technology.
spring.mustache.expose-request-attributes= # Set whether all
request attributes should be added to the model prior to merging
with the template.
spring.mustache.expose-session-attributes= # Set whether all
HttpSession attributes should be added to the model prior to
merging with the template.
spring.mustache.expose-spring-macro-helpers= # Set whether to
expose a RequestContext for use by Spring's macro library, under
the name "springMacroRequestContext".
spring.mustache.prefix=classpath:/templates/ # Prefix to apply to
template names.
spring.mustache.request-context-attribute= # Name of the
RequestContext attribute for all views.
spring.mustache.suffix=.html # Suffix to apply to template names.
spring.mustache.view-names= # White list of view names that can
be resolved.
```

```
# SPRING MVC (WebMvcProperties)
spring.mvc.async.request-timeout= # Amount of time (in
milliseconds) before asynchronous request handling times out.
spring.mvc.date-format= # Date format to use. For instance
`dd/MM/yyyy`.
spring.mvc.dispatch-trace-request=false # Dispatch TRACE requests
to the FrameworkServlet doService method.
spring.mvc.dispatch-options-request=true # Dispatch OPTIONS
requests to the FrameworkServlet doService method.
spring.mvc.favicon.enabled=true # Enable resolution of
favicon.ico.
spring.mvc.formcontent.putfilter.enabled=true # Enable Spring's
HttpPutFormContentFilter.
spring.mvc.ignore-default-model-on-redirect=true # If the content
of the "default" model should be ignored during redirect
scenarios.
spring.mvc.locale= # Locale to use. By default, this locale is
overridden by the "Accept-Language" header.
spring.mvc.locale-resolver=accept-header # Define how the locale
should be resolved.
spring.mvc.log-resolved-exception=false # Enable warn logging of
exceptions resolved by a "HandlerExceptionResolver".
spring.mvc.media-types.*= # Maps file extensions to media types
for content negotiation.
spring.mvc.message-codes-resolver-format= # Formatting strategy
for message codes. For instance `PREFIX_ERROR_CODE`.
spring.mvc.servlet.load-on-startup=-1 # Load on startup priority
of the Spring Web Services servlet.
spring.mvc.static-path-pattern=/** # Path pattern used for static
resources.
spring.mvc.throw-exception-if-no-handler-found=false # If a
"NoHandlerFoundException" should be thrown if no Handler was
found to process a request.
spring.mvc.view.prefix= # Spring MVC view prefix.
spring.mvc.view.suffix= # Spring MVC view suffix.

# SPRING RESOURCES HANDLING (ResourceProperties)
spring.resources.add-mappings=true # Enable default resource
handling.
spring.resources.cache-period= # Cache period for the resources
served by the resource handler, in seconds.
spring.resources.chain.cache=true # Enable caching in the
Resource chain.
```

```
spring.resources.chain.enabled= # Enable the Spring Resource
Handling chain. Disabled by default unless at least one strategy
has been enabled.
spring.resources.chain.gzipped=false # Enable resolution of
already gzipped resources.
spring.resources.chain.html-application-cache=false # Enable
HTML5 application cache manifest rewriting.
spring.resources.chain.strategy.content.enabled=false # Enable
the content Version Strategy.
spring.resources.chain.strategy.content.paths=/** # Comma-
separated list of patterns to apply to the Version Strategy.
spring.resources.chain.strategy.fixed.enabled=false # Enable the
fixed Version Strategy.
spring.resources.chain.strategy.fixed.paths=/** # Comma-separated
list of patterns to apply to the Version Strategy.
spring.resources.chain.strategy.fixed.version= # Version string
to use for the Version Strategy.
spring.resources.static-locations=classpath:/META-
INF/resources/,classpath:/resources/,classpath:/static/,classpath
:/public/ # Locations of static resources.

# SPRING SESSION (SessionProperties)
spring.session.hazelcast.flush-mode=on-save # Sessions flush
mode.
spring.session.hazelcast.map-name=spring:session:sessions # Name
of the map used to store sessions.
spring.session.jdbc.initializer.enabled= # Create the required
session tables on startup if necessary. Enabled automatically if
the default table name is set or a custom schema is configured.
spring.session.jdbc.schema=classpath:org/springframework/session/
jdbc/schema-@@platform@@.sql # Path to the SQL file to use to
initialize the database schema.
spring.session.jdbc.table-name=SPRING_SESSION # Name of database
table used to store sessions.
spring.session.mongo.collection-name=sessions # Collection name
used to store sessions.
spring.session.redis.flush-mode=on-save # Sessions flush mode.
spring.session.redis.namespace= # Namespace for keys used to
store sessions.
spring.session.store-type= # Session store type.

# SPRING SOCIAL (SocialWebAutoConfiguration)
spring.social.auto-connection-views=false # Enable the connection
status view for supported providers.
```

```
# SPRING SOCIAL FACEBOOK (FacebookAutoConfiguration)
spring.social.facebook.app-id= # your application's Facebook App
ID
spring.social.facebook.app-secret= # your application's Facebook
App Secret

# SPRING SOCIAL LINKEDIN (LinkedInAutoConfiguration)
spring.social.linkedin.app-id= # your application's LinkedIn App
ID
spring.social.linkedin.app-secret= # your application's LinkedIn
App Secret

# SPRING SOCIAL TWITTER (TwitterAutoConfiguration)
spring.social.twitter.app-id= # your application's Twitter App ID
spring.social.twitter.app-secret= # your application's Twitter
App Secret

# THYMELEAF (ThymeleafAutoConfiguration)
spring.thymeleaf.cache=true # Enable template caching.
spring.thymeleaf.check-template=true # Check that the template
exists before rendering it.
spring.thymeleaf.check-template-location=true # Check that the
templates location exists.
spring.thymeleaf.content-type=text/html # Content-Type value.
spring.thymeleaf.enabled=true # Enable MVC Thymeleaf view
resolution.
spring.thymeleaf.encoding=UTF-8 # Template encoding.
spring.thymeleaf.excluded-view-names= # Comma-separated list of
view names that should be excluded from resolution.
spring.thymeleaf.mode=HTML5 # Template mode to be applied to
templates. See also StandardTemplateModeHandlers.
spring.thymeleaf.prefix=classpath:/templates/ # Prefix that gets
prepended to view names when building a URL.
spring.thymeleaf.suffix=.html # Suffix that gets appended to view
names when building a URL.
spring.thymeleaf.template-resolver-order= # Order of the template
resolver in the chain.
spring.thymeleaf.view-names= # Comma-separated list of view names
that can be resolved.

# SPRING WEB SERVICES (WebServicesProperties)
spring.webservices.path=/services # Path that serves as the base
URI for the services.
```

```
spring.webservices.servlet.init= # Servlet init parameters to
pass to Spring Web Services.
spring.webservices.servlet.load-on-startup=-1 # Load on startup
priority of the Spring Web Services servlet.



# ----------------------------------------
# SECURITY PROPERTIES
# ----------------------------------------
# SECURITY (SecurityProperties)
security.basic.authorize-mode=role # Security authorize mode to
apply.
security.basic.enabled=true # Enable basic authentication.
security.basic.path=/** # Comma-separated list of paths to
secure.
security.basic.realm=Spring # HTTP basic realm name.
security.enable-csrf=false # Enable Cross Site Request Forgery
support.
security.filter-order=0 # Security filter chain order.
security.filter-dispatcher-types=ASYNC, FORWARD, INCLUDE, REQUEST
# Security filter chain dispatcher types.
security.headers.cache=true # Enable cache control HTTP headers.
security.headers.content-security-policy= # Value for content
security policy header.
security.headers.content-security-policy-mode=default # Content
security policy mode.
security.headers.content-type=true # Enable "X-Content-Type-
Options" header.
security.headers.frame=true # Enable "X-Frame-Options" header.
security.headers.hsts=all # HTTP Strict Transport Security (HSTS)
mode (none, domain, all).
security.headers.xss=true # Enable cross site scripting (XSS)
protection.
security.ignored= # Comma-separated list of paths to exclude from
the default secured paths.
security.require-ssl=false # Enable secure channel for all
requests.
security.sessions=stateless # Session creation policy (always,
never, if_required, stateless).
security.user.name=user # Default user name.
security.user.password= # Password for the default user name. A
random password is logged on startup by default.
```

```
security.user.role=USER # Granted roles for the default user
name.

# SECURITY OAUTH2 CLIENT (OAuth2ClientProperties)
security.oauth2.client.client-id= # OAuth2 client id.
security.oauth2.client.client-secret= # OAuth2 client secret. A
random secret is generated by default

# SECURITY OAUTH2 RESOURCES (ResourceServerProperties)
security.oauth2.resource.filter-order= # The order of the filter
chain used to authenticate tokens.
security.oauth2.resource.id= # Identifier of the resource.
security.oauth2.resource.jwt.key-uri= # The URI of the JWT token.
Can be set if the value is not available and the key is public.
security.oauth2.resource.jwt.key-value= # The verification key of
the JWT token. Can either be a symmetric secret or PEM-encoded
RSA public key.
security.oauth2.resource.prefer-token-info=true # Use the token
info, can be set to false to use the user info.
security.oauth2.resource.service-id=resource #
security.oauth2.resource.token-info-uri= # URI of the token
decoding endpoint.
security.oauth2.resource.token-type= # The token type to send
when using the userInfoUri.
security.oauth2.resource.user-info-uri= # URI of the user
endpoint.

# SECURITY OAUTH2 SSO (OAuth2SsoProperties)
security.oauth2.sso.filter-order= # Filter order to apply if not
providing an explicit WebSecurityConfigurerAdapter
security.oauth2.sso.login-path=/login # Path to the login page,
i.e. the one that triggers the redirect to the OAuth2
Authorization Server


# ----------------------------------------
# DATA PROPERTIES
# ----------------------------------------

# FLYWAY (FlywayProperties)
flyway.baseline-description= #
flyway.baseline-version=1 # version to start migration
flyway.baseline-on-migrate= #
flyway.check-location=false # Check that migration scripts
location exists.
flyway.clean-on-validation-error= #
flyway.enabled=true # Enable flyway.
flyway.encoding= #
flyway.ignore-failed-future-migration= #
flyway.init-sqls= # SQL statements to execute to initialize a
connection immediately after obtaining it.
flyway.locations=classpath:db/migration # locations of migrations
scripts
flyway.out-of-order= #
flyway.password= # JDBC password if you want Flyway to create its
own DataSource
flyway.placeholder-prefix= #
flyway.placeholder-replacement= #
flyway.placeholder-suffix= #
flyway.placeholders.*= #
flyway.schemas= # schemas to update
flyway.sql-migration-prefix=V #
flyway.sql-migration-separator= #
flyway.sql-migration-suffix=.sql #
flyway.table= #
flyway.url= # JDBC url of the database to migrate. If not set,
the primary configured data source is used.
flyway.user= # Login user of the database to migrate.
flyway.validate-on-migrate= #

# LIQUIBASE (LiquibaseProperties)
liquibase.change-log=classpath:/db/changelog/db.changelog-
master.yaml # Change log configuration path.
liquibase.check-change-log-location=true # Check the change log
location exists.
liquibase.contexts= # Comma-separated list of runtime contexts to
use.
liquibase.default-schema= # Default database schema.
liquibase.drop-first=false # Drop the database schema first.
liquibase.enabled=true # Enable liquibase support.
liquibase.labels= # Comma-separated list of runtime labels to
use.
liquibase.parameters.*= # Change log parameters.
liquibase.password= # Login password of the database to migrate.
liquibase.rollback-file= # File to which rollback SQL will be
written when an update is performed.
```

```
liquibase.url= # JDBC url of the database to migrate. If not set,
the primary configured data source is used.
liquibase.user= # Login user of the database to migrate.

# COUCHBASE (CouchbaseProperties)
spring.couchbase.bootstrap-hosts= # Couchbase nodes (host or IP
address) to bootstrap from.
spring.couchbase.bucket.name=default # Name of the bucket to
connect to.
spring.couchbase.bucket.password=  # Password of the bucket.
spring.couchbase.env.endpoints.key-value=1 # Number of sockets
per node against the Key/value service.
spring.couchbase.env.endpoints.query=1 # Number of sockets per
node against the Query (N1QL) service.
spring.couchbase.env.endpoints.view=1 # Number of sockets per
node against the view service.
spring.couchbase.env.ssl.enabled= # Enable SSL support. Enabled
automatically if a "keyStore" is provided unless specified
otherwise.
spring.couchbase.env.ssl.key-store= # Path to the JVM key store
that holds the certificates.
spring.couchbase.env.ssl.key-store-password= # Password used to
access the key store.
spring.couchbase.env.timeouts.connect=5000 # Bucket connections
timeout in milliseconds.
spring.couchbase.env.timeouts.key-value=2500 # Blocking
operations performed on a specific key timeout in milliseconds.
spring.couchbase.env.timeouts.query=7500 # N1QL query operations
timeout in milliseconds.
spring.couchbase.env.timeouts.socket-connect=1000 # Socket
connect connections timeout in milliseconds.
spring.couchbase.env.timeouts.view=7500 # Regular and geospatial
view operations timeout in milliseconds.

# DAO (PersistenceExceptionTranslationAutoConfiguration)
spring.dao.exceptiontranslation.enabled=true # Enable the
PersistenceExceptionTranslationPostProcessor.

# CASSANDRA (CassandraProperties)
spring.data.cassandra.cluster-name= # Name of the Cassandra
cluster.
spring.data.cassandra.compression=none # Compression supported by
the Cassandra binary protocol.
```

```
spring.data.cassandra.connect-timeout-millis= # Socket option:
connection time out.
spring.data.cassandra.consistency-level= # Queries consistency
level.
spring.data.cassandra.contact-points=localhost # Comma-separated
list of cluster node addresses.
spring.data.cassandra.fetch-size= # Queries default fetch size.
spring.data.cassandra.keyspace-name= # Keyspace name to use.
spring.data.cassandra.load-balancing-policy= # Class name of the
load balancing policy.
spring.data.cassandra.port= # Port of the Cassandra server.
spring.data.cassandra.password= # Login password of the server.
spring.data.cassandra.read-timeout-millis= # Socket option: read
time out.
spring.data.cassandra.reconnection-policy= # Reconnection policy
class.
spring.data.cassandra.retry-policy= # Class name of the retry
policy.
spring.data.cassandra.serial-consistency-level= # Queries serial
consistency level.
spring.data.cassandra.schema-action=none # Schema action to take
at startup.
spring.data.cassandra.ssl=false # Enable SSL support.
spring.data.cassandra.username= # Login user of the server.

# DATA COUCHBASE (CouchbaseDataProperties)
spring.data.couchbase.auto-index=false # Automatically create
views and indexes.
spring.data.couchbase.consistency=read-your-own-writes #
Consistency to apply by default on generated queries.
spring.data.couchbase.repositories.enabled=true # Enable
Couchbase repositories.

# ELASTICSEARCH (ElasticsearchProperties)
spring.data.elasticsearch.cluster-name=elasticsearch #
Elasticsearch cluster name.
spring.data.elasticsearch.cluster-nodes= # Comma-separated list
of cluster node addresses. If not specified, starts a client
node.
spring.data.elasticsearch.properties.*= # Additional properties
used to configure the client.
spring.data.elasticsearch.repositories.enabled=true # Enable
Elasticsearch repositories.
```

```
# DATA LDAP
spring.data.ldap.repositories.enabled=true # Enable LDAP
repositories.

# MONGODB (MongoProperties)
spring.data.mongodb.authentication-database= # Authentication
database name.
spring.data.mongodb.database=test # Database name.
spring.data.mongodb.field-naming-strategy= # Fully qualified name
of the FieldNamingStrategy to use.
spring.data.mongodb.grid-fs-database= # GridFS database name.
spring.data.mongodb.host=localhost # Mongo server host. Cannot be
set with uri.
spring.data.mongodb.password= # Login password of the mongo
server. Cannot be set with uri.
spring.data.mongodb.port=27017 # Mongo server port. Cannot be set
with uri.
spring.data.mongodb.repositories.enabled=true # Enable Mongo
repositories.
spring.data.mongodb.uri=mongodb://localhost/test # Mongo database
URI. Cannot be set with host, port and credentials.
spring.data.mongodb.username= # Login user of the mongo server.
Cannot be set with uri.

# DATA REDIS
spring.data.redis.repositories.enabled=true # Enable Redis
repositories.

# NEO4J (Neo4jProperties)
spring.data.neo4j.compiler= # Compiler to use.
spring.data.neo4j.embedded.enabled=true # Enable embedded mode if
the embedded driver is available.
spring.data.neo4j.open-in-view=false # Register
OpenSessionInViewInterceptor. Binds a Neo4j Session to the thread
for the entire processing of the request.
spring.data.neo4j.password= # Login password of the server.
spring.data.neo4j.repositories.enabled=true # Enable Neo4j
repositories.
spring.data.neo4j.uri= # URI used by the driver. Auto-detected by
default.
spring.data.neo4j.username= # Login user of the server.

# DATA REST (RepositoryRestProperties)
```

```
spring.data.rest.base-path= # Base path to be used by Spring Data
REST to expose repository resources.
spring.data.rest.default-page-size= # Default size of pages.
spring.data.rest.detection-strategy=default # Strategy to use to
determine which repositories get exposed.
spring.data.rest.enable-enum-translation= # Enable enum value
translation via the Spring Data REST default resource bundle.
spring.data.rest.limit-param-name= # Name of the URL query string
parameter that indicates how many results to return at once.
spring.data.rest.max-page-size= # Maximum size of pages.
spring.data.rest.page-param-name= # Name of the URL query string
parameter that indicates what page to return.
spring.data.rest.return-body-on-create= # Return a response body
after creating an entity.
spring.data.rest.return-body-on-update= # Return a response body
after updating an entity.
spring.data.rest.sort-param-name= # Name of the URL query string
parameter that indicates what direction to sort results.

# SOLR (SolrProperties)
spring.data.solr.host=http://127.0.0.1:8983/solr # Solr host.
Ignored if "zk-host" is set.
spring.data.solr.repositories.enabled=true # Enable Solr
repositories.
spring.data.solr.zk-host= # ZooKeeper host address in the form
HOST:PORT.

# DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.continue-on-error=false # Do not stop if an
error occurs while initializing the database.
spring.datasource.data= # Data (DML) script resource references.
spring.datasource.data-username= # User of the database to
execute DML scripts (if different).
spring.datasource.data-password= # Password of the database to
execute DML scripts (if different).
spring.datasource.dbcp2.*= # Commons DBCP2 specific settings
spring.datasource.driver-class-name= # Fully qualified name of
the JDBC driver. Auto-detected based on the URL by default.
spring.datasource.generate-unique-name=false # Generate a random
datasource name.
spring.datasource.hikari.*= # Hikari specific settings
spring.datasource.initialize=true # Populate the database using
'data.sql'.
```

```
spring.datasource.jmx-enabled=false # Enable JMX support (if
provided by the underlying pool).
spring.datasource.jndi-name= # JNDI location of the datasource.
Class, url, username & password are ignored when set.
spring.datasource.name=testdb # Name of the datasource.
spring.datasource.password= # Login password of the database.
spring.datasource.platform=all # Platform to use in the schema
resource (schema-${platform}.sql).
spring.datasource.schema= # Schema (DDL) script resource
references.
spring.datasource.schema-username= # User of the database to
execute DDL scripts (if different).
spring.datasource.schema-password= # Password of the database to
execute DDL scripts (if different).
spring.datasource.separator=; # Statement separator in SQL
initialization scripts.
spring.datasource.sql-script-encoding= # SQL scripts encoding.
spring.datasource.tomcat.*= # Tomcat datasource specific settings
spring.datasource.type= # Fully qualified name of the connection
pool implementation to use. By default, it is auto-detected from
the classpath.
spring.datasource.url= # JDBC url of the database.
spring.datasource.username=

# JEST (Elasticsearch HTTP client) (JestProperties)
spring.elasticsearch.jest.connection-timeout=3000 # Connection
timeout in milliseconds.
spring.elasticsearch.jest.multi-threaded=true # Enable connection
requests from multiple execution threads.
spring.elasticsearch.jest.password= # Login password.
spring.elasticsearch.jest.proxy.host= # Proxy host the HTTP
client should use.
spring.elasticsearch.jest.proxy.port= # Proxy port the HTTP
client should use.
spring.elasticsearch.jest.read-timeout=3000 # Read timeout in
milliseconds.
spring.elasticsearch.jest.uris=http://localhost:9200 # Comma-
separated list of the Elasticsearch instances to use.
spring.elasticsearch.jest.username= # Login user.

# H2 Web Console (H2ConsoleProperties)
spring.h2.console.enabled=false # Enable the console.
spring.h2.console.path=/h2-console # Path at which the console
will be available.
```

```
spring.h2.console.settings.trace=false # Enable trace output.
spring.h2.console.settings.web-allow-others=false # Enable remote
access.

# JOOQ (JooqAutoConfiguration)
spring.jooq.sql-dialect= # SQLDialect JOOQ used when
communicating with the configured datasource. For instance
`POSTGRES`

# JPA (JpaBaseConfiguration, HibernateJpaAutoConfiguration)
spring.data.jpa.repositories.enabled=true # Enable JPA
repositories.
spring.jpa.database= # Target database to operate on, auto-
detected by default. Can be alternatively set using the
"databasePlatform" property.
spring.jpa.database-platform= # Name of the target database to
operate on, auto-detected by default. Can be alternatively set
using the "Database" enum.
spring.jpa.generate-ddl=false # Initialize the schema on startup.
spring.jpa.hibernate.ddl-auto= # DDL mode. This is actually a
shortcut for the "hibernate.hbm2ddl.auto" property. Default to
"create-drop" when using an embedded database, "none" otherwise.
spring.jpa.hibernate.naming.implicit-strategy= # Hibernate 5
implicit naming strategy fully qualified name.
spring.jpa.hibernate.naming.physical-strategy= # Hibernate 5
physical naming strategy fully qualified name.
spring.jpa.hibernate.naming.strategy= # Hibernate 4 naming
strategy fully qualified name. Not supported with Hibernate 5.
spring.jpa.hibernate.use-new-id-generator-mappings= # Use
Hibernate's newer IdentifierGenerator for AUTO, TABLE and
SEQUENCE.
spring.jpa.open-in-view=true # Register
OpenEntityManagerInViewInterceptor. Binds a JPA EntityManager to
the thread for the entire processing of the request.
spring.jpa.properties.*= # Additional native properties to set on
the JPA provider.
spring.jpa.show-sql=false # Enable logging of SQL statements.

# JTA (JtaAutoConfiguration)
spring.jta.enabled=true # Enable JTA support.
spring.jta.log-dir= # Transaction logs directory.
spring.jta.transaction-manager-id= # Transaction manager unique
identifier.
```

```
# ATOMIKOS (AtomikosProperties)
spring.jta.atomikos.connectionfactory.borrow-connection-
timeout=30 # Timeout, in seconds, for borrowing connections from
the pool.
spring.jta.atomikos.connectionfactory.ignore-session-transacted-
flag=true # Whether or not to ignore the transacted flag when
creating session.
spring.jta.atomikos.connectionfactory.local-transaction-
mode=false # Whether or not local transactions are desired.
spring.jta.atomikos.connectionfactory.maintenance-interval=60 #
The time, in seconds, between runs of the pool's maintenance
thread.
spring.jta.atomikos.connectionfactory.max-idle-time=60 # The
time, in seconds, after which connections are cleaned up from the
pool.
spring.jta.atomikos.connectionfactory.max-lifetime=0 # The time,
in seconds, that a connection can be pooled for before being
destroyed. 0 denotes no limit.
spring.jta.atomikos.connectionfactory.max-pool-size=1 # The
maximum size of the pool.
spring.jta.atomikos.connectionfactory.min-pool-size=1 # The
minimum size of the pool.
spring.jta.atomikos.connectionfactory.reap-timeout=0 # The reap
timeout, in seconds, for borrowed connections. 0 denotes no
limit.
spring.jta.atomikos.connectionfactory.unique-resource-
name=jmsConnectionFactory # The unique name used to identify the
resource during recovery.
spring.jta.atomikos.datasource.borrow-connection-timeout=30 #
Timeout, in seconds, for borrowing connections from the pool.
spring.jta.atomikos.datasource.default-isolation-level= # Default
isolation level of connections provided by the pool.
spring.jta.atomikos.datasource.login-timeout= # Timeout, in
seconds, for establishing a database connection.
spring.jta.atomikos.datasource.maintenance-interval=60 # The
time, in seconds, between runs of the pool's maintenance thread.
spring.jta.atomikos.datasource.max-idle-time=60 # The time, in
seconds, after which connections are cleaned up from the pool.
spring.jta.atomikos.datasource.max-lifetime=0 # The time, in
seconds, that a connection can be pooled for before being
destroyed. 0 denotes no limit.
spring.jta.atomikos.datasource.max-pool-size=1 # The maximum size
of the pool.
```

```
spring.jta.atomikos.datasource.min-pool-size=1 # The minimum size
of the pool.
spring.jta.atomikos.datasource.reap-timeout=0 # The reap timeout,
in seconds, for borrowed connections. 0 denotes no limit.
spring.jta.atomikos.datasource.test-query= # SQL query or
statement used to validate a connection before returning it.
spring.jta.atomikos.datasource.unique-resource-name=dataSource #
The unique name used to identify the resource during recovery.
spring.jta.atomikos.properties.checkpoint-interval=500 # Interval
between checkpoints.
spring.jta.atomikos.properties.console-file-count=1 # Number of
debug logs files that can be created.
spring.jta.atomikos.properties.console-file-limit=-1 # How many
bytes can be stored at most in debug logs files.
spring.jta.atomikos.properties.console-file-name=tm.out # Debug
logs file name.
spring.jta.atomikos.properties.console-log-level=warn # Console
log level.
spring.jta.atomikos.properties.default-jta-timeout=10000 #
Default timeout for JTA transactions.
spring.jta.atomikos.properties.enable-logging=true # Enable disk
logging.
spring.jta.atomikos.properties.force-shutdown-on-vm-exit=false #
Specify if a VM shutdown should trigger forced shutdown of the
transaction core.
spring.jta.atomikos.properties.log-base-dir= # Directory in which
the log files should be stored.
spring.jta.atomikos.properties.log-base-name=tmlog # Transactions
log file base name.
spring.jta.atomikos.properties.max-actives=50 # Maximum number of
active transactions.
spring.jta.atomikos.properties.max-timeout=300000 # Maximum
timeout (in milliseconds) that can be allowed for transactions.
spring.jta.atomikos.properties.output-dir= # Directory in which
to store the debug log files.
spring.jta.atomikos.properties.serial-jta-transactions=true #
Specify if sub-transactions should be joined when possible.
spring.jta.atomikos.properties.service= # Transaction manager
implementation that should be started.
spring.jta.atomikos.properties.threaded-two-phase-commit=true #
Use different (and concurrent) threads for two-phase commit on
the participating resources.
spring.jta.atomikos.properties.transaction-manager-unique-name= #
Transaction manager's unique name.
```

```
# BITRONIX
spring.jta.bitronix.connectionfactory.acquire-increment=1 #
Number of connections to create when growing the pool.
spring.jta.bitronix.connectionfactory.acquisition-interval=1 #
Time, in seconds, to wait before trying to acquire a connection
again after an invalid connection was acquired.
spring.jta.bitronix.connectionfactory.acquisition-timeout=30 #
Timeout, in seconds, for acquiring connections from the pool.
spring.jta.bitronix.connectionfactory.allow-local-
transactions=true # Whether or not the transaction manager should
allow mixing XA and non-XA transactions.
spring.jta.bitronix.connectionfactory.apply-transaction-
timeout=false # Whether or not the transaction timeout should be
set on the XAResource when it is enlisted.
spring.jta.bitronix.connectionfactory.automatic-enlisting-
enabled=true # Whether or not resources should be enlisted and
delisted automatically.
spring.jta.bitronix.connectionfactory.cache-producers-
consumers=true # Whether or not produces and consumers should be
cached.
spring.jta.bitronix.connectionfactory.defer-connection-
release=true # Whether or not the provider can run many
transactions on the same connection and supports transaction
interleaving.
spring.jta.bitronix.connectionfactory.ignore-recovery-
failures=false # Whether or not recovery failures should be
ignored.
spring.jta.bitronix.connectionfactory.max-idle-time=60 # The
time, in seconds, after which connections are cleaned up from the
pool.
spring.jta.bitronix.connectionfactory.max-pool-size=10 # The
maximum size of the pool. 0 denotes no limit.
spring.jta.bitronix.connectionfactory.min-pool-size=0 # The
minimum size of the pool.
spring.jta.bitronix.connectionfactory.password= # The password to
use to connect to the JMS provider.
spring.jta.bitronix.connectionfactory.share-transaction-
connections=false #  Whether or not connections in the ACCESSIBLE
state can be shared within the context of a transaction.
spring.jta.bitronix.connectionfactory.test-connections=true #
Whether or not connections should be tested when acquired from
the pool.
```

```
spring.jta.bitronix.connectionfactory.two-pc-ordering-position=1
# The position that this resource should take during two-phase
commit (always first is Integer.MIN_VALUE, always last is
Integer.MAX_VALUE).
spring.jta.bitronix.connectionfactory.unique-
name=jmsConnectionFactory # The unique name used to identify the
resource during recovery.
spring.jta.bitronix.connectionfactory.use-tm-join=true Whether or
not TMJOIN should be used when starting XAResources.
spring.jta.bitronix.connectionfactory.user= # The user to use to
connect to the JMS provider.
spring.jta.bitronix.datasource.acquire-increment=1 # Number of
connections to create when growing the pool.
spring.jta.bitronix.datasource.acquisition-interval=1 # Time, in
seconds, to wait before trying to acquire a connection again
after an invalid connection was acquired.
spring.jta.bitronix.datasource.acquisition-timeout=30 # Timeout,
in seconds, for acquiring connections from the pool.
spring.jta.bitronix.datasource.allow-local-transactions=true #
Whether or not the transaction manager should allow mixing XA and
non-XA transactions.
spring.jta.bitronix.datasource.apply-transaction-timeout=false #
Whether or not the transaction timeout should be set on the
XAResource when it is enlisted.
spring.jta.bitronix.datasource.automatic-enlisting-enabled=true #
Whether or not resources should be enlisted and delisted
automatically.
spring.jta.bitronix.datasource.cursor-holdability= # The default
cursor holdability for connections.
spring.jta.bitronix.datasource.defer-connection-release=true #
Whether or not the database can run many transactions on the same
connection and supports transaction interleaving.
spring.jta.bitronix.datasource.enable-jdbc4-connection-test= #
Whether or not Connection.isValid() is called when acquiring a
connection from the pool.
spring.jta.bitronix.datasource.ignore-recovery-failures=false #
Whether or not recovery failures should be ignored.
spring.jta.bitronix.datasource.isolation-level= # The default
isolation level for connections.
spring.jta.bitronix.datasource.local-auto-commit= # The default
auto-commit mode for local transactions.
spring.jta.bitronix.datasource.login-timeout= # Timeout, in
seconds, for establishing a database connection.
```

```
spring.jta.bitronix.datasource.max-idle-time=60 # The time, in
seconds, after which connections are cleaned up from the pool.
spring.jta.bitronix.datasource.max-pool-size=10 # The maximum
size of the pool. 0 denotes no limit.
spring.jta.bitronix.datasource.min-pool-size=0 # The minimum size
of the pool.
spring.jta.bitronix.datasource.prepared-statement-cache-size=0 #
The target size of the prepared statement cache. 0 disables the
cache.
spring.jta.bitronix.datasource.share-transaction-
connections=false #  Whether or not connections in the ACCESSIBLE
state can be shared within the context of a transaction.
spring.jta.bitronix.datasource.test-query= # SQL query or
statement used to validate a connection before returning it.
spring.jta.bitronix.datasource.two-pc-ordering-position=1 # The
position that this resource should take during two-phase commit
(always first is Integer.MIN_VALUE, always last is
Integer.MAX_VALUE).
spring.jta.bitronix.datasource.unique-name=dataSource # The
unique name used to identify the resource during recovery.
spring.jta.bitronix.datasource.use-tm-join=true Whether or not
TMJOIN should be used when starting XAResources.
spring.jta.bitronix.properties.allow-multiple-lrc=false # Allow
multiple LRC resources to be enlisted into the same transaction.
spring.jta.bitronix.properties.asynchronous2-pc=false # Enable
asynchronously execution of two phase commit.
spring.jta.bitronix.properties.background-recovery-interval-
seconds=60 # Interval in seconds at which to run the recovery
process in the background.
spring.jta.bitronix.properties.current-node-only-recovery=true #
Recover only the current node.
spring.jta.bitronix.properties.debug-zero-resource-
transaction=false # Log the creation and commit call stacks of
transactions executed without a single enlisted resource.
spring.jta.bitronix.properties.default-transaction-timeout=60 #
Default transaction timeout in seconds.
spring.jta.bitronix.properties.disable-jmx=false # Enable JMX
support.
spring.jta.bitronix.properties.exception-analyzer= # Set the
fully qualified name of the exception analyzer implementation to
use.
spring.jta.bitronix.properties.filter-log-status=false # Enable
filtering of logs so that only mandatory logs are written.
```

```
spring.jta.bitronix.properties.force-batching-enabled=true #  Set
if disk forces are batched.
spring.jta.bitronix.properties.forced-write-enabled=true # Set if
logs are forced to disk.
spring.jta.bitronix.properties.graceful-shutdown-interval=60 #
Maximum amount of seconds the TM will wait for transactions to
get done before aborting them at shutdown time.
spring.jta.bitronix.properties.jndi-transaction-synchronization-
registry-name= # JNDI name of the
TransactionSynchronizationRegistry.
spring.jta.bitronix.properties.jndi-user-transaction-name= # JNDI
name of the UserTransaction.
spring.jta.bitronix.properties.journal=disk # Name of the
journal. Can be 'disk', 'null' or a class name.
spring.jta.bitronix.properties.log-part1-filename=btm1.tlog #
Name of the first fragment of the journal.
spring.jta.bitronix.properties.log-part2-filename=btm2.tlog #
Name of the second fragment of the journal.
spring.jta.bitronix.properties.max-log-size-in-mb=2 # Maximum
size in megabytes of the journal fragments.
spring.jta.bitronix.properties.resource-configuration-filename= #
ResourceLoader configuration file name.
spring.jta.bitronix.properties.server-id= # ASCII ID that must
uniquely identify this TM instance. Default to the machine's IP
address.
spring.jta.bitronix.properties.skip-corrupted-logs=false # Skip
corrupted transactions log entries.
spring.jta.bitronix.properties.warn-about-zero-resource-
transaction=true # Log a warning for transactions executed
without a single enlisted resource.

# NARAYANA (NarayanaProperties)
spring.jta.narayana.default-timeout=60 # Transaction timeout in
seconds.
spring.jta.narayana.expiry-
scanners=com.arjuna.ats.internal.arjuna.recovery.ExpiredTransacti
onStatusManagerScanner # Comma-separated list of expiry scanners.
spring.jta.narayana.log-dir= # Transaction object store
directory.
spring.jta.narayana.one-phase-commit=true # Enable one phase
commit optimisation.
spring.jta.narayana.periodic-recovery-period=120 # Interval in
which periodic recovery scans are performed in seconds.
```

```
spring.jta.narayana.recovery-backoff-period=10 # Back off period
between first and second phases of the recovery scan in seconds.
spring.jta.narayana.recovery-db-pass= # Database password to be
used by recovery manager.
spring.jta.narayana.recovery-db-user= # Database username to be
used by recovery manager.
spring.jta.narayana.recovery-jms-pass= # JMS password to be used
by recovery manager.
spring.jta.narayana.recovery-jms-user= # JMS username to be used
by recovery manager.
spring.jta.narayana.recovery-modules= # Comma-separated list of
recovery modules.
spring.jta.narayana.transaction-manager-id=1 # Unique transaction
manager id.
spring.jta.narayana.xa-resource-orphan-filters= # Comma-separated
list of orphan filters.

# EMBEDDED MONGODB (EmbeddedMongoProperties)
spring.mongodb.embedded.features=SYNC_DELAY # Comma-separated
list of features to enable.
spring.mongodb.embedded.storage.database-dir= # Directory used
for data storage.
spring.mongodb.embedded.storage.oplog-size= # Maximum size of the
oplog in megabytes.
spring.mongodb.embedded.storage.repl-set-name= # Name of the
replica set.
spring.mongodb.embedded.version=2.6.10 # Version of Mongo to use.

# REDIS (RedisProperties)
spring.redis.cluster.max-redirects= # Maximum number of redirects
to follow when executing commands across the cluster.
spring.redis.cluster.nodes= # Comma-separated list of "host:port"
pairs to bootstrap from.
spring.redis.database=0 # Database index used by the connection
factory.
spring.redis.url= # Connection URL, will override host, port and
password (user will be ignored), e.g.
redis://user:password@example.com:6379
spring.redis.host=localhost # Redis server host.
spring.redis.password= # Login password of the redis server.
spring.redis.ssl=false # Enable SSL support.
spring.redis.pool.max-active=8 # Max number of connections that
can be allocated by the pool at a given time. Use a negative
value for no limit.
```

```
spring.redis.pool.max-idle=8 # Max number of "idle" connections
in the pool. Use a negative value to indicate an unlimited number
of idle connections.
spring.redis.pool.max-wait=-1 # Maximum amount of time (in
milliseconds) a connection allocation should block before
throwing an exception when the pool is exhausted. Use a negative
value to block indefinitely.
spring.redis.pool.min-idle=0 # Target for the minimum number of
idle connections to maintain in the pool. This setting only has
an effect if it is positive.
spring.redis.port=6379 # Redis server port.
spring.redis.sentinel.master= # Name of Redis server.
spring.redis.sentinel.nodes= # Comma-separated list of host:port
pairs.
spring.redis.timeout=0 # Connection timeout in milliseconds.

# TRANSACTION (TransactionProperties)
spring.transaction.default-timeout= # Default transaction timeout
in seconds.
spring.transaction.rollback-on-commit-failure= # Perform the
rollback on commit failures.



# ----------------------------------------
# INTEGRATION PROPERTIES
# ----------------------------------------

# ACTIVEMQ (ActiveMQProperties)
spring.activemq.broker-url= # URL of the ActiveMQ broker. Auto-
generated by default. For instance `tcp://localhost:61616`
spring.activemq.in-memory=true # Specify if the default broker
URL should be in memory. Ignored if an explicit broker has been
specified.
spring.activemq.password= # Login password of the broker.
spring.activemq.user= # Login user of the broker.
spring.activemq.packages.trust-all=false # Trust all packages.
spring.activemq.packages.trusted= # Comma-separated list of
specific packages to trust (when not trusting all packages).
spring.activemq.pool.configuration.*= # See
PooledConnectionFactory.
spring.activemq.pool.enabled=false # Whether a
PooledConnectionFactory should be created instead of a regular
ConnectionFactory.
```

```
spring.activemq.pool.expiry-timeout=0 # Connection expiration
timeout in milliseconds.
spring.activemq.pool.idle-timeout=30000 # Connection idle timeout
in milliseconds.
spring.activemq.pool.max-connections=1 # Maximum number of pooled
connections.

# ARTEMIS (ArtemisProperties)
spring.artemis.embedded.cluster-password= # Cluster password.
Randomly generated on startup by default.
spring.artemis.embedded.data-directory= # Journal file directory.
Not necessary if persistence is turned off.
spring.artemis.embedded.enabled=true # Enable embedded mode if
the Artemis server APIs are available.
spring.artemis.embedded.persistent=false # Enable persistent
store.
spring.artemis.embedded.queues= # Comma-separated list of queues
to create on startup.
spring.artemis.embedded.server-id= # Server id. By default, an
auto-incremented counter is used.
spring.artemis.embedded.topics= # Comma-separated list of topics
to create on startup.
spring.artemis.host=localhost # Artemis broker host.
spring.artemis.mode= # Artemis deployment mode, auto-detected by
default.
spring.artemis.password= # Login password of the broker.
spring.artemis.port=61616 # Artemis broker port.
spring.artemis.user= # Login user of the broker.

# SPRING BATCH (BatchProperties)
spring.batch.initializer.enabled= # Create the required batch
tables on startup if necessary. Enabled automatically if no
custom table prefix is set or if a custom schema is configured.
spring.batch.job.enabled=true # Execute all Spring Batch jobs in
the context on startup.
spring.batch.job.names= # Comma-separated list of job names to
execute on startup (For instance `job1,job2`). By default, all
Jobs found in the context are executed.
spring.batch.schema=classpath:org/springframework/batch/core/sche
ma-@@platform@@.sql # Path to the SQL file to use to initialize
the database schema.
spring.batch.table-prefix= # Table prefix for all the batch meta-
data tables.
```

```
# JMS (JmsProperties)
spring.jms.jndi-name= # Connection factory JNDI name. When set,
takes precedence to others connection factory auto-
configurations.
spring.jms.listener.acknowledge-mode= # Acknowledge mode of the
container. By default, the listener is transacted with automatic
acknowledgment.
spring.jms.listener.auto-startup=true # Start the container
automatically on startup.
spring.jms.listener.concurrency= # Minimum number of concurrent
consumers.
spring.jms.listener.max-concurrency= # Maximum number of
concurrent consumers.
spring.jms.pub-sub-domain=false # Specify if the default
destination type is topic.
spring.jms.template.default-destination= # Default destination to
use on send/receive operations that do not have a destination
parameter.
spring.jms.template.delivery-delay= # Delivery delay to use for
send calls in milliseconds.
spring.jms.template.delivery-mode= # Delivery mode. Enable QoS
when set.
spring.jms.template.priority= # Priority of a message when
sending. Enable QoS when set.
spring.jms.template.qos-enabled= # Enable explicit QoS when
sending a message.
spring.jms.template.receive-timeout= # Timeout to use for receive
calls in milliseconds.
spring.jms.template.time-to-live= # Time-to-live of a message
when sending in milliseconds. Enable QoS when set.

# APACHE KAFKA (KafkaProperties)
spring.kafka.bootstrap-servers= # Comma-delimited list of
host:port pairs to use for establishing the initial connection to
the Kafka cluster.
spring.kafka.client-id= # Id to pass to the server when making
requests; used for server-side logging.
spring.kafka.consumer.auto-commit-interval= # Frequency in
milliseconds that the consumer offsets are auto-committed to
Kafka if 'enable.auto.commit' true.
spring.kafka.consumer.auto-offset-reset= # What to do when there
is no initial offset in Kafka or if the current offset does not
exist any more on the server.
```

```
spring.kafka.consumer.bootstrap-servers= # Comma-delimited list
of host:port pairs to use for establishing the initial connection
to the Kafka cluster.
spring.kafka.consumer.client-id= # Id to pass to the server when
making requests; used for server-side logging.
spring.kafka.consumer.enable-auto-commit= # If true the
consumer's offset will be periodically committed in the
background.
spring.kafka.consumer.fetch-max-wait= # Maximum amount of time in
milliseconds the server will block before answering the fetch
request if there isn't sufficient data to immediately satisfy the
requirement given by "fetch.min.bytes".
spring.kafka.consumer.fetch-min-size= # Minimum amount of data
the server should return for a fetch request in bytes.
spring.kafka.consumer.group-id= # Unique string that identifies
the consumer group this consumer belongs to.
spring.kafka.consumer.heartbeat-interval= # Expected time in
milliseconds between heartbeats to the consumer coordinator.
spring.kafka.consumer.key-deserializer= # Deserializer class for
keys.
spring.kafka.consumer.max-poll-records= # Maximum number of
records returned in a single call to poll().
spring.kafka.consumer.value-deserializer= # Deserializer class
for values.
spring.kafka.listener.ack-count= # Number of records between
offset commits when ackMode is "COUNT" or "COUNT_TIME".
spring.kafka.listener.ack-mode= # Listener AckMode; see the
spring-kafka documentation.
spring.kafka.listener.ack-time= # Time in milliseconds between
offset commits when ackMode is "TIME" or "COUNT_TIME".
spring.kafka.listener.concurrency= # Number of threads to run in
the listener containers.
spring.kafka.listener.poll-timeout= # Timeout in milliseconds to
use when polling the consumer.
spring.kafka.producer.acks= # Number of acknowledgments the
producer requires the leader to have received before considering
a request complete.
spring.kafka.producer.batch-size= # Number of records to batch
before sending.
spring.kafka.producer.bootstrap-servers= # Comma-delimited list
of host:port pairs to use for establishing the initial connection
to the Kafka cluster.
```

```
spring.kafka.producer.buffer-memory= # Total bytes of memory the
producer can use to buffer records waiting to be sent to the
server.
spring.kafka.producer.client-id= # Id to pass to the server when
making requests; used for server-side logging.
spring.kafka.producer.compression-type= # Compression type for
all data generated by the producer.
spring.kafka.producer.key-serializer= # Serializer class for
keys.
spring.kafka.producer.retries= # When greater than zero, enables
retrying of failed sends.
spring.kafka.producer.value-serializer= # Serializer class for
values.
spring.kafka.properties.*= # Additional properties used to
configure the client.
spring.kafka.ssl.key-password= # Password of the private key in
the key store file.
spring.kafka.ssl.keystore-location= # Location of the key store
file.
spring.kafka.ssl.keystore-password= # Store password for the key
store file.
spring.kafka.ssl.truststore-location= # Location of the trust
store file.
spring.kafka.ssl.truststore-password= # Store password for the
trust store file.
spring.kafka.template.default-topic= # Default topic to which
messages will be sent.

# RABBIT (RabbitProperties)
spring.rabbitmq.addresses= # Comma-separated list of addresses to
which the client should connect.
spring.rabbitmq.cache.channel.checkout-timeout= # Number of
milliseconds to wait to obtain a channel if the cache size has
been reached.
spring.rabbitmq.cache.channel.size= # Number of channels to
retain in the cache.
spring.rabbitmq.cache.connection.mode=channel # Connection
factory cache mode.
spring.rabbitmq.cache.connection.size= # Number of connections to
cache.
spring.rabbitmq.connection-timeout= # Connection timeout, in
milliseconds; zero for infinite.
spring.rabbitmq.dynamic=true # Create an AmqpAdmin bean.
spring.rabbitmq.host=localhost # RabbitMQ host.
spring.rabbitmq.listener.acknowledge-mode= # Acknowledge mode of
container.
spring.rabbitmq.listener.auto-startup=true # Start the container
automatically on startup.
spring.rabbitmq.listener.concurrency= # Minimum number of
consumers.
spring.rabbitmq.listener.default-requeue-rejected= # Whether or
not to requeue delivery failures; default `true`.
spring.rabbitmq.listener.idle-event-interval= # How often idle
container events should be published in milliseconds.
spring.rabbitmq.listener.max-concurrency= # Maximum number of
consumers.
spring.rabbitmq.listener.prefetch= # Number of messages to be
handled in a single request. It should be greater than or equal
to the transaction size (if used).
spring.rabbitmq.listener.retry.enabled=false # Whether or not
publishing retries are enabled.
spring.rabbitmq.listener.retry.initial-interval=1000 # Interval
between the first and second attempt to deliver a message.
spring.rabbitmq.listener.retry.max-attempts=3 # Maximum number of
attempts to deliver a message.
spring.rabbitmq.listener.retry.max-interval=10000 # Maximum
interval between attempts.
spring.rabbitmq.listener.retry.multiplier=1.0 # A multiplier to
apply to the previous delivery retry interval.
spring.rabbitmq.listener.retry.stateless=true # Whether or not
retry is stateless or stateful.
spring.rabbitmq.listener.transaction-size= # Number of messages
to be processed in a transaction. For best results it should be
less than or equal to the prefetch count.
spring.rabbitmq.password= # Login to authenticate against the
broker.
spring.rabbitmq.port=5672 # RabbitMQ port.
spring.rabbitmq.publisher-confirms=false # Enable publisher
confirms.
spring.rabbitmq.publisher-returns=false # Enable publisher
returns.
spring.rabbitmq.requested-heartbeat= # Requested heartbeat
timeout, in seconds; zero for none.
spring.rabbitmq.ssl.enabled=false # Enable SSL support.
spring.rabbitmq.ssl.key-store= # Path to the key store that holds
the SSL certificate.
spring.rabbitmq.ssl.key-store-password= # Password used to access
the key store.
```

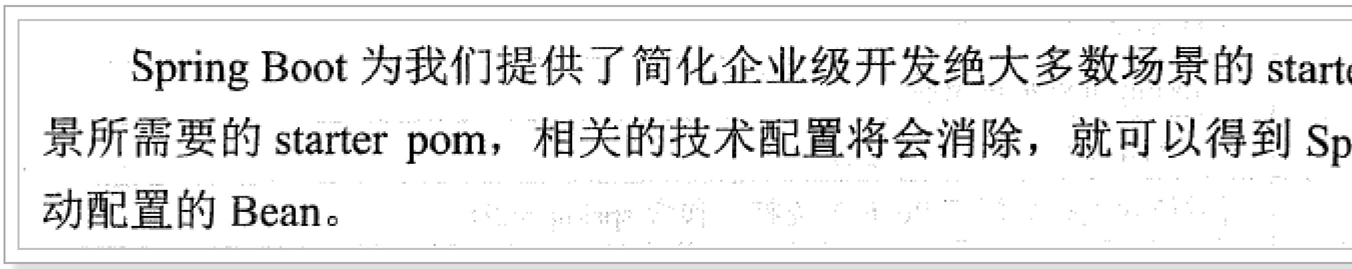
```
spring.rabbitmq.ssl.trust-store= # Trust store that holds SSL
certificates.
spring.rabbitmq.ssl.trust-store-password= # Password used to
access the trust store.
spring.rabbitmq.ssl.algorithm= # SSL algorithm to use. By default
configure by the rabbit client library.
spring.rabbitmq.template.mandatory=false # Enable mandatory
messages.
spring.rabbitmq.template.receive-timeout=0 # Timeout for
`receive()` methods.
spring.rabbitmq.template.reply-timeout=5000 # Timeout for
`sendAndReceive()` methods.
spring.rabbitmq.template.retry.enabled=false # Set to true to
enable retries in the `RabbitTemplate`.
spring.rabbitmq.template.retry.initial-interval=1000 # Interval
between the first and second attempt to publish a message.
spring.rabbitmq.template.retry.max-attempts=3 # Maximum number of
attempts to publish a message.
spring.rabbitmq.template.retry.max-interval=10000 # Maximum
number of attempts to publish a message.
spring.rabbitmq.template.retry.multiplier=1.0 # A multiplier to
apply to the previous publishing retry interval.
spring.rabbitmq.username= # Login user to authenticate to the
broker.
spring.rabbitmq.virtual-host= # Virtual host to use when
connecting to the broker.


# ----------------------------------------
# ACTUATOR PROPERTIES
# ----------------------------------------

# ENDPOINTS (AbstractEndpoint subclasses)
endpoints.enabled=true # Enable endpoints.
endpoints.sensitive= # Default endpoint sensitive setting.
endpoints.actuator.enabled=true # Enable the endpoint.
endpoints.actuator.path= # Endpoint URL path.
endpoints.actuator.sensitive=false # Enable security on the
endpoint.
endpoints.auditevents.enabled= # Enable the endpoint.
endpoints.auditevents.path= # Endpoint path.
endpoints.auditevents.sensitive=false # Enable security on the
endpoint.
endpoints.autoconfig.enabled= # Enable the endpoint.
endpoints.autoconfig.id= # Endpoint identifier.
endpoints.autoconfig.path= # Endpoint path.
endpoints.autoconfig.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.beans.enabled= # Enable the endpoint.
endpoints.beans.id= # Endpoint identifier.
endpoints.beans.path= # Endpoint path.
endpoints.beans.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.configprops.enabled= # Enable the endpoint.
endpoints.configprops.id= # Endpoint identifier.
endpoints.configprops.keys-to-
sanitize=password,secret,key,token,.*credentials.*,vcap_services
# Keys that should be sanitized. Keys can be simple strings that
the property ends with or regex expressions.
endpoints.configprops.path= # Endpoint path.
endpoints.configprops.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.docs.curies.enabled=false # Enable the curie
generation.
endpoints.docs.enabled=true # Enable actuator docs endpoint.
endpoints.docs.path=/docs #
endpoints.docs.sensitive=false #
endpoints.dump.enabled= # Enable the endpoint.
endpoints.dump.id= # Endpoint identifier.
endpoints.dump.path= # Endpoint path.
endpoints.dump.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.env.enabled= # Enable the endpoint.
endpoints.env.id= # Endpoint identifier.
endpoints.env.keys-to-
sanitize=password,secret,key,token,.*credentials.*,vcap_services
# Keys that should be sanitized. Keys can be simple strings that
the property ends with or regex expressions.
endpoints.env.path= # Endpoint path.
endpoints.env.sensitive= # Mark if the endpoint exposes sensitive
information.
endpoints.flyway.enabled= # Enable the endpoint.
endpoints.flyway.id= # Endpoint identifier.
endpoints.flyway.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.health.enabled= # Enable the endpoint.
endpoints.health.id= # Endpoint identifier.
endpoints.health.mapping.*= # Mapping of health statuses to
HttpStatus codes. By default, registered health statuses map to
sensible defaults (i.e. UP maps to 200).
endpoints.health.path= # Endpoint path.
endpoints.health.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.health.time-to-live=1000 # Time to live for cached
result, in milliseconds.
endpoints.heapdump.enabled= # Enable the endpoint.
endpoints.heapdump.path= # Endpoint path.
endpoints.heapdump.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.hypermedia.enabled=false # Enable hypermedia support
for endpoints.
endpoints.info.enabled= # Enable the endpoint.
endpoints.info.id= # Endpoint identifier.
endpoints.info.path= # Endpoint path.
endpoints.info.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.jolokia.enabled=true # Enable Jolokia endpoint.
endpoints.jolokia.path=/jolokia # Endpoint URL path.
endpoints.jolokia.sensitive=true # Enable security on the
endpoint.
endpoints.liquibase.enabled= # Enable the endpoint.
endpoints.liquibase.id= # Endpoint identifier.
endpoints.liquibase.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.logfile.enabled=true # Enable the endpoint.
endpoints.logfile.external-file= # External Logfile to be
accessed.
endpoints.logfile.path=/logfile # Endpoint URL path.
endpoints.logfile.sensitive=true # Enable security on the
endpoint.
endpoints.loggers.enabled=true # Enable the endpoint.
endpoints.loggers.id= # Endpoint identifier.
endpoints.loggers.path=/logfile # Endpoint path.
endpoints.loggers.sensitive=true # Mark if the endpoint exposes
sensitive information.
endpoints.mappings.enabled= # Enable the endpoint.
endpoints.mappings.id= # Endpoint identifier.
endpoints.mappings.path= # Endpoint path.
endpoints.mappings.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.metrics.enabled= # Enable the endpoint.
endpoints.metrics.filter.enabled=true # Enable the metrics
servlet filter.
endpoints.metrics.filter.gauge-submissions=merged # Http filter
gauge submissions (merged, per-http-method)
endpoints.metrics.filter.counter-submissions=merged # Http filter
counter submissions (merged, per-http-method)
endpoints.metrics.id= # Endpoint identifier.
endpoints.metrics.path= # Endpoint path.
endpoints.metrics.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.shutdown.enabled= # Enable the endpoint.
endpoints.shutdown.id= # Endpoint identifier.
endpoints.shutdown.path= # Endpoint path.
endpoints.shutdown.sensitive= # Mark if the endpoint exposes
sensitive information.
endpoints.trace.enabled= # Enable the endpoint.
endpoints.trace.id= # Endpoint identifier.
endpoints.trace.path= # Endpoint path.
endpoints.trace.sensitive= # Mark if the endpoint exposes
sensitive information.

# ENDPOINTS CORS CONFIGURATION (EndpointCorsProperties)
endpoints.cors.allow-credentials= # Set whether credentials are
supported. When not set, credentials are not supported.
endpoints.cors.allowed-headers= # Comma-separated list of headers
to allow in a request. '*' allows all headers.
endpoints.cors.allowed-methods=GET # Comma-separated list of
methods to allow. '*' allows all methods.
endpoints.cors.allowed-origins= # Comma-separated list of origins
to allow. '*' allows all origins. When not set, CORS support is
disabled.
endpoints.cors.exposed-headers= # Comma-separated list of headers
to include in a response.
endpoints.cors.max-age=1800 # How long, in seconds, the response
from a pre-flight request can be cached by clients.

# JMX ENDPOINT (EndpointMBeanExportProperties)
endpoints.jmx.domain= # JMX domain name. Initialized with the
value of 'spring.jmx.default-domain' if set.
endpoints.jmx.enabled=true # Enable JMX export of all endpoints.
endpoints.jmx.static-names= # Additional static properties to
append to all ObjectNames of MBeans representing Endpoints.
endpoints.jmx.unique-names=false # Ensure that ObjectNames are
modified in case of conflict.
```

```
# JOLOKIA (JolokiaProperties)
jolokia.config.*= # See Jolokia manual

# MANAGEMENT HTTP SERVER (ManagementServerProperties)
management.add-application-context-header=true # Add the "X-
Application-Context" HTTP header in each response.
management.address= # Network address that the management
endpoints should bind to.
management.context-path= # Management endpoint context-path. For
instance `/actuator`
management.cloudfoundry.enabled= # Enable extended Cloud Foundry
actuator endpoints
management.cloudfoundry.skip-ssl-validation= # Skip SSL
verification for Cloud Foundry actuator endpoint security calls
management.port= # Management endpoint HTTP port. Uses the same
port as the application by default. Configure a different port to
use management-specific SSL.
management.security.enabled=true # Enable security.
management.security.roles=ACTUATOR # Comma-separated list of
roles that can access the management endpoint.
management.security.sessions=stateless # Session creating policy
to use (always, never, if_required, stateless).
management.ssl.ciphers= # Supported SSL ciphers. Requires a
custom management.port.
management.ssl.client-auth= # Whether client authentication is
wanted ("want") or needed ("need"). Requires a trust store.
Requires a custom management.port.
management.ssl.enabled= # Enable SSL support. Requires a custom
management.port.
management.ssl.enabled-protocols= # Enabled SSL protocols.
Requires a custom management.port.
management.ssl.key-alias= # Alias that identifies the key in the
key store. Requires a custom management.port.
management.ssl.key-password= # Password used to access the key in
the key store. Requires a custom management.port.
management.ssl.key-store= # Path to the key store that holds the
SSL certificate (typically a jks file). Requires a custom
management.port.
management.ssl.key-store-password= # Password used to access the
key store. Requires a custom management.port.
management.ssl.key-store-provider= # Provider for the key store.
Requires a custom management.port.
```

```
management.ssl.key-store-type= # Type of the key store. Requires
a custom management.port.
management.ssl.protocol=TLS # SSL protocol to use. Requires a
custom management.port.
management.ssl.trust-store= # Trust store that holds SSL
certificates. Requires a custom management.port.
management.ssl.trust-store-password= # Password used to access
the trust store. Requires a custom management.port.
management.ssl.trust-store-provider= # Provider for the trust
store. Requires a custom management.port.
management.ssl.trust-store-type= # Type of the trust store.
Requires a custom management.port.

# HEALTH INDICATORS
management.health.db.enabled=true # Enable database health check.
management.health.cassandra.enabled=true # Enable cassandra
health check.
management.health.couchbase.enabled=true # Enable couchbase
health check.
management.health.defaults.enabled=true # Enable default health
indicators.
management.health.diskspace.enabled=true # Enable disk space
health check.
management.health.diskspace.path= # Path used to compute the
available disk space.
management.health.diskspace.threshold=0 # Minimum disk space that
should be available, in bytes.
management.health.elasticsearch.enabled=true # Enable
elasticsearch health check.
management.health.elasticsearch.indices= # Comma-separated index
names.
management.health.elasticsearch.response-timeout=100 # The time,
in milliseconds, to wait for a response from the cluster.
management.health.jms.enabled=true # Enable JMS health check.
management.health.ldap.enabled=true # Enable LDAP health check.
management.health.mail.enabled=true # Enable Mail health check.
management.health.mongo.enabled=true # Enable MongoDB health
check.
management.health.rabbit.enabled=true # Enable RabbitMQ health
check.
management.health.redis.enabled=true # Enable Redis health check.
management.health.solr.enabled=true # Enable Solr health check.
management.health.status.order=DOWN, OUT_OF_SERVICE, UP, UNKNOWN
# Comma-separated list of health statuses in order of severity.
# INFO CONTRIBUTORS (InfoContributorProperties)
management.info.build.enabled=true # Enable build info.
management.info.defaults.enabled=true # Enable default info
contributors.
management.info.env.enabled=true # Enable environment info.
management.info.git.enabled=true # Enable git info.
management.info.git.mode=simple # Mode to use to expose git
information.

# REMOTE SHELL (ShellProperties)
management.shell.auth.type=simple # Authentication type. Auto-
detected according to the environment.
management.shell.auth.jaas.domain=my-domain # JAAS domain.
management.shell.auth.key.path= # Path to the authentication key.
This should point to a valid ".pem" file.
management.shell.auth.simple.user.name=user # Login user.
management.shell.auth.simple.user.password= # Login password.
management.shell.auth.spring.roles=ACTUATOR # Comma-separated
list of required roles to login to the CRaSH console.
management.shell.command-path-
patterns=classpath*:/commands/**,classpath*:/crash/commands/** #
Patterns to use to look for commands.
management.shell.command-refresh-interval=-1 # Scan for changes
and update the command if necessary (in seconds).
management.shell.config-path-patterns=classpath*:/crash/* #
Patterns to use to look for configurations.
management.shell.disabled-commands=jpa*,jdbc*,jndi* # Comma-
separated list of commands to disable.
management.shell.disabled-plugins= # Comma-separated list of
plugins to disable. Certain plugins are disabled by default based
on the environment.
management.shell.ssh.auth-timeout = # Number of milliseconds
after user will be prompted to login again.
management.shell.ssh.enabled=true # Enable CRaSH SSH support.
management.shell.ssh.idle-timeout = # Number of milliseconds
after which unused connections are closed.
management.shell.ssh.key-path= # Path to the SSH server key.
management.shell.ssh.port=2000 # SSH port.
management.shell.telnet.enabled=false # Enable CRaSH telnet
support. Enabled by default if the TelnetPlugin is  available.
management.shell.telnet.port=5000 # Telnet port.

# TRACING (TraceProperties)
```

```
management.trace.include=request-headers,response-
headers,cookies,errors # Items to be included in the trace.

# METRICS EXPORT (MetricExportProperties)
spring.metrics.export.aggregate.key-pattern= # Pattern that tells
the aggregator what to do with the keys from the source
repository.
spring.metrics.export.aggregate.prefix= # Prefix for global
repository if active.
spring.metrics.export.delay-millis=5000 # Delay in milliseconds
between export ticks. Metrics are exported to external sources on
a schedule with this delay.
spring.metrics.export.enabled=true # Flag to enable metric export
(assuming a MetricWriter is available).
spring.metrics.export.excludes= # List of patterns for metric
names to exclude. Applied after the includes.
spring.metrics.export.includes= # List of patterns for metric
names to include.
spring.metrics.export.redis.key=keys.spring.metrics # Key for
redis repository export (if active).
spring.metrics.export.redis.prefix=spring.metrics # Prefix for
redis repository if active.
spring.metrics.export.send-latest= # Flag to switch off any
available optimizations based on not exporting unchanged metric
values.
spring.metrics.export.statsd.host= # Host of a statsd server to
receive exported metrics.
spring.metrics.export.statsd.port=8125 # Port of a statsd server
to receive exported metrics.
spring.metrics.export.statsd.prefix= # Prefix for statsd exported
metrics.
spring.metrics.export.triggers.*= # Specific trigger properties
per MetricWriter bean name.


# ----------------------------------------
# DEVTOOLS PROPERTIES
# ----------------------------------------

# DEVTOOLS (DevToolsProperties)
spring.devtools.livereload.enabled=true # Enable a livereload.com
compatible server.
spring.devtools.livereload.port=35729 # Server port.
spring.devtools.restart.additional-exclude= # Additional patterns
that should be excluded from triggering a full restart.
spring.devtools.restart.additional-paths= # Additional paths to
watch for changes.
spring.devtools.restart.enabled=true # Enable automatic restart.
spring.devtools.restart.exclude=META-INF/maven/**,META-
INF/resources/**,resources/**,static/**,public/**,templates/**,**
/*Test.class,**/*Tests.class,git.properties # Patterns that
should be excluded from triggering a full restart.
spring.devtools.restart.poll-interval=1000 # Amount of time (in
milliseconds) to wait between polling for classpath changes.
spring.devtools.restart.quiet-period=400 # Amount of quiet time
(in milliseconds) required without any classpath changes before a
restart is triggered.
spring.devtools.restart.trigger-file= # Name of a specific file
that when changed will trigger the restart check. If not
specified any classpath file change will trigger the restart.

# REMOTE DEVTOOLS (RemoteDevToolsProperties)
spring.devtools.remote.context-path=/.~~spring-boot!~ # Context
path used to handle the remote connection.
spring.devtools.remote.debug.enabled=true # Enable remote debug
support.
spring.devtools.remote.debug.local-port=8000 # Local remote debug
server port.
spring.devtools.remote.proxy.host= # The host of the proxy to use
to connect to the remote application.
spring.devtools.remote.proxy.port= # The port of the proxy to use
to connect to the remote application.
spring.devtools.remote.restart.enabled=true # Enable remote
restart.
spring.devtools.remote.secret= # A shared secret required to
establish a connection (required to enable remote support).
spring.devtools.remote.secret-header-name=X-AUTH-TOKEN # HTTP
header used to transfer the shared secret.


# ----------------------------------------
# TESTING PROPERTIES
# ----------------------------------------

spring.test.database.replace=any # Type of existing DataSource to
replace.
spring.test.mockmvc.print=default # MVC Print option.
```

4.4.5. Starter pom



Spring Boot 为我们提供了简化企业级开发绝大多数场景的 start
景所需要的 starter pom，相关的技术配置将会消除，就可以得到 Sp
动配置的 Bean。

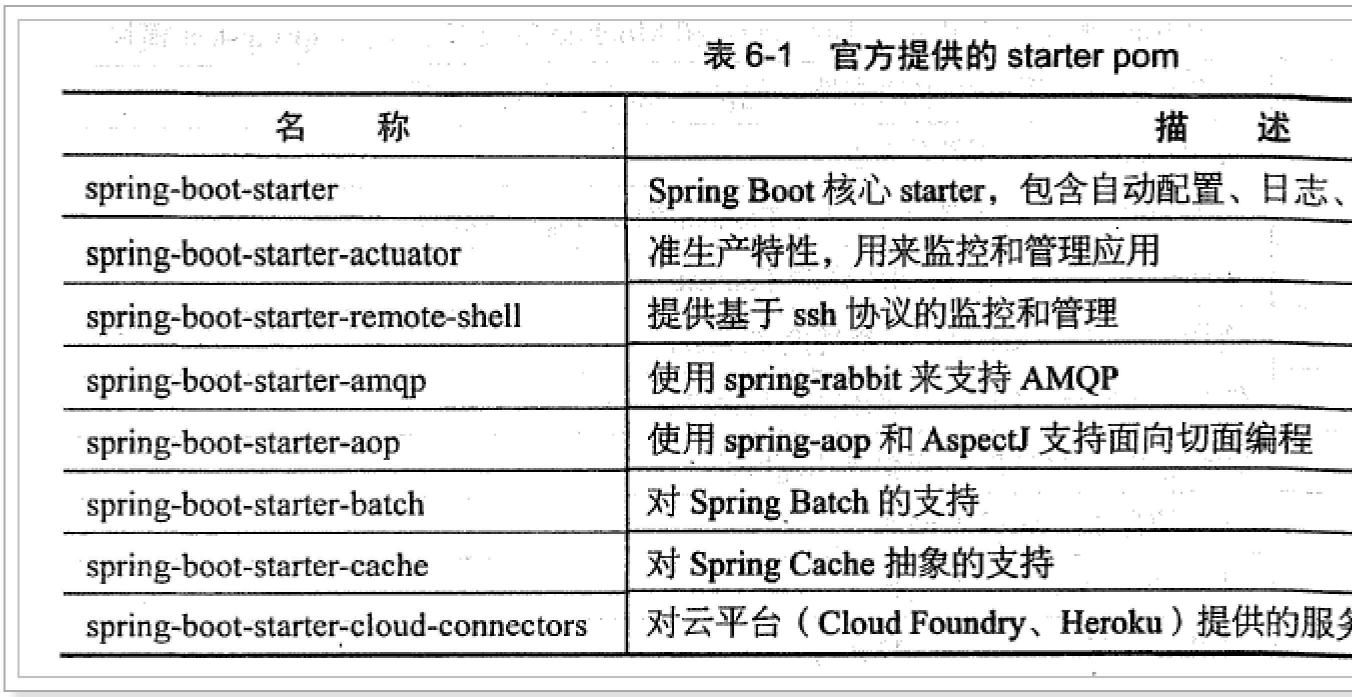
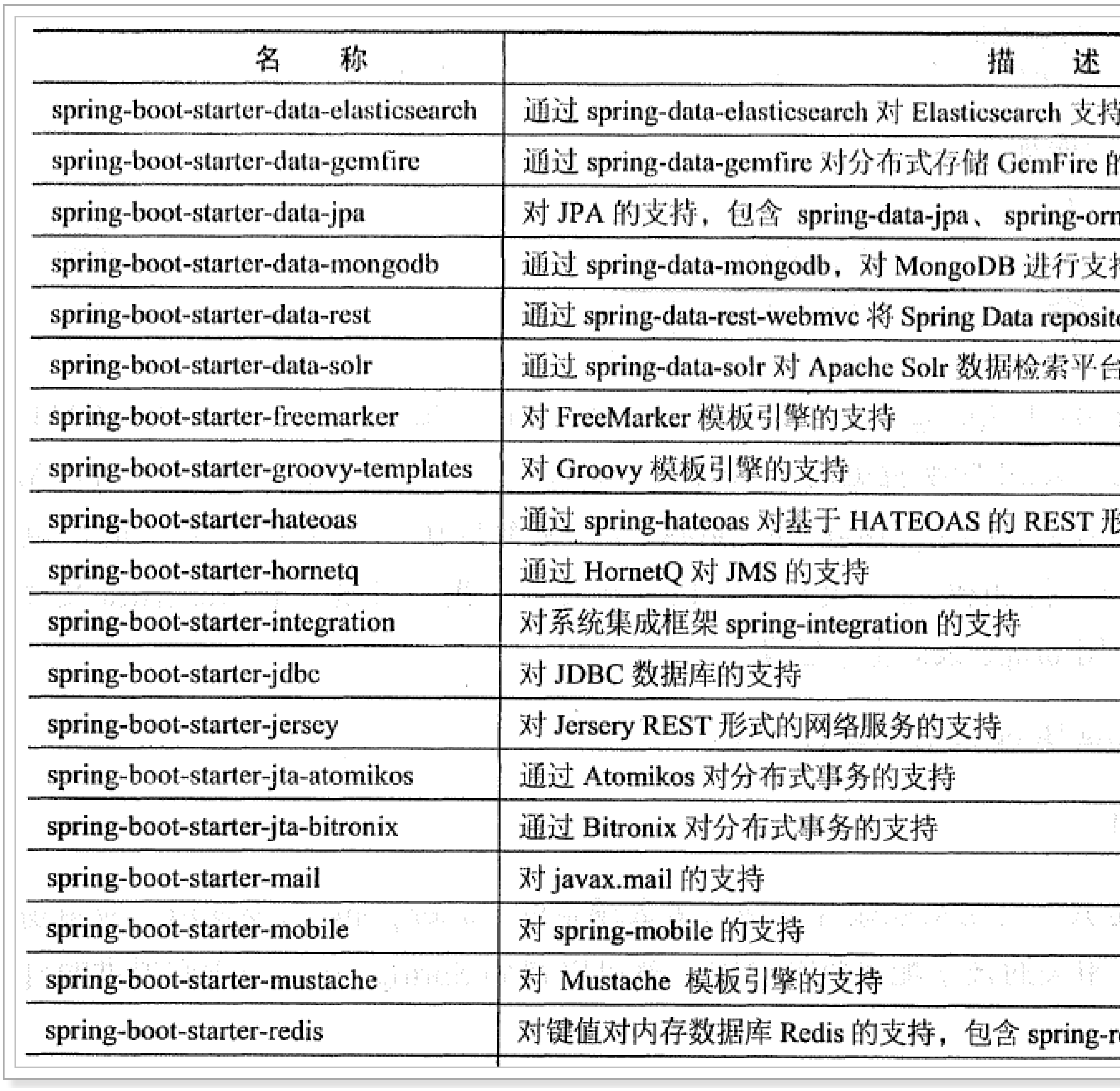
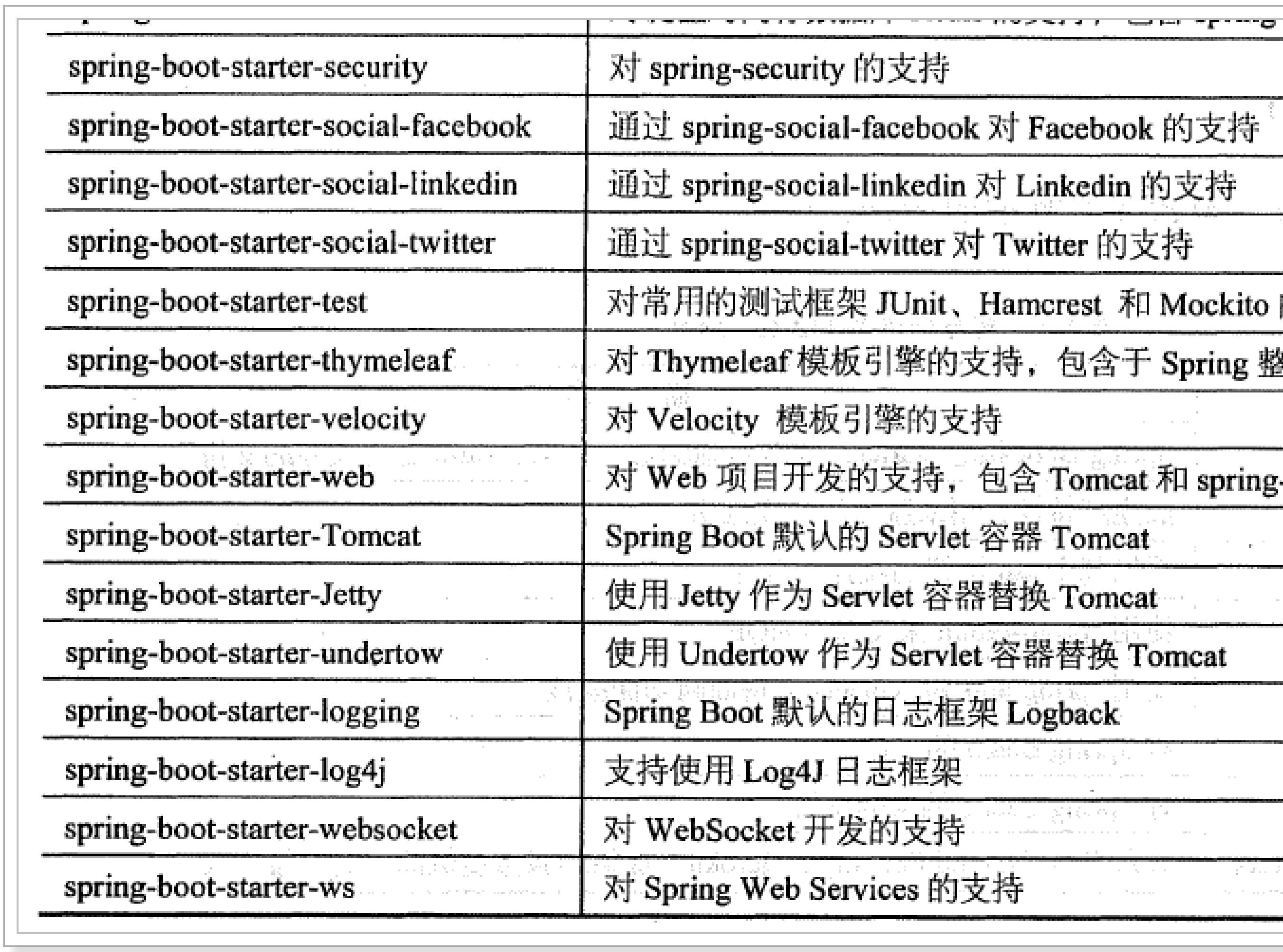


表 6-1 官方提供的 starter pom

名　称	描　述
spring-boot-starter	Spring Boot 核心 starter，包含自动配置、日志、
spring-boot-starter-actuator	准生产特性，用来监控和管理应用
spring-boot-starter-remote-shell	提供基于 ssh 协议的监控和管理
spring-boot-starter-amqp	使用 spring-rabbit 来支持 AMQP
spring-boot-starter-aop	使用 spring-aop 和 AspectJ 支持面向切面编程
spring-boot-starter-batch	对 Spring Batch 的支持
spring-boot-starter-cache	对 Spring Cache 抽象的支持
spring-boot-starter-cloud-connectors	对云平台（Cloud Foundry、Heroku）提供的服务

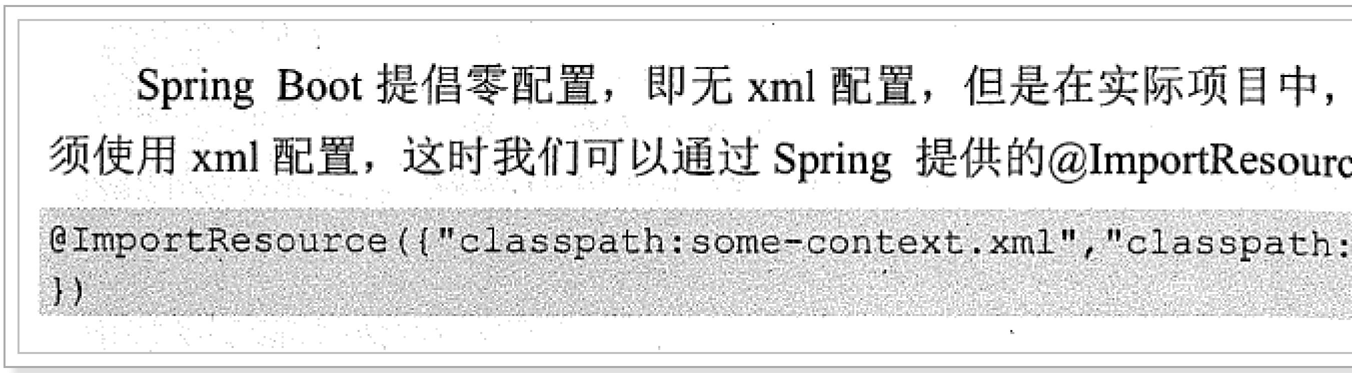


名　称	描　述
spring-boot-starter-data-elasticsearch	通过 spring-data-elasticsearch 对 Elasticsearch 支持
spring-boot-starter-data-gemfire	通过 spring-data-gemfire 对分布式存储 GemFire 的
spring-boot-starter-data-jpa	对 JPA 的支持，包含 spring-data-jpa、 spring-orm
spring-boot-starter-data-mongodb	通过 spring-data-mongodb，对 MongoDB 进行支持
spring-boot-starter-data-rest	通过 spring-data-rest-webmvc 将 Spring Data reposito
spring-boot-starter-data-solr	通过 spring-data-solr 对 Apache Solr 数据检索平台
spring-boot-starter-freemarker	对 FreeMarker 模板引擎的支持
spring-boot-starter-groovy-templates	对 Groovy 模板引擎的支持
spring-boot-starter-hateoas	通过 spring-hateoas 对基于 HATEOAS 的 REST 形
spring-boot-starter-hornetq	通过 HornetQ 对 JMS 的支持
spring-boot-starter-integration	对系统集成框架 spring-integration 的支持
spring-boot-starter-jdbc	对 JDBC 数据库的支持
spring-boot-starter-jersey	对 Jersery REST 形式的网络服务的支持
spring-boot-starter-jta-atomikos	通过 Atomikos 对分布式事务的支持
spring-boot-starter-jta-bitronix	通过 Bitronix 对分布式事务的支持
spring-boot-starter-mail	对 javax.mail 的支持
spring-boot-starter-mobile	对 spring-mobile 的支持
spring-boot-starter-mustache	对 Mustache 模板引擎的支持
spring-boot-starter-redis	对键值对内存数据库 Redis 的支持，包含 spring-re



[illegible]	[illegible]
spring-boot-starter-security	对 spring-security 的支持
spring-boot-starter-social-facebook	通过 spring-social-facebook 对 Facebook 的支持
spring-boot-starter-social-linkedin	通过 spring-social-linkedin 对 Linkedin 的支持
spring-boot-starter-social-twitter	通过 spring-social-twitter 对 Twitter 的支持
spring-boot-starter-test	对常用的测试框架 JUnit、Hamcrest 和 Mockito
spring-boot-starter-thymeleaf	对 Thymeleaf 模板引擎的支持，包含于 Spring
spring-boot-starter-velocity	对 Velocity 模板引擎的支持
spring-boot-starter-web	对 Web 项目开发的支持，包含 Tomcat 和 spring-
spring-boot-starter-Tomcat	Spring Boot 默认的 Servlet 容器 Tomcat
spring-boot-starter-Jetty	使用 Jetty 作为 Servlet 容器替换 Tomcat
spring-boot-starter-undertow	使用 Undertow 作为 Servlet 容器替换 Tomcat
spring-boot-starter-logging	Spring Boot 默认的日志框架 Logback
spring-boot-starter-log4j	支持使用 Log4J 日志框架
spring-boot-starter-websocket	对 WebSocket 开发的支持
spring-boot-starter-ws	对 Spring Web Services 的支持

4.4.6. Xml 配置文件



Spring Boot 提倡零配置，即无 xml 配置，但是在实际项目中，
须使用 xml 配置，这时我们可以通过 Spring 提供的@ImportResourc

```
@ImportResource({"classpath:some-context.xml","classpath:
})
```

4.4.7. 日志

Spring Boot 对各种日志框架都做了支持，我们可以通过配置来修改默认的日志的配置：

```
#设置日志级别
logging.level.org.springframework=DEBUG
```

格式：

```
logging.level.*= # Log levels severity mapping. For instance
`logging.level.org.springframework=DEBUG`
```

4.5. Spring Boot 的自动配置的原理

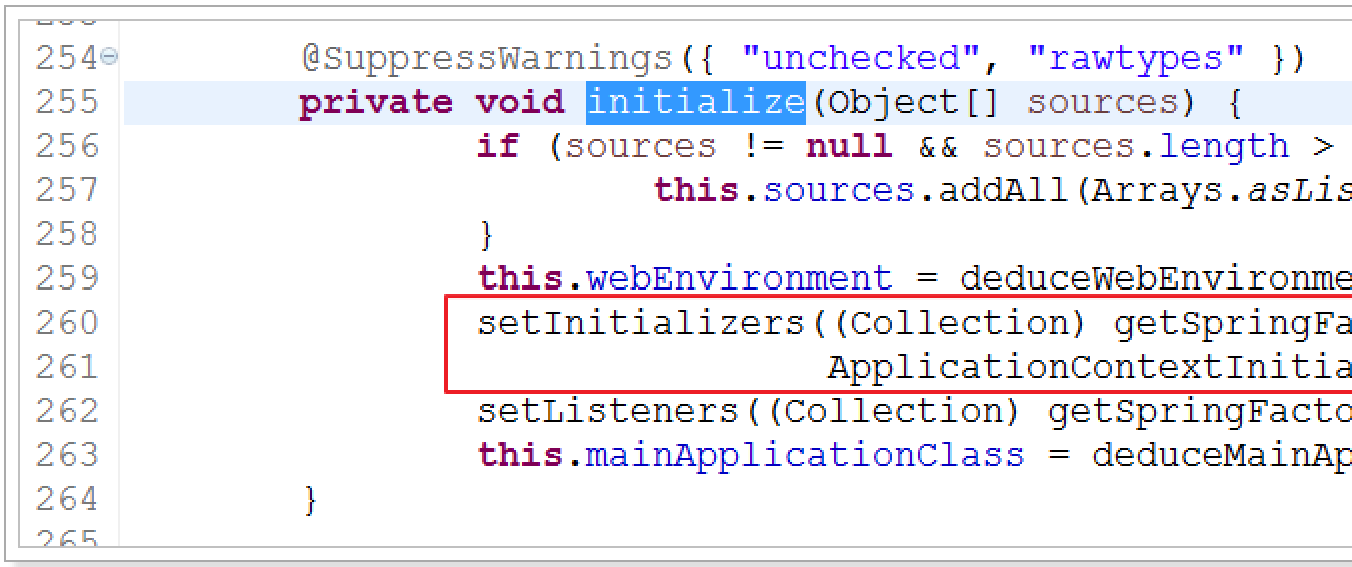
Spring Boot 在进行 SpringApplication 对象实例化时会加载 META-INF/spring.factories 文件，将该配置文件中的配置载入到 Spring 容器。

4.5.1. Maven 下载源码

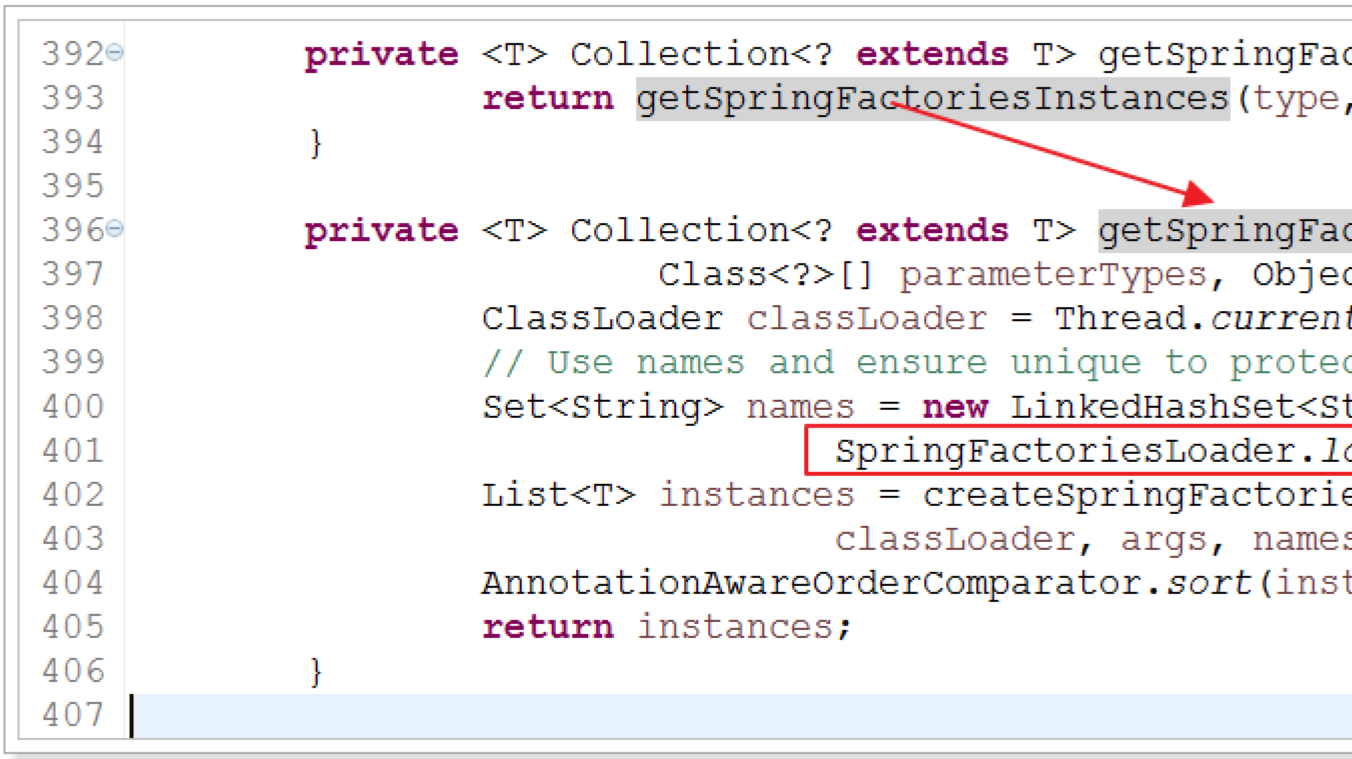
通过 dependency:sources 该命令可以下载该项目中所有的依赖的包的源码。

4.5.2. 源码分析

org.springframework.boot.SpringApplication：

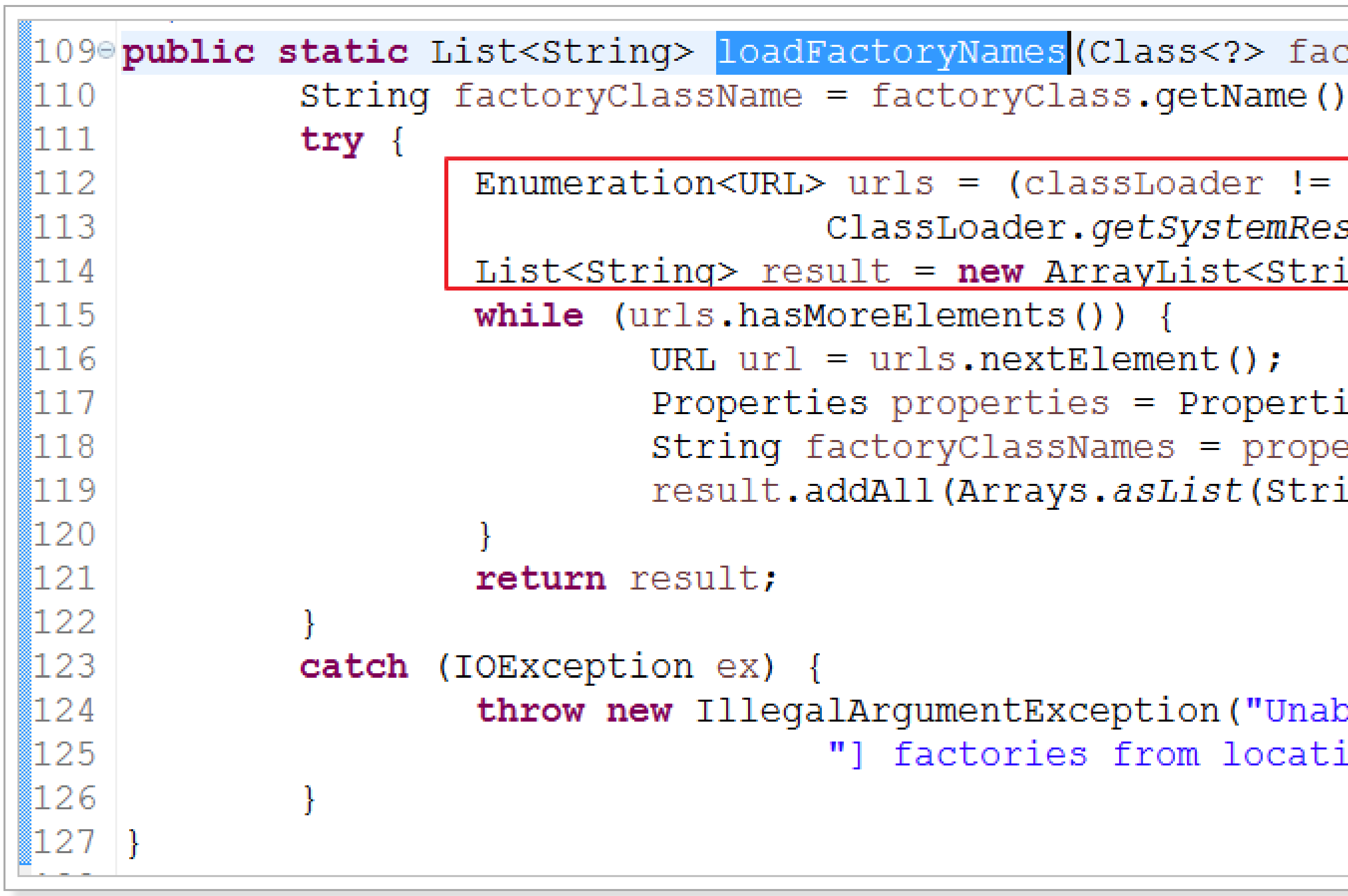


```
    @SuppressWarnings({ "unchecked", "rawtypes" })
    private void initialize(Object[] sources) {
        if (sources != null && sources.length >
            this.sources.addAll(Arrays.asLis
        }
        this.webEnvironment = deduceWebEnvironme
        setInitializers((Collection) getSpringFa
                ApplicationContextInitia
        setListeners((Collection) getSpringFacto
        this.mainApplicationClass = deduceMainAp
    }
```

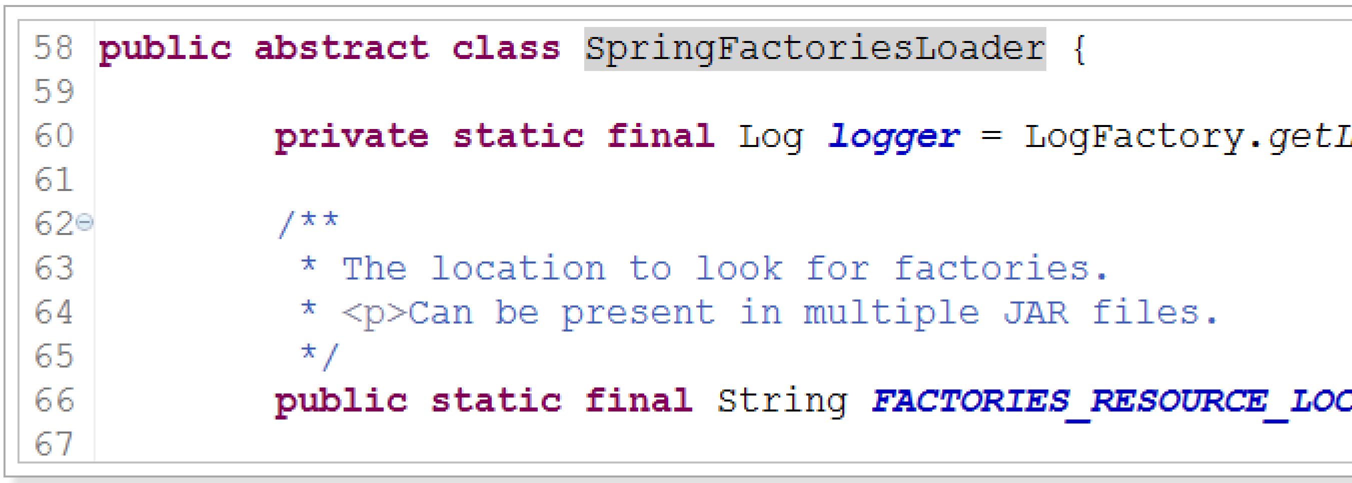


```
	private <T> Collection<? extends T> getSpringFac
		return getSpringFactoriesInstances(type,
	}

	private <T> Collection<? extends T> getSpringFac
			Class<?>[] parameterTypes, Objec
		ClassLoader classLoader = Thread.current
		// Use names and ensure unique to protec
		Set<String> names = new LinkedHashSet<St
				SpringFactoriesLoader.lo
		List<T> instances = createSpringFactorie
				classLoader, args, names
		AnnotationAwareOrderComparator.sort(inst
		return instances;
	}

```

org.springframework.core.io.support.SpringFactoriesLoader:



```
public static List<String> loadFactoryNames(Class<?> fac
	String factoryClassName = factoryClass.getName()
	try {
		Enumeration<URL> urls = (classLoader !=
				ClassLoader.getSystemRes
		List<String> result = new ArrayList<Stri
		while (urls.hasMoreElements()) {
			URL url = urls.nextElement();
			Properties properties = Properti
			String factoryClassNames = prope
			result.addAll(Arrays.asList(Stri
		}
		return result;
	}
	catch (IOException ex) {
		throw new IllegalArgumentException("Unab
				"] factories from locati
	}
}
```



```java
public abstract class SpringFactoriesLoader {

        private static final Log logger = LogFactory.getL

        /**
         * The location to look for factories.
         * <p>Can be present in multiple JAR files.
         */
        public static final String FACTORIES_RESOURCE_LOC

```

由此可见，读取该配置文件来加载内容。

4.5.3. Spring.factories 文件

```
# Initializers
org.springframework.context.ApplicationContextInitializer
=\
org.springframework.boot.autoconfigure.SharedMetadataRead
erFactoryContextInitializer,\
org.springframework.boot.autoconfigure.logging.AutoConfig
urationReportLoggingInitializer

# Application Listeners
org.springframework.context.ApplicationListener=\
org.springframework.boot.autoconfigure.BackgroundPreiniti
alizer

# Auto Configuration Import Listeners
org.springframework.boot.autoconfigure.AutoConfigurationI
mportListener=\
org.springframework.boot.autoconfigure.condition.Conditio
nEvaluationReportAutoConfigurationImportListener

# Auto Configuration Import Filters
org.springframework.boot.autoconfigure.AutoConfigurationI
mportFilter=\
org.springframework.boot.autoconfigure.condition.OnClassC
ondition

# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfigur
```

```
ation=\
org.springframework.boot.autoconfigure.admin.SpringApplic
ationAdminJmxAutoConfiguration,\
org.springframework.boot.autoconfigure.aop.AopAutoConfigu
ration,\
org.springframework.boot.autoconfigure.amqp.RabbitAutoCon
figuration,\
org.springframework.boot.autoconfigure.batch.BatchAutoCon
figuration,\
org.springframework.boot.autoconfigure.cache.CacheAutoCon
figuration,\
org.springframework.boot.autoconfigure.cassandra.Cassandr
aAutoConfiguration,\
org.springframework.boot.autoconfigure.cloud.CloudAutoCon
figuration,\
org.springframework.boot.autoconfigure.context.Configurat
ionPropertiesAutoConfiguration,\
org.springframework.boot.autoconfigure.context.MessageSou
rceAutoConfiguration,\
org.springframework.boot.autoconfigure.context.PropertyPl
aceholderAutoConfiguration,\
org.springframework.boot.autoconfigure.couchbase.Couchbas
eAutoConfiguration,\
org.springframework.boot.autoconfigure.dao.PersistenceExc
eptionTranslationAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.Cas
sandraDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.cassandra.Cas
sandraRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.Cou
chbaseDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.couchbase.Cou
chbaseRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearch
.ElasticsearchAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearch
.ElasticsearchDataAutoConfiguration,\
org.springframework.boot.autoconfigure.data.elasticsearch
.ElasticsearchRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.jpa.JpaReposi
toriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.ldap.LdapData
AutoConfiguration,\
org.springframework.boot.autoconfigure.data.ldap.LdapRepo
```

```
sitoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.mongo.MongoDa
taAutoConfiguration,\
org.springframework.boot.autoconfigure.data.mongo.MongoRe
positoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.neo4j.Neo4jDa
taAutoConfiguration,\
org.springframework.boot.autoconfigure.data.neo4j.Neo4jRe
positoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.solr.SolrRepo
sitoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.redis.RedisAu
toConfiguration,\
org.springframework.boot.autoconfigure.data.redis.RedisRe
positoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.rest.Reposito
ryRestMvcAutoConfiguration,\
org.springframework.boot.autoconfigure.data.web.SpringDat
aWebAutoConfiguration,\
org.springframework.boot.autoconfigure.elasticsearch.jest
.JestAutoConfiguration,\
org.springframework.boot.autoconfigure.freemarker.FreeMar
kerAutoConfiguration,\
org.springframework.boot.autoconfigure.gson.GsonAutoConfi
guration,\
org.springframework.boot.autoconfigure.h2.H2ConsoleAutoCo
nfiguration,\
org.springframework.boot.autoconfigure.hateoas.Hypermedia
AutoConfiguration,\
org.springframework.boot.autoconfigure.hazelcast.Hazelcas
tAutoConfiguration,\
org.springframework.boot.autoconfigure.hazelcast.Hazelcas
tJpaDependencyAutoConfiguration,\
org.springframework.boot.autoconfigure.info.ProjectInfoAu
toConfiguration,\
org.springframework.boot.autoconfigure.integration.Integr
ationAutoConfiguration,\
org.springframework.boot.autoconfigure.jackson.JacksonAut
oConfiguration,\
org.springframework.boot.autoconfigure.jdbc.DataSourceAut
oConfiguration,\
org.springframework.boot.autoconfigure.jdbc.JdbcTemplateA
utoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.JndiDataSourc
```

```
eAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.XADataSourceA
utoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.DataSourceTra
nsactionManagerAutoConfiguration,\
org.springframework.boot.autoconfigure.jms.JmsAutoConfigu
ration,\
org.springframework.boot.autoconfigure.jmx.JmxAutoConfigu
ration,\
org.springframework.boot.autoconfigure.jms.JndiConnection
FactoryAutoConfiguration,\
org.springframework.boot.autoconfigure.jms.activemq.Activ
eMQAutoConfiguration,\
org.springframework.boot.autoconfigure.jms.artemis.Artemi
sAutoConfiguration,\
org.springframework.boot.autoconfigure.flyway.FlywayAutoC
onfiguration,\
org.springframework.boot.autoconfigure.groovy.template.Gr
oovyTemplateAutoConfiguration,\
org.springframework.boot.autoconfigure.jersey.JerseyAutoC
onfiguration,\
org.springframework.boot.autoconfigure.jooq.JooqAutoConfi
guration,\
org.springframework.boot.autoconfigure.kafka.KafkaAutoCon
figuration,\
org.springframework.boot.autoconfigure.ldap.embedded.Embe
ddedLdapAutoConfiguration,\
org.springframework.boot.autoconfigure.ldap.LdapAutoConfi
guration,\
org.springframework.boot.autoconfigure.liquibase.Liquibas
eAutoConfiguration,\
org.springframework.boot.autoconfigure.mail.MailSenderAut
oConfiguration,\
org.springframework.boot.autoconfigure.mail.MailSenderVal
idatorAutoConfiguration,\
org.springframework.boot.autoconfigure.mobile.DeviceResol
verAutoConfiguration,\
org.springframework.boot.autoconfigure.mobile.DeviceDeleg
atingViewResolverAutoConfiguration,\
org.springframework.boot.autoconfigure.mobile.SitePrefere
nceAutoConfiguration,\
org.springframework.boot.autoconfigure.mongo.embedded.Emb
eddedMongoAutoConfiguration,\
org.springframework.boot.autoconfigure.mongo.MongoAutoCon
```

```
figuration,\
org.springframework.boot.autoconfigure.mustache.MustacheA
utoConfiguration,\
org.springframework.boot.autoconfigure.orm.jpa.HibernateJ
paAutoConfiguration,\
org.springframework.boot.autoconfigure.reactor.ReactorAut
oConfiguration,\
org.springframework.boot.autoconfigure.security.SecurityA
utoConfiguration,\
org.springframework.boot.autoconfigure.security.SecurityF
ilterAutoConfiguration,\
org.springframework.boot.autoconfigure.security.FallbackW
ebSecurityAutoConfiguration,\
org.springframework.boot.autoconfigure.security.oauth2.OA
uth2AutoConfiguration,\
org.springframework.boot.autoconfigure.sendgrid.SendGridA
utoConfiguration,\
org.springframework.boot.autoconfigure.session.SessionAut
oConfiguration,\
org.springframework.boot.autoconfigure.social.SocialWebAu
toConfiguration,\
org.springframework.boot.autoconfigure.social.FacebookAut
oConfiguration,\
org.springframework.boot.autoconfigure.social.LinkedInAut
oConfiguration,\
org.springframework.boot.autoconfigure.social.TwitterAuto
Configuration,\
org.springframework.boot.autoconfigure.solr.SolrAutoConfi
guration,\
org.springframework.boot.autoconfigure.thymeleaf.Thymelea
fAutoConfiguration,\
org.springframework.boot.autoconfigure.transaction.Transa
ctionAutoConfiguration,\
org.springframework.boot.autoconfigure.transaction.jta.Jt
aAutoConfiguration,\
org.springframework.boot.autoconfigure.validation.Validat
ionAutoConfiguration,\
org.springframework.boot.autoconfigure.web.DispatcherServ
letAutoConfiguration,\
org.springframework.boot.autoconfigure.web.EmbeddedServle
tContainerAutoConfiguration,\
org.springframework.boot.autoconfigure.web.ErrorMvcAutoCo
nfiguration,\
org.springframework.boot.autoconfigure.web.HttpEncodingAu
```

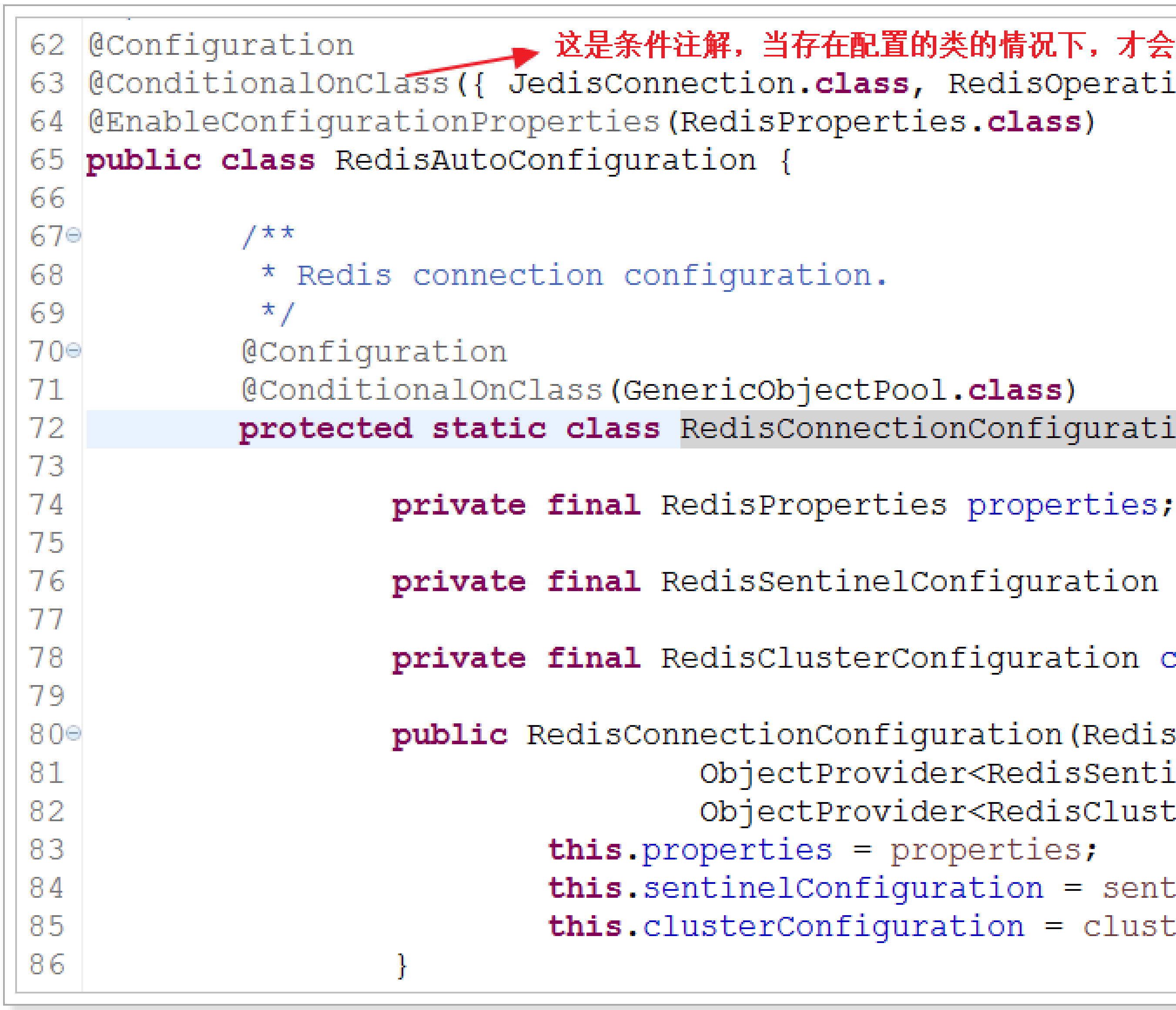
```
toConfiguration,\
org.springframework.boot.autoconfigure.web.HttpMessageCon
vertersAutoConfiguration,\
org.springframework.boot.autoconfigure.web.MultipartAutoC
onfiguration,\
org.springframework.boot.autoconfigure.web.ServerProperti
esAutoConfiguration,\
org.springframework.boot.autoconfigure.web.WebClientAutoC
onfiguration,\
org.springframework.boot.autoconfigure.web.WebMvcAutoConf
iguration,\
org.springframework.boot.autoconfigure.websocket.WebSocke
tAutoConfiguration,\
org.springframework.boot.autoconfigure.websocket.WebSocke
tMessagingAutoConfiguration,\
org.springframework.boot.autoconfigure.webservices.WebSer
vicesAutoConfiguration

# Failure analyzers
org.springframework.boot.diagnostics.FailureAnalyzer=\
org.springframework.boot.autoconfigure.diagnostics.analyz
er.NoSuchBeanDefinitionFailureAnalyzer,\
org.springframework.boot.autoconfigure.jdbc.DataSourceBea
nCreationFailureAnalyzer,\
org.springframework.boot.autoconfigure.jdbc.HikariDriverC
onfigurationFailureAnalyzer

# Template availability providers
org.springframework.boot.autoconfigure.template.TemplateA
vailabilityProvider=\
org.springframework.boot.autoconfigure.freemarker.FreeMar
kerTemplateAvailabilityProvider,\
org.springframework.boot.autoconfigure.mustache.MustacheT
emplateAvailabilityProvider,\
org.springframework.boot.autoconfigure.groovy.template.Gr
oovyTemplateAvailabilityProvider,\
org.springframework.boot.autoconfigure.thymeleaf.Thymelea
fTemplateAvailabilityProvider,\
org.springframework.boot.autoconfigure.web.JspTemplateAva
ilabilityProvider
```

4.5.4. 举例：Redis 的自动配置

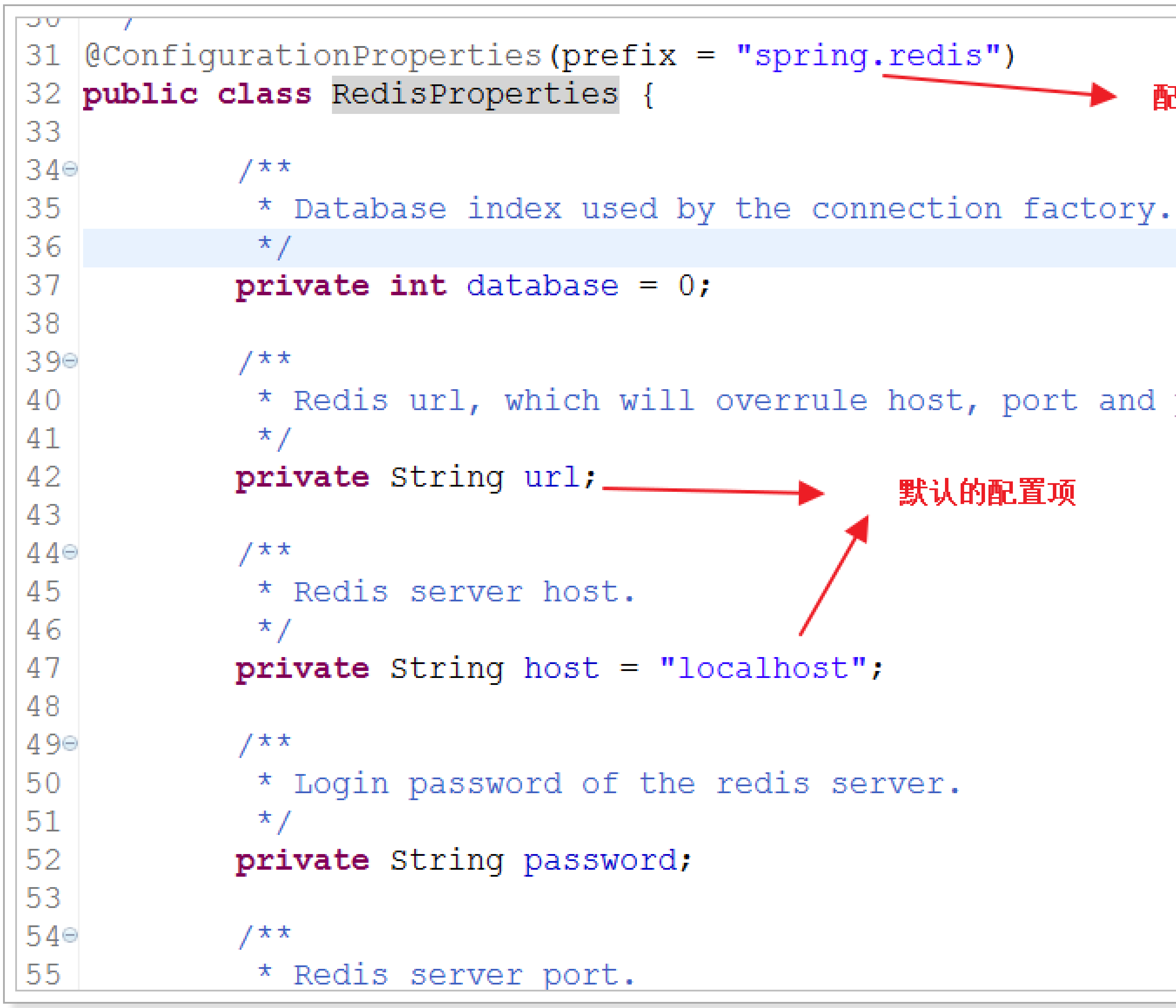
从上述的配置中可以看出，

org.springframework.boot.autoconfigure.data.redis.RedisAutoConfiguration 是 Redis 的自动配置。

内容：



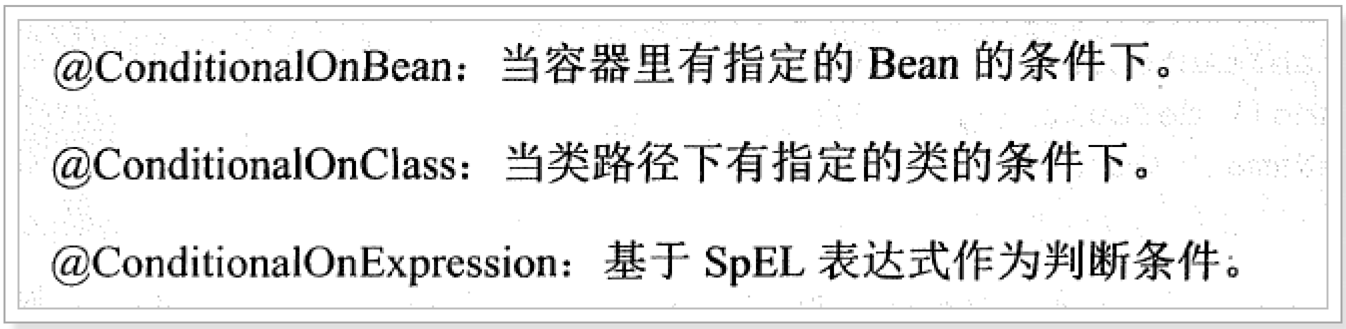
```
@Configuration
@ConditionalOnClass({ JedisConnection.class, RedisOperati
@EnableConfigurationProperties(RedisProperties.class)
public class RedisAutoConfiguration {

        /**
         * Redis connection configuration.
         */
        @Configuration
        @ConditionalOnClass(GenericObjectPool.class)
        protected static class RedisConnectionConfigurati

                private final RedisProperties properties;

                private final RedisSentinelConfiguration

                private final RedisClusterConfiguration c

                public RedisConnectionConfiguration(Redis
                                ObjectProvider<RedisSenti
                                ObjectProvider<RedisClust
                        this.properties = properties;
                        this.sentinelConfiguration = sent
                        this.clusterConfiguration = clust
                }
```



```java
 */
@ConfigurationProperties(prefix = "spring.redis")
public class RedisProperties {

	/**
	 * Database index used by the connection factory.
	 */
	private int database = 0;

	/**
	 * Redis url, which will overrule host, port and
	 */
	private String url;

	/**
	 * Redis server host.
	 */
	private String host = "localhost";

	/**
	 * Login password of the redis server.
	 */
	private String password;

	/**
	 * Redis server port.
```

】

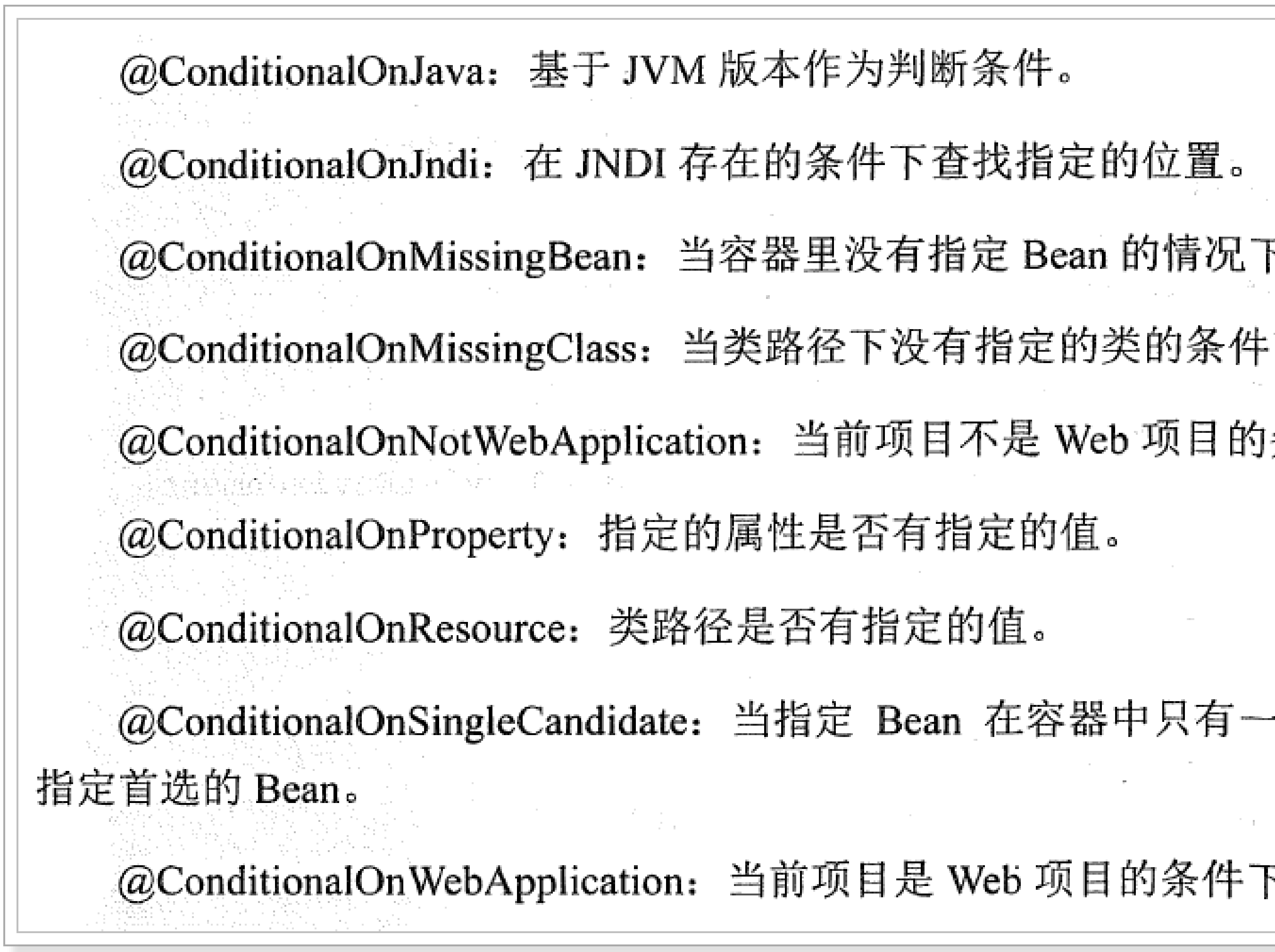
4.5.5. 条件注解



@ConditionalOnBean：当容器里有指定的 Bean 的条件下。

@ConditionalOnClass：当类路径下有指定的类的条件下。

@ConditionalOnExpression：基于 SpEL 表达式作为判断条件。



@ConditionalOnJava：基于 JVM 版本作为判断条件。

@ConditionalOnJndi：在 JNDI 存在的条件下查找指定的位置。

@ConditionalOnMissingBean：当容器里没有指定 Bean 的情况下

@ConditionalOnMissingClass：当类路径下没有指定的类的条件

@ConditionalOnNotWebApplication：当前项目不是 Web 项目的

@ConditionalOnProperty：指定的属性是否有指定的值。

@ConditionalOnResource：类路径是否有指定的值。

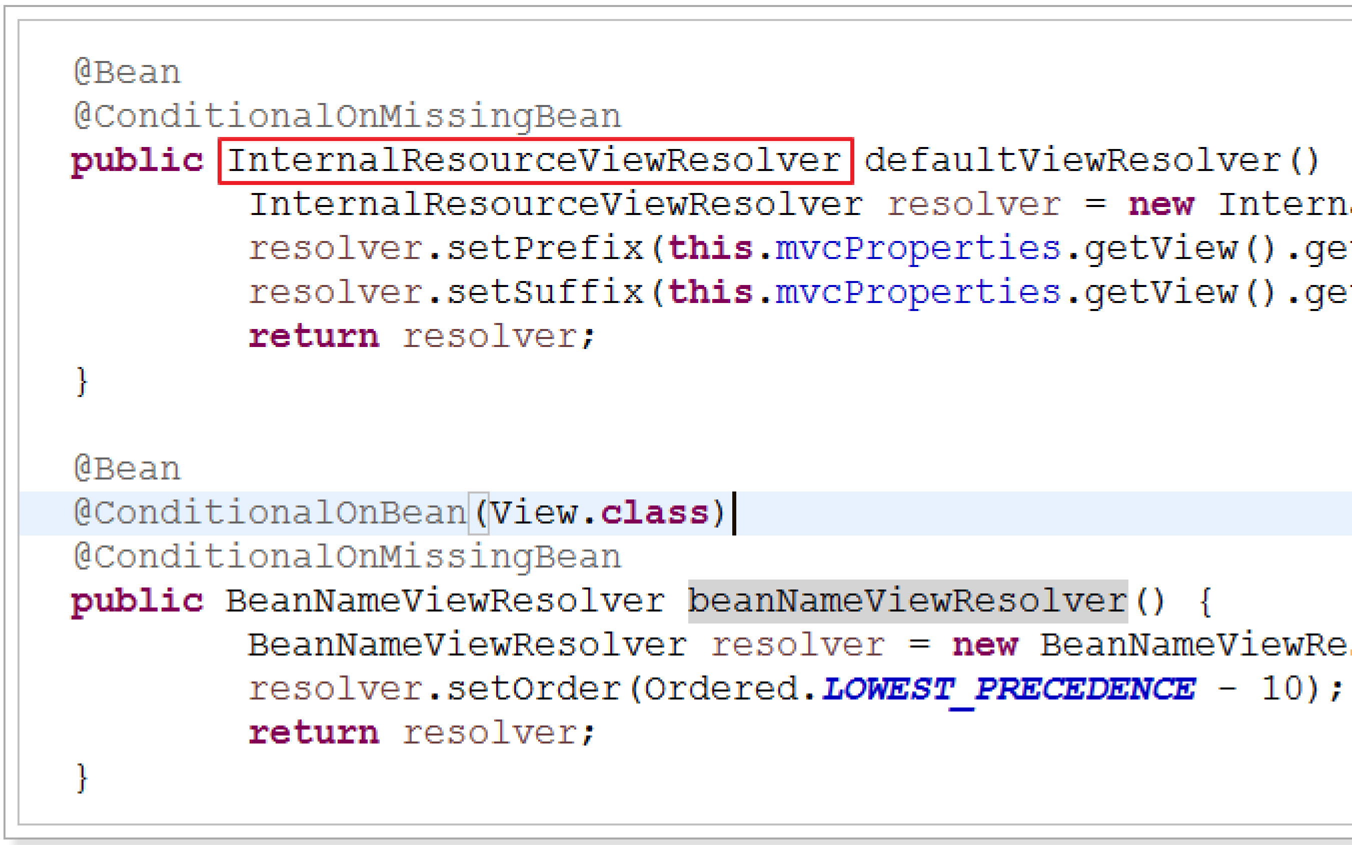
@ConditionalOnSingleCandidate：当指定 Bean 在容器中只有一
指定首选的 Bean。

@ConditionalOnWebApplication：当前项目是 Web 项目的条件下

5. Spring Boot 的 web 开发

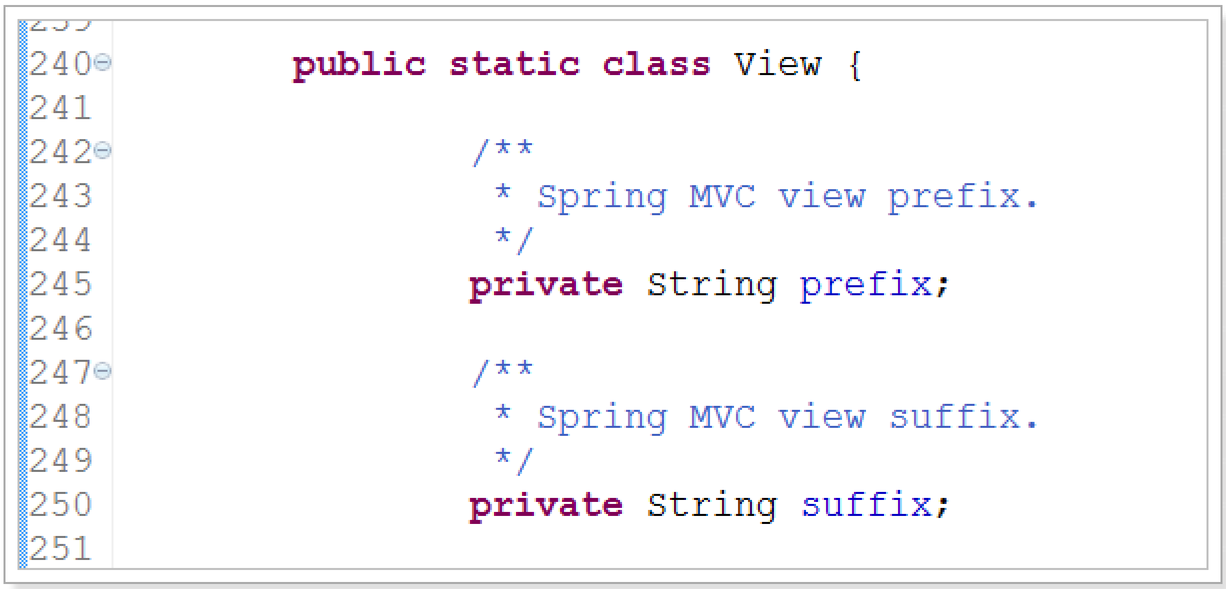
Web 开发的自动配置类：org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration

5.1. 自动配置的 ViewResolver



```
@Bean
@ConditionalOnMissingBean
public InternalResourceViewResolver defaultViewResolver()
        InternalResourceViewResolver resolver = new Intern
        resolver.setPrefix(this.mvcProperties.getView().ge
        resolver.setSuffix(this.mvcProperties.getView().ge
        return resolver;
}

@Bean
@ConditionalOnBean(View.class)
@ConditionalOnMissingBean
public BeanNameViewResolver beanNameViewResolver() {
        BeanNameViewResolver resolver = new BeanNameViewRe
        resolver.setOrder(Ordered.LOWEST_PRECEDENCE - 10);
        return resolver;
}
```

视图的配置 mvcProperties 对象中：
org.springframework.boot.autoconfigure.web.WebMvcProperties.View



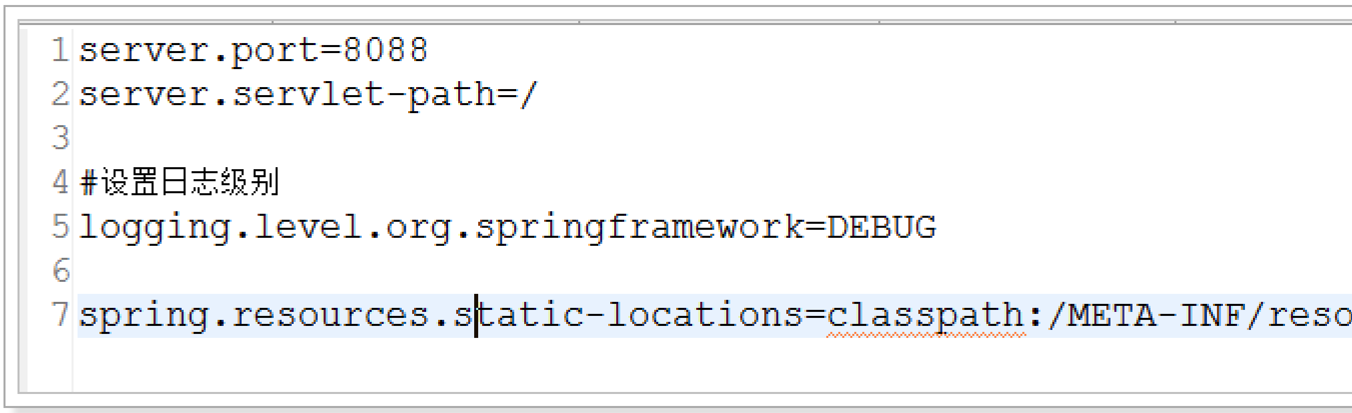
```
        public static class View {

                /**
                 * Spring MVC view prefix.
                 */
                private String prefix;

                /**
                 * Spring MVC view suffix.
                 */
                private String suffix;
```

5.2. 自动配置静态资源

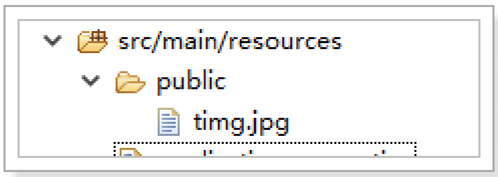
5.2.1. 进入规则为 /

如果进入 SpringMVC 的规则为/时，Spring Boot 的默认静态资源的路径为：
spring.resources.static-locations=classpath:/META-INF/resources/,classpath:/resources/,classpath:/static/,classpath:/public/

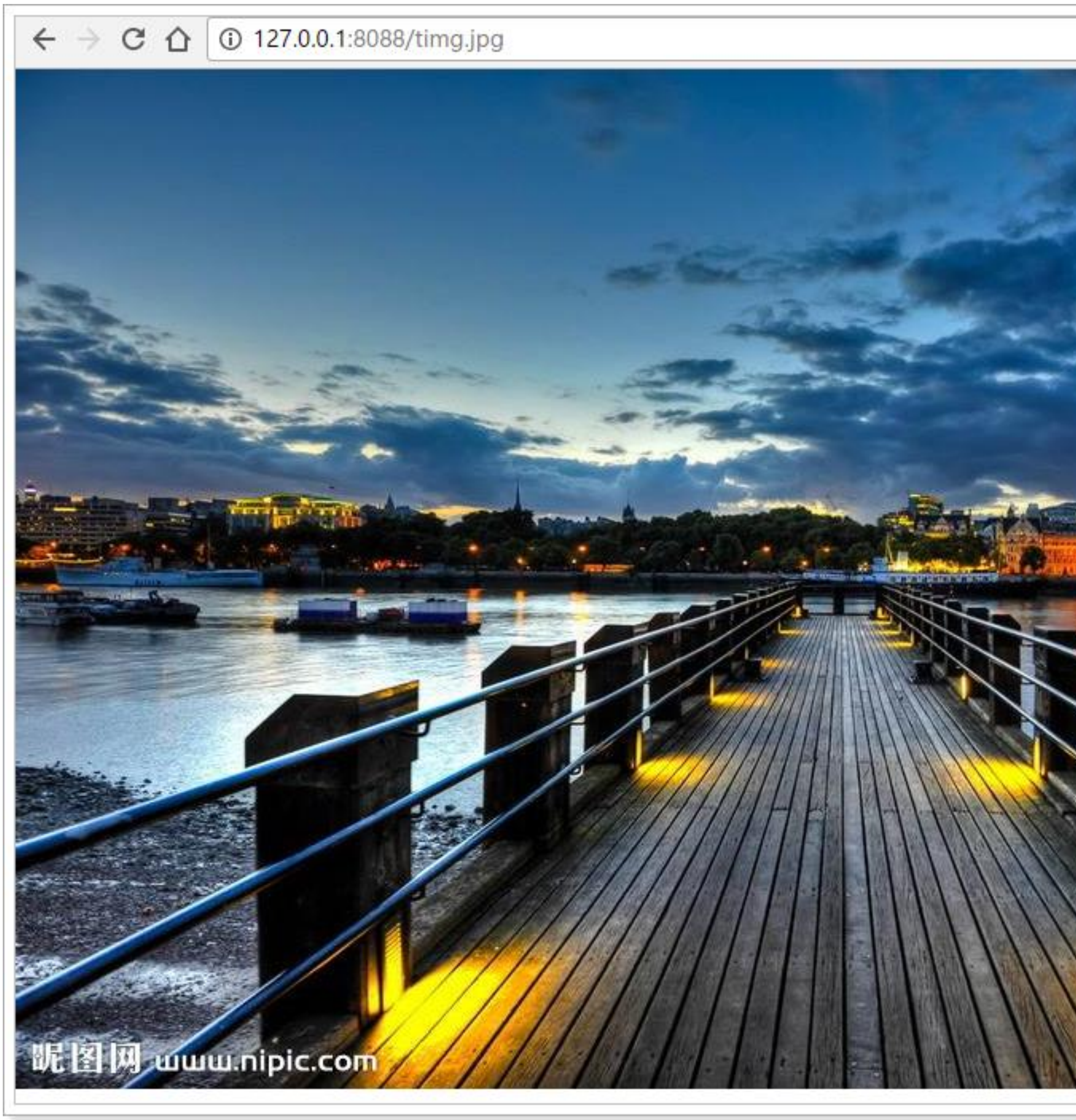
测试：



```
server.port=8088
server.servlet-path=/

#设置日志级别
logging.level.org.springframework=DEBUG

spring.resources.static-locations=classpath:/META-INF/reso
```

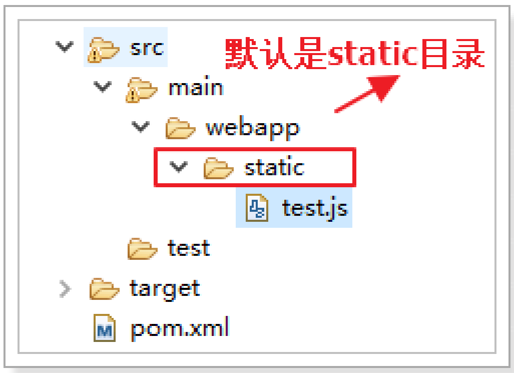


```
src/main/resources
  public
    timg.jpg
```

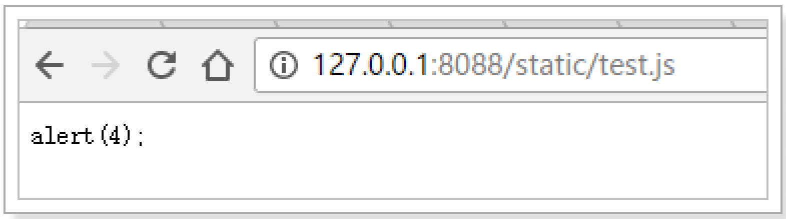


5.2.2. 进入规则为*.xxx 或者 不指定静态文件路径时

将静态资源放置到 webapp 下的 static 目录中即可通过地址访问：



测试：



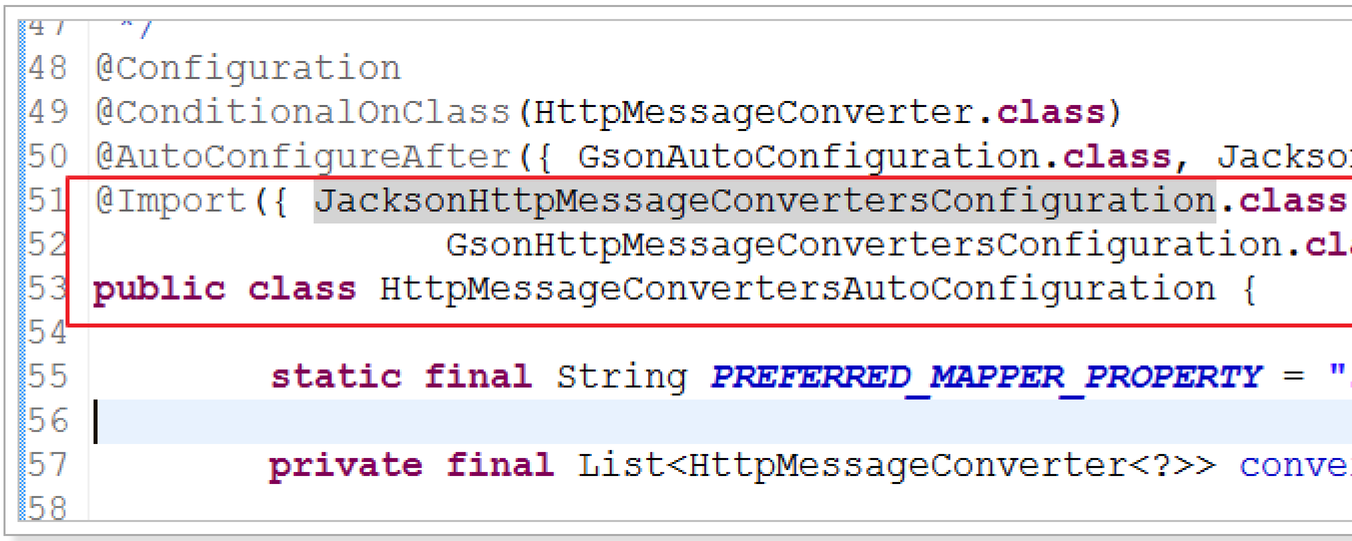
```
alert(4);
```

5.3. 自定义消息转化器

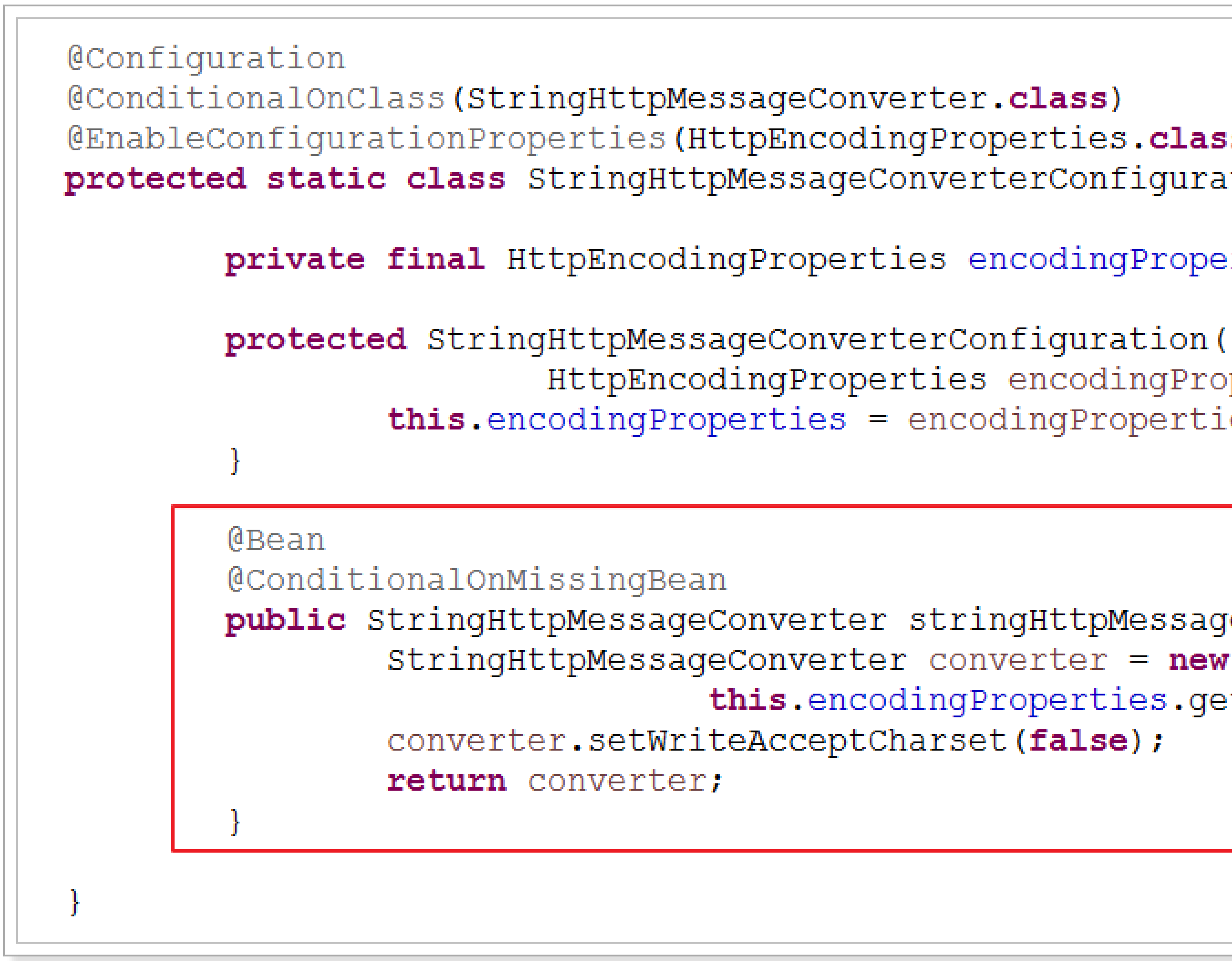
自定义消息转化器，只需要在@Configuration 的类中添加消息转化器的@bean 加入到 Spring 容器，就会被 Spring Boot 自动加入到容器中。

```
    @Bean
    public StringHttpMessageConverter
stringHttpMessageConverter(){
        StringHttpMessageConverter converter  = new
StringHttpMessageConverter(Charset.forName("UTF-8"));
        return converter;
    }
```

默认配置：



```
 */
@Configuration
@ConditionalOnClass(HttpMessageConverter.class)
@AutoConfigureAfter({ GsonAutoConfiguration.class, Jackso
@Import({ JacksonHttpMessageConvertersConfiguration.class
                GsonHttpMessageConvertersConfiguration.cl
public class HttpMessageConvertersAutoConfiguration {

        static final String PREFERRED_MAPPER_PROPERTY = "

        private final List<HttpMessageConverter<?>> conve

```



```
@Configuration
@ConditionalOnClass(StringHttpMessageConverter.class)
@EnableConfigurationProperties(HttpEncodingProperties.clas
protected static class StringHttpMessageConverterConfigura

        private final HttpEncodingProperties encodingPrope

        protected StringHttpMessageConverterConfiguration(
                        HttpEncodingProperties encodingPro
                this.encodingProperties = encodingPropertie
        }

        @Bean
        @ConditionalOnMissingBean
        public StringHttpMessageConverter stringHttpMessag
                StringHttpMessageConverter converter = new
                                this.encodingProperties.ge
                converter.setWriteAcceptCharset(false);
                return converter;
        }

}
```

5.4. 自定义 SpringMVC 的配置

有些时候我们需要自已配置 SpringMVC 而不是采用默认，比如说增加一个拦截器，这个时候

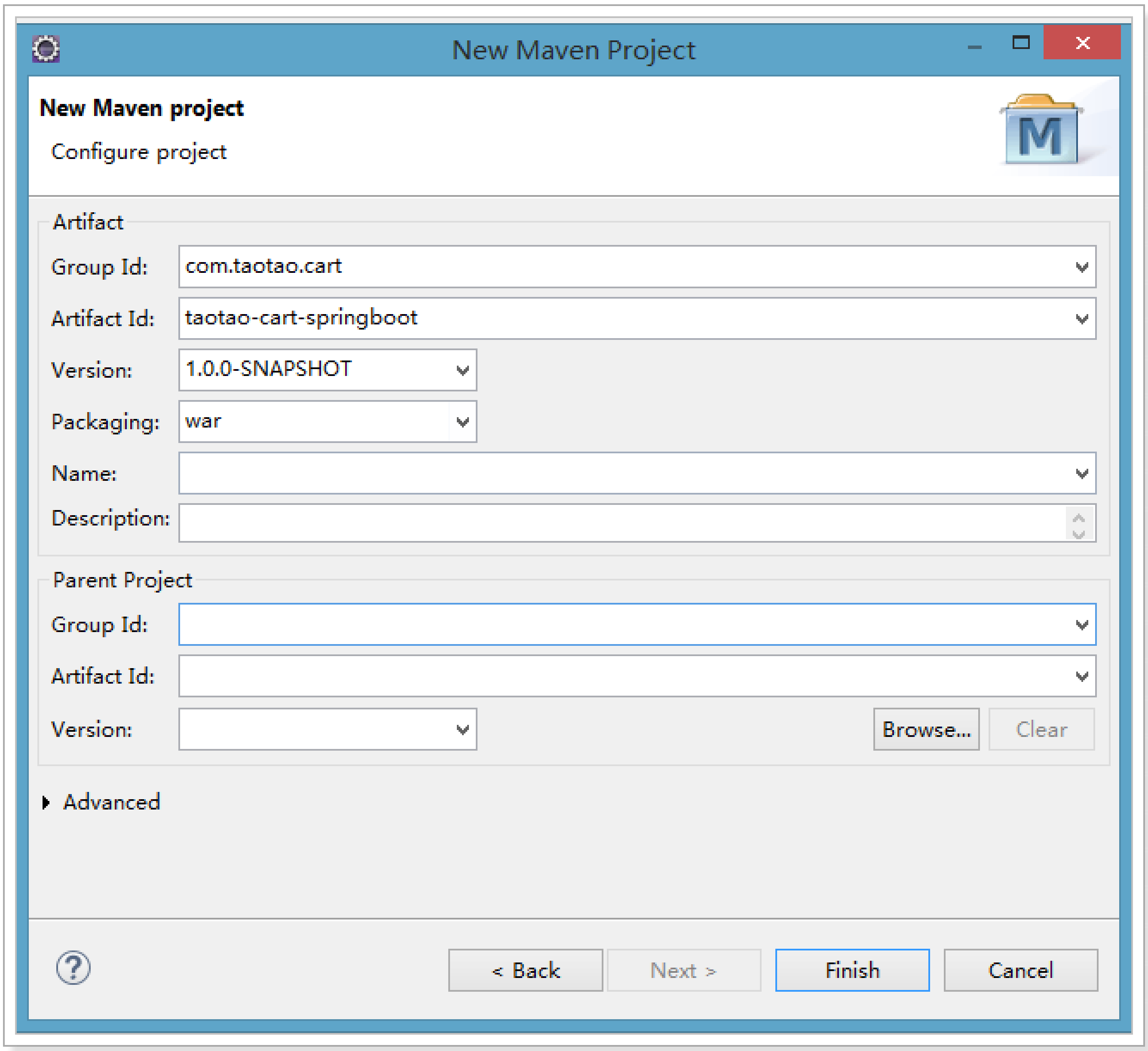
就得通过继承 WebMvcConfigurerAdapter 然后重写父类中的方法进行扩展。

```
import java.nio.charset.Charset;
import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import
org.springframework.context.annotation.Configuration;
import
org.springframework.http.converter.HttpMessageConverter;
import
org.springframework.http.converter.StringHttpMessageConve
rter;
import
org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;
import
org.springframework.web.servlet.config.annotation.Interce
ptorRegistry;
import
org.springframework.web.servlet.config.annotation.WebMvcC
onfigurerAdapter;


@Configuration //申明这是一个配置

public class MySrpingMVCConfig extends
WebMvcConfigurerAdapter{


   // 自定义拦截器

   @Override
   public void addInterceptors(InterceptorRegistry
registry) {
      HandlerInterceptor handlerInterceptor = new
HandlerInterceptor() {
          @Override
          public boolean preHandle(HttpServletRequest
request, HttpServletResponse response, Object handler)
                throws Exception {

             System.out.println("自定义拦截

器............");
```

```
                return true;
            }

            @Override
            public void postHandle(HttpServletRequest 
request, HttpServletResponse response, Object handler,
                    ModelAndView modelAndView)  throws 
Exception {

            }

            @Override
            public void afterCompletion(HttpServletRequest 
request, HttpServletResponse response, Object handler,
                    Exception ex)  throws Exception {
            }
        };

registry.addInterceptor(handlerInterceptor).addPathPatter
ns("/**");
    }


    // 自定义消息转化器的第二种方法
    @Override
    public void 
configureMessageConverters(List<HttpMessageConverter<?>> 
converters) {
        StringHttpMessageConverter converter  = new 
StringHttpMessageConverter(Charset.forName("UTF-8"));
        converters.add(converter);
    }

}
```

6. 改造购物车系统

6.1. 创建购物车的 Spring Boot 工程



Group Id: com.taotao.cart
Artifact Id: taotao-cart-springboot
Version: 1.0.0-SNAPSHOT
Packaging: war
Name:
Description:

Parent Project
Group Id:
Artifact Id:
Version:

6.2. 导入依赖

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.2.RELEASE</version>
    </parent>
    <groupId>com.taotao.cart</groupId>
    <artifactId>taotao-cart-springboot</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>war</packaging>

    <dependencies>
        <dependency>
            <groupId>com.taotao.common</groupId>
            <artifactId>taotao-common</artifactId>
            <version>1.0.0-SNAPSHOT</version>
        </dependency>
        <dependency>
            <groupId>com.taotao.sso</groupId>
            <artifactId>taotao-sso-interface</artifactId>
            <version>1.0.0-SNAPSHOT</version>
        </dependency>

        <!-- 单元测试 -->

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aspects</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Mybatis -->
        <dependency>
```

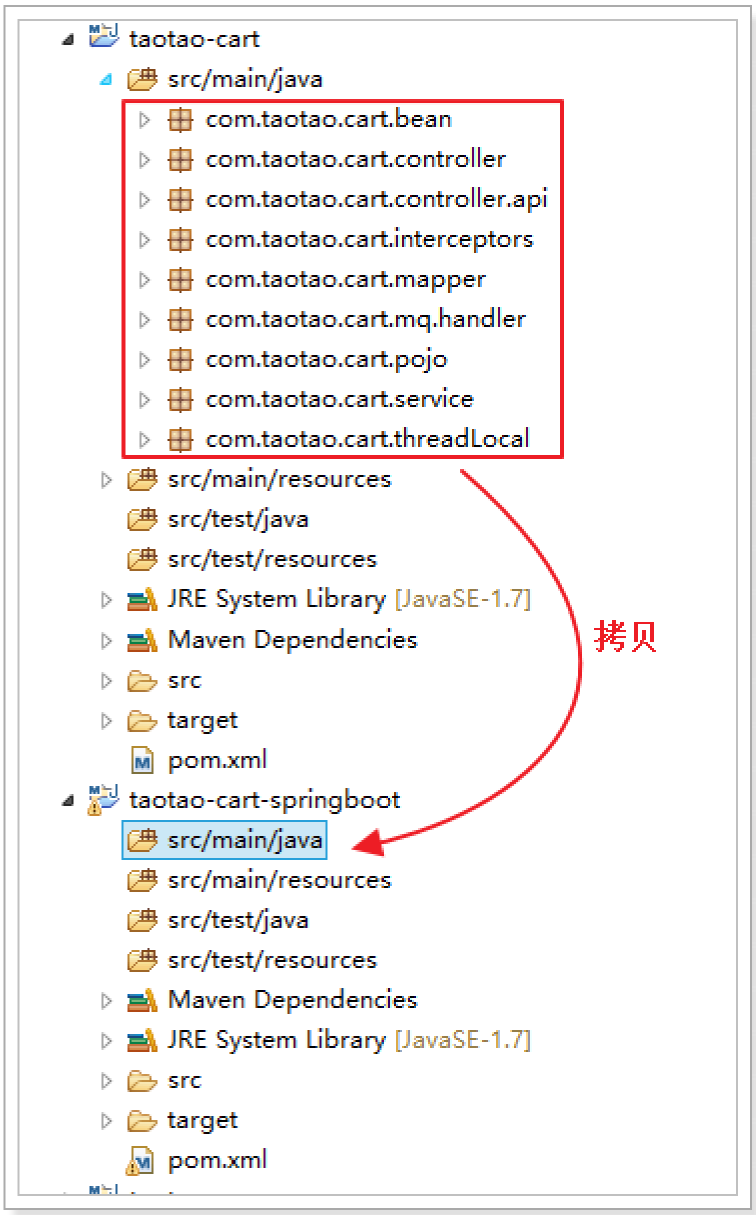
```
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.2.8</version>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>1.2.2</version>
        </dependency>
        <!-- 分页助手 -->
        <dependency>
            <groupId>com.github.pagehelper</groupId>
            <artifactId>pagehelper</artifactId>
            <version>3.7.5</version>
        </dependency>
        <dependency>
            <groupId>com.github.jsqlparser</groupId>
            <artifactId>jsqlparser</artifactId>
            <version>0.9.1</version>
        </dependency>


        <!-- 通用Mapper -->
        <dependency>
            <groupId>com.github.abel533</groupId>
            <artifactId>mapper</artifactId>
            <version>2.3.4</version>
        </dependency>
        <!-- MySql -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
        </dependency>
        <!-- 连接池 -->
        <dependency>
            <groupId>com.jolbox</groupId>
            <artifactId>bonecp-spring</artifactId>
            <version>0.8.0.RELEASE</version>
```

```xml
    </dependency>

    <!-- httpclient -->
    <dependency>
       <groupId>org.apache.httpcomponents</groupId>
       <artifactId>httpclient</artifactId>
    </dependency>


    <!-- JSP相关 -->

    <dependency>
       <groupId>jstl</groupId>
       <artifactId>jstl</artifactId>
       <version>1.2</version>
    </dependency>


    <!-- Apache工具组件 -->

    <dependency>
       <groupId>org.apache.commons</groupId>
       <artifactId>commons-lang3</artifactId>
       <version>3.3.2</version>
    </dependency>
    <dependency>
       <groupId>org.apache.commons</groupId>
       <artifactId>commons-io</artifactId>
       <version>1.3.2</version>
    </dependency>
    <dependency>
       <groupId>commons-codec</groupId>
       <artifactId>commons-codec</artifactId>
    </dependency>
    <dependency>
       <groupId>org.springframework.amqp</groupId>
       <artifactId>spring-rabbit</artifactId>
       <version>1.4.0.RELEASE</version>
    </dependency>

    <dependency>
       <groupId>com.alibaba</groupId>
       <artifactId>dubbo</artifactId>
       <version>2.5.3</version>
       <exclusions>
          <exclusion>
```

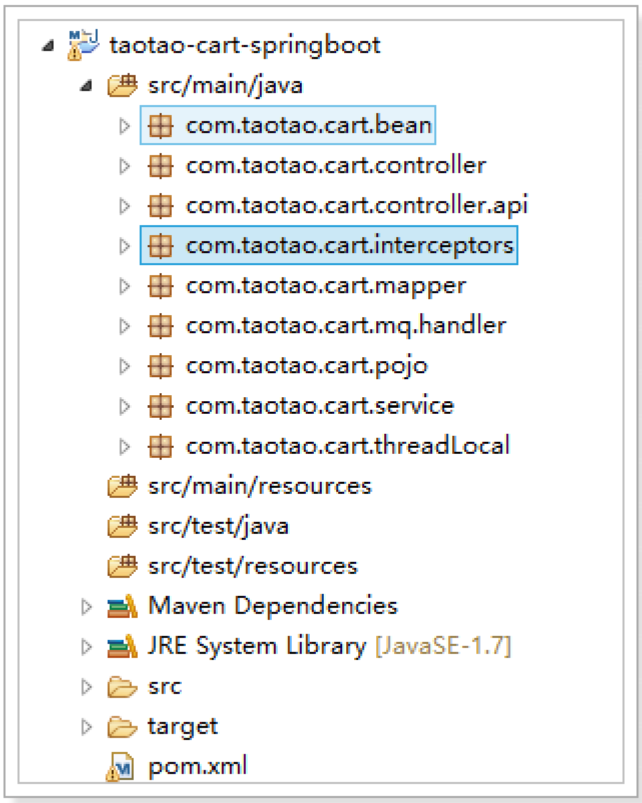
```xml
                <!-- 排除传递spring依赖 -->
                <artifactId>spring</artifactId>
                <groupId>org.springframework</groupId>
            </exclusion>
        </exclusions>
    </dependency>

    <dependency>
        <groupId>org.apache.zookeeper</groupId>
        <artifactId>zookeeper</artifactId>
        <version>3.3.3</version>
    </dependency>

    <dependency>
        <groupId>com.github.sgroschupf</groupId>
        <artifactId>zkclient</artifactId>
        <version>0.1</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <!-- 资源文件拷贝插件 -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-resources-
plugin</artifactId>
            <configuration>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
        <!-- java编译插件 -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>1.7</source>
                <target>1.7</target>
                <encoding>UTF-8</encoding>
            </configuration>
        </plugin>
```

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

6.3. 将 taotao-cart 中的 java 代码拷贝到 taotao-car-springboot

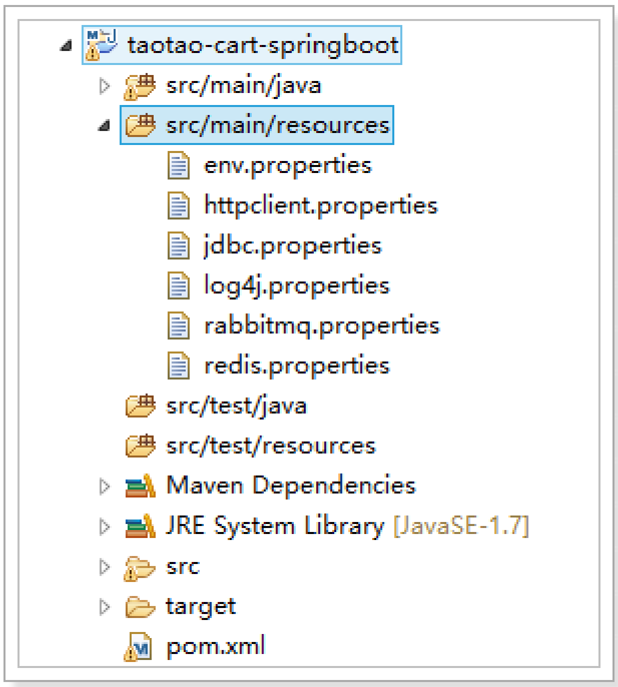

taotao-cart
src/main/java
com.taotao.cart.bean
com.taotao.cart.controller
com.taotao.cart.controller.api
com.taotao.cart.interceptors
com.taotao.cart.mapper
com.taotao.cart.mq.handler
com.taotao.cart.pojo
com.taotao.cart.service
com.taotao.cart.threadLocal
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
target
pom.xml
taotao-cart-springboot
src/main/java
src/main/resources
src/test/java
src/test/resources
Maven Dependencies
JRE System Library [JavaSE-1.7]
src
target
pom.xml

拷贝完成后：



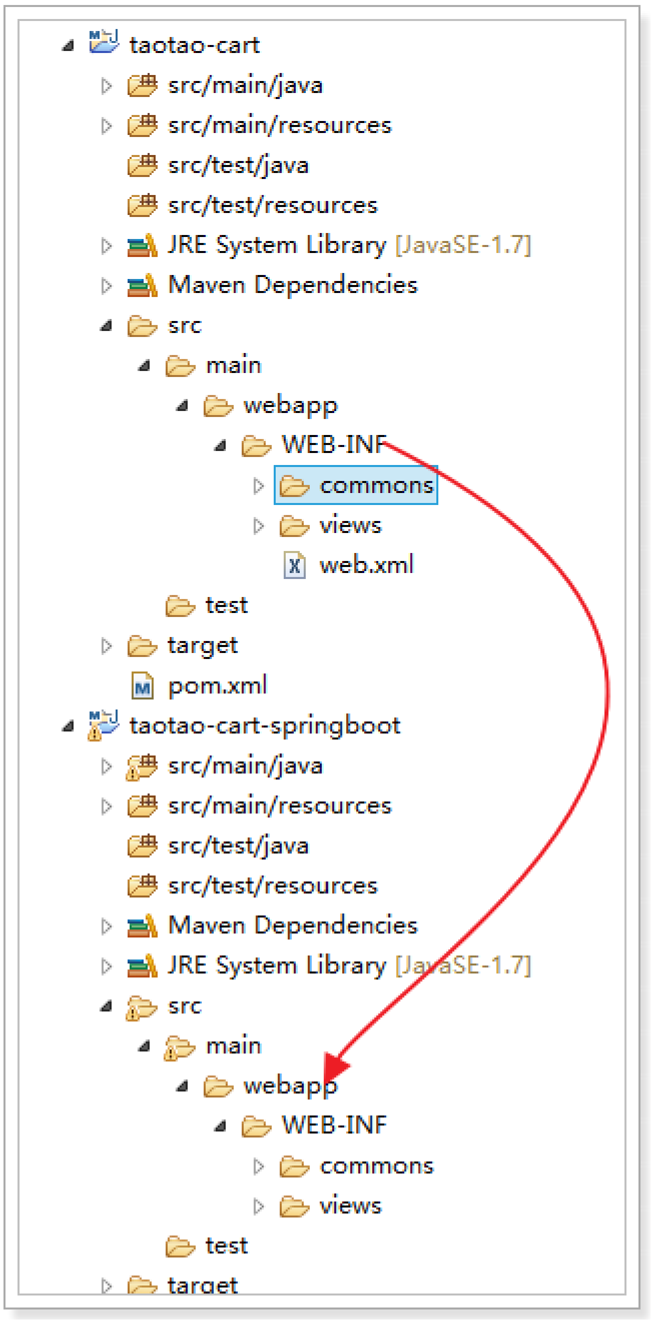
```
◢ taotao-cart-springboot
  ◢ src/main/java
    ▷ com.taotao.cart.bean
    ▷ com.taotao.cart.controller
    ▷ com.taotao.cart.controller.api
    ▷ com.taotao.cart.interceptors
    ▷ com.taotao.cart.mapper
    ▷ com.taotao.cart.mq.handler
    ▷ com.taotao.cart.pojo
    ▷ com.taotao.cart.service
    ▷ com.taotao.cart.threadLocal
    src/main/resources
    src/test/java
    src/test/resources
  ▷ Maven Dependencies
  ▷ JRE System Library [JavaSE-1.7]
  ▷ src
  ▷ target
    pom.xml
```

并且将 properties 文件也拷贝过来：



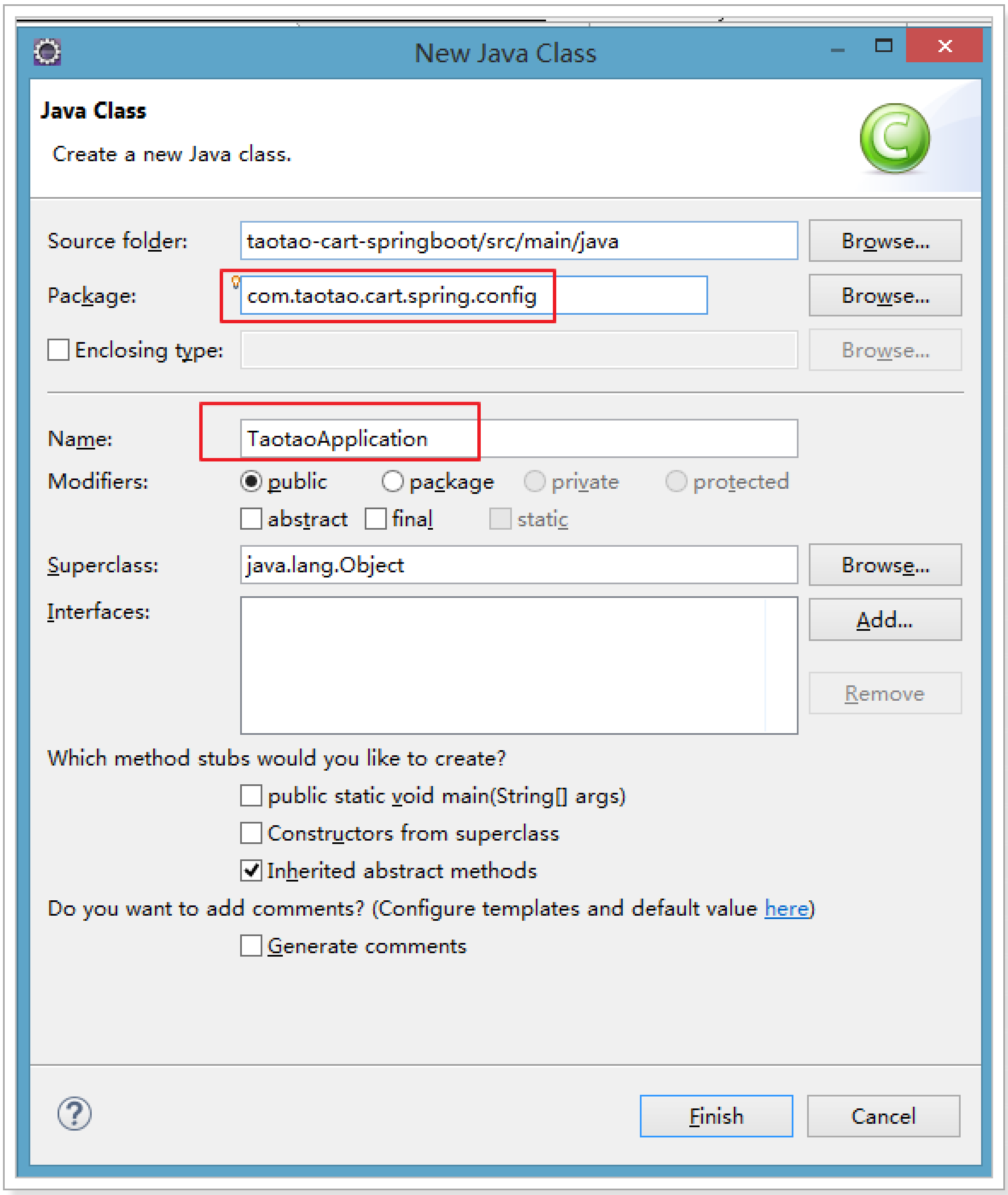
```
◢ taotao-cart-springboot
  ▷ src/main/java
  ◢ src/main/resources
      env.properties
      httpclient.properties
      jdbc.properties
      log4j.properties
      rabbitmq.properties
      redis.properties
    src/test/java
    src/test/resources
  ▷ Maven Dependencies
  ▷ JRE System Library [JavaSE-1.7]
  ▷ src
  ▷ target
    pom.xml
```

将页面也拷贝过来：



taotao-cart
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-1.7]
Maven Dependencies
src
main
webapp
WEB-INF
commons
views
web.xml
test
target
pom.xml
taotao-cart-springboot
src/main/java
src/main/resources
src/test/java
src/test/resources
Maven Dependencies
JRE System Library [JavaSE-1.7]
src
main
webapp
WEB-INF
commons
views
test
target

6.3.1. 编写 Spring 配置类 TaotaoApplication

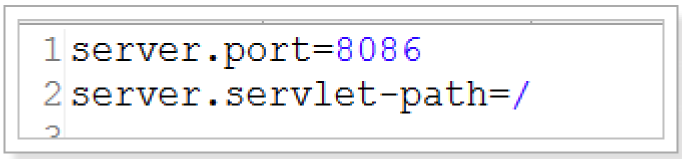


Source folder: taotao-cart-springboot/src/main/java
Package: com.taotao.cart.spring.config
Enclosing type: [ ]
Name: TaotaoApplication
Modifiers: [✓ public] package private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
[ ] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
[ ] Generate comments

6.3.2. 设置 tomcat 端口

application.properties：

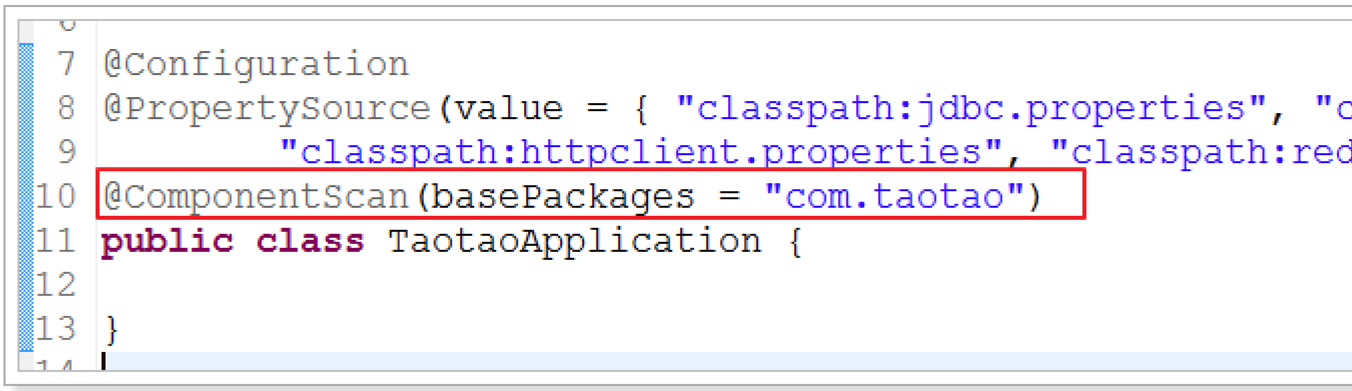


```
server.port=8086
server.servlet-path=/
```

6.3.3. 读取外部的配置文件

```
@Configuration
@PropertySource(value = { "classpath:jdbc.properties",
"classpath:env.properties",
        "classpath:httpclient.properties",
"classpath:redis.properties",
"classpath:rabbitmq.properties" }, ignoreResourceNotFound
= true)
public class TaotaoApplication {

}
```

6.3.4. 设置扫描包



```
@Configuration
@PropertySource(value = { "classpath:jdbc.properties", "c
         "classpath:httpclient.properties", "classpath:red
@ComponentScan(basePackages = "com.taotao")
public class TaotaoApplication {

}
```

6.3.5. 定义数据源

```
    @Value("${jdbc.url}")
   private String jdbcUrl;

   @Value("${jdbc.driverClassName}")
   private String jdbcDriverClassName;

   @Value("${jdbc.username}")
   private String jdbcUsername;
```

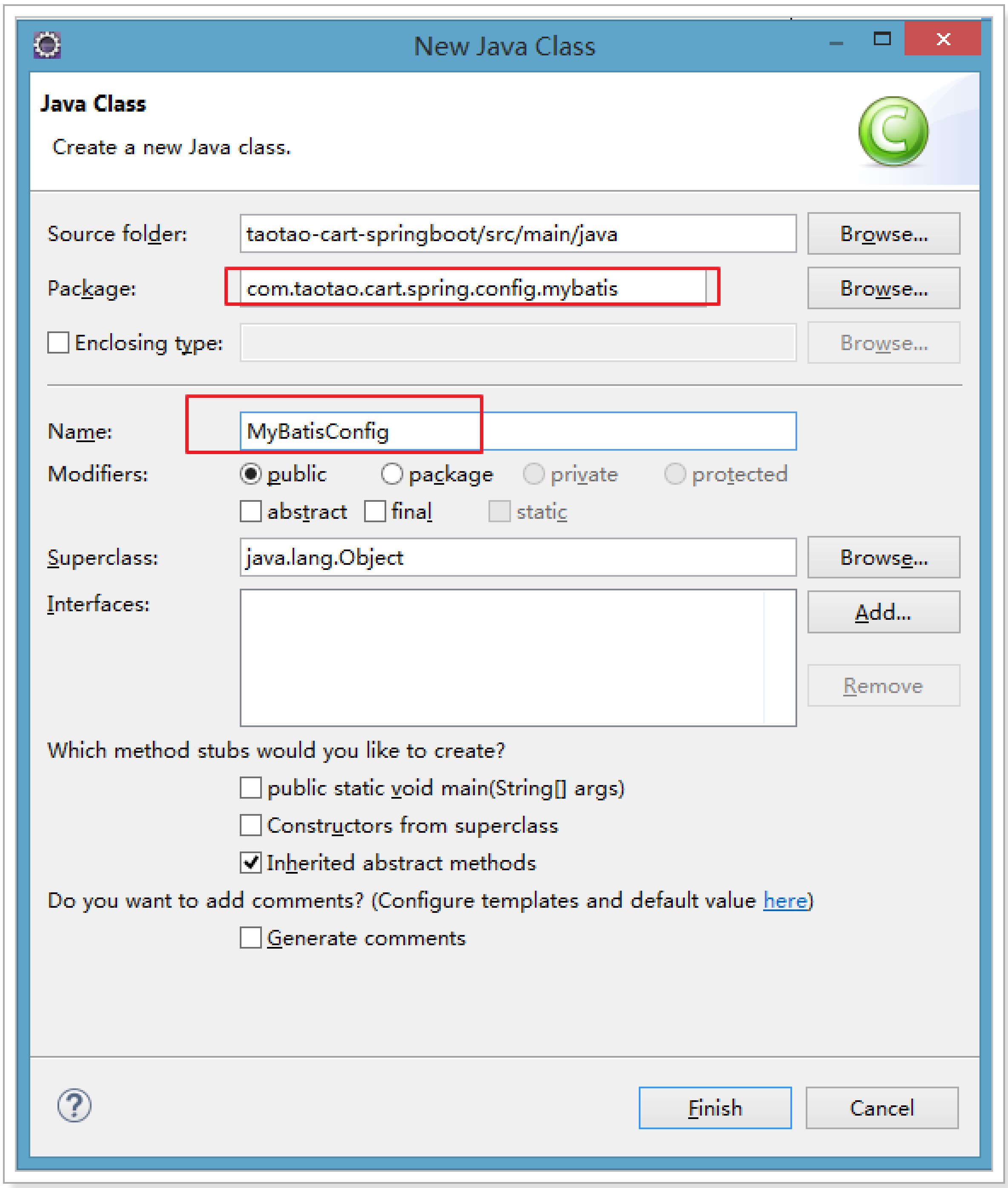
```
    @Value("${jdbc.password}")
    private String jdbcPassword;

    @Bean(destroyMethod = "close")
    public DataSource dataSource() {
        BoneCPDataSource boneCPDataSource = new
BoneCPDataSource();
        // 数据库驱动

boneCPDataSource.setDriverClass(jdbcDriverClassName);
        // 相应驱动的jdbcUrl
        boneCPDataSource.setJdbcUrl(jdbcUrl);
        // 数据库的用户名
        boneCPDataSource.setUsername(jdbcUsername);
        // 数据库的密码
        boneCPDataSource.setPassword(jdbcUsername);
        // 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：
240，如果要取消则设置为0

boneCPDataSource.setIdleConnectionTestPeriodInMinutes(60)
;
        // 连接池中未使用的链接最大存活时间，单位是分，默认值：60，
如果要永远存活设置为0
        boneCPDataSource.setIdleMaxAgeInMinutes(30);
        // 每个分区最大的连接数

boneCPDataSource.setMaxConnectionsPerPartition(100);
        // 每个分区最小的连接数
        boneCPDataSource.setMinConnectionsPerPartition(5);
        return boneCPDataSource;
    }
```

6.3.6. 设置 Mybatis 和 Spring Boot 整合

Mybatis 和 Spring Boot 的整合有两种方式：
第一种：使用 mybatis 官方提供的 Spring Boot 整合包实现，地址：https://github.com/mybatis/spring-boot-starter
第二种：使用 mybatis-spring 整合的方式，也就是我们传统的方式

这里我们推荐使用第二种，因为这样我们可以很方便的控制 Mybatis 的各种配置。

首先，创建一个 Mybatis 的配置类：


New Java Class

Java Class

Create a new Java class.

Source folder: taotao-cart-springboot/src/main/java

Package: com.taotao.cart.spring.config.mybatis

☐ Enclosing type:

Name: MyBatisConfig

Modifiers: ● public ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

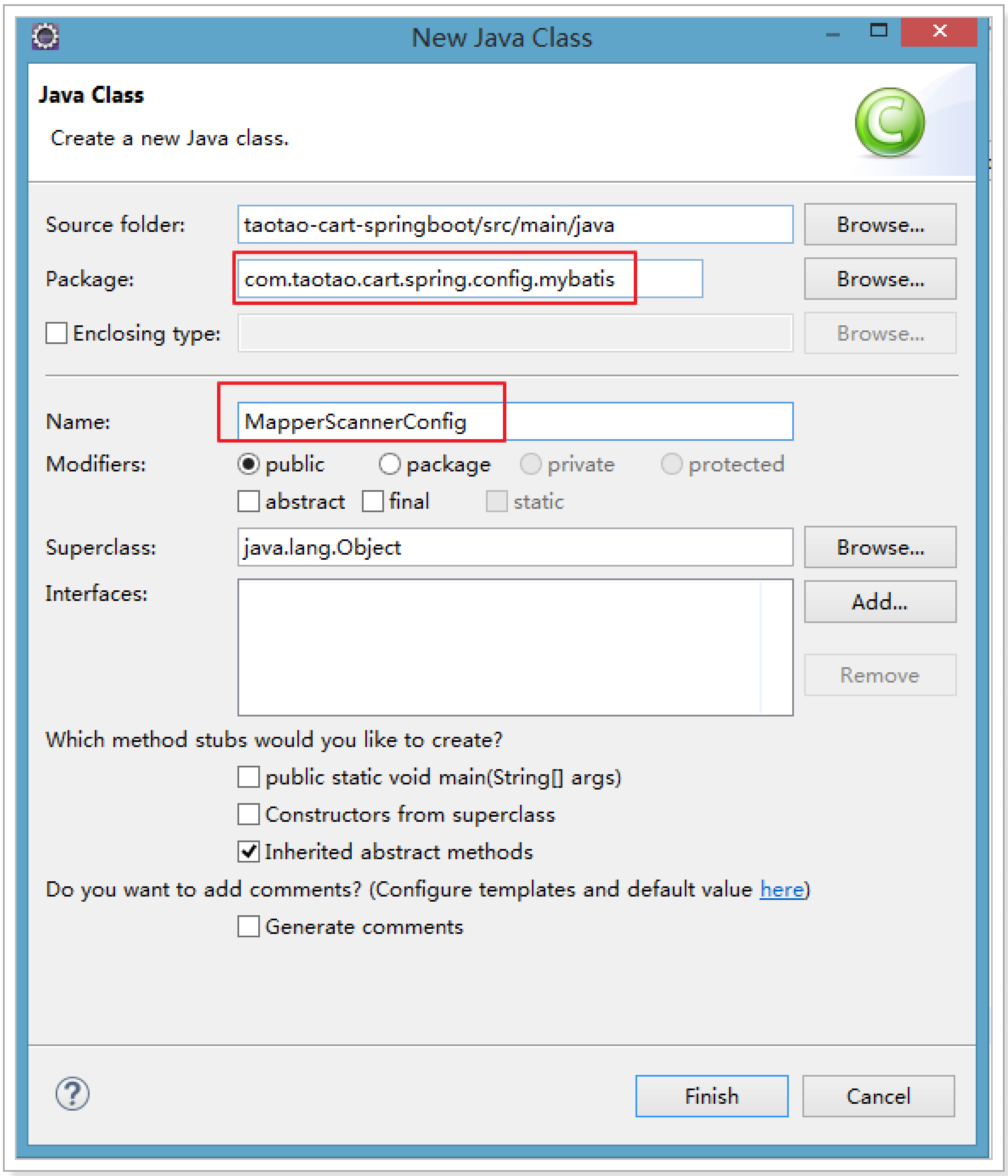
☐ Generate comments

代码：

```
import javax.sql.DataSource;

import org.mybatis.spring.SqlSessionFactoryBean;
import
org.springframework.boot.autoconfigure.condition.Conditio
nalOnMissingBean;
import org.springframework.context.annotation.Bean;
import
```

```
org.springframework.context.annotation.Configuration;
import org.springframework.core.io.Resource;
import org.springframework.core.io.support.PathMatchingResourcePatternResolver;
import org.springframework.core.io.support.ResourcePatternResolver;

@Configuration
public class MyBatisConfig {

    @Bean
    @ConditionalOnMissingBean //当容器里没有指定的Bean的情况下创建该对象
    public SqlSessionFactoryBean sqlSessionFactory(DataSource dataSource) {
        SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
        // 设置数据源
        sqlSessionFactoryBean.setDataSource(dataSource);
        // 设置mybatis的主配置文件
        ResourcePatternResolver resolver = new PathMatchingResourcePatternResolver();
        Resource mybatisConfigXml = resolver.getResource("classpath:mybatis/mybatis-config.xml");
        sqlSessionFactoryBean.setConfigLocation(mybatisConfigXml);
        // 设置别名包
        sqlSessionFactoryBean.setTypeAliasesPackage("com.taotao.cart.pojo");
        return sqlSessionFactoryBean;
    }
}
```

然后，创建 Mapper 接口的扫描类 MapperScannerConfig：



New Java Class

Java Class

Create a new Java class.

Source folder: taotao-cart-springboot/src/main/java

Package: com.taotao.cart.spring.config.mybatis

Enclosing type:

Name: MapperScannerConfig

Modifiers: (● public) ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

代码：

```
import org.mybatis.spring.mapper.MapperScannerConfigurer;
import
org.springframework.boot.autoconfigure.AutoConfigureAfter
;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
```

```
@Configuration
@AutoConfigureAfter(MyBatisConfig.class) //保证在
MyBatisConfig实例化之后再实例化该类
public class MapperScannerConfig {

    // mapper接口的扫描器
    @Bean
    public MapperScannerConfigurer
mapperScannerConfigurer() {
        MapperScannerConfigurer mapperScannerConfigurer =
new MapperScannerConfigurer();

mapperScannerConfigurer.setBasePackage("com.taotao.cart.m
apper");
        return mapperScannerConfigurer;
    }
}
```

6.3.7. 设置事务管理

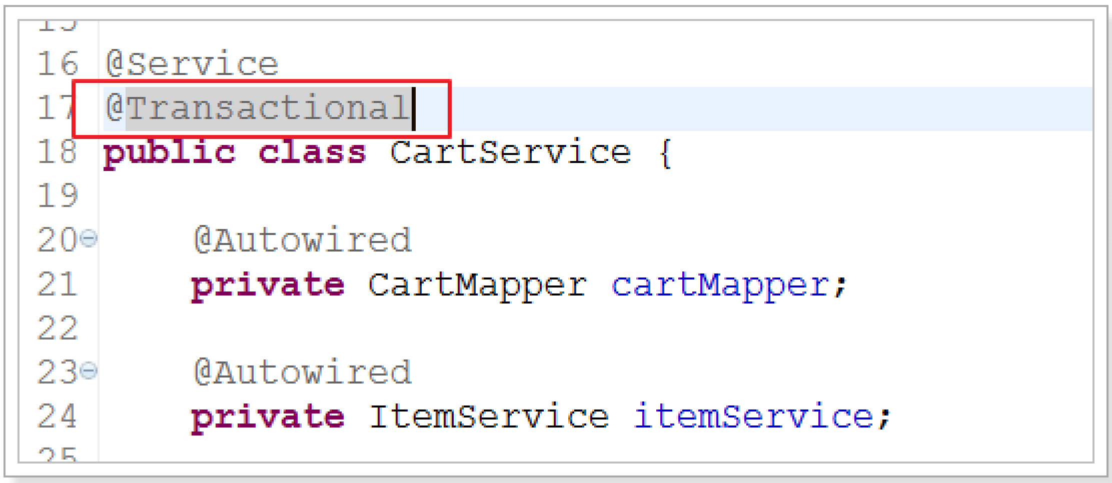
在 Spring Boot 中推荐使用@Transactional 注解来申明事务。

首先需要导入依赖：

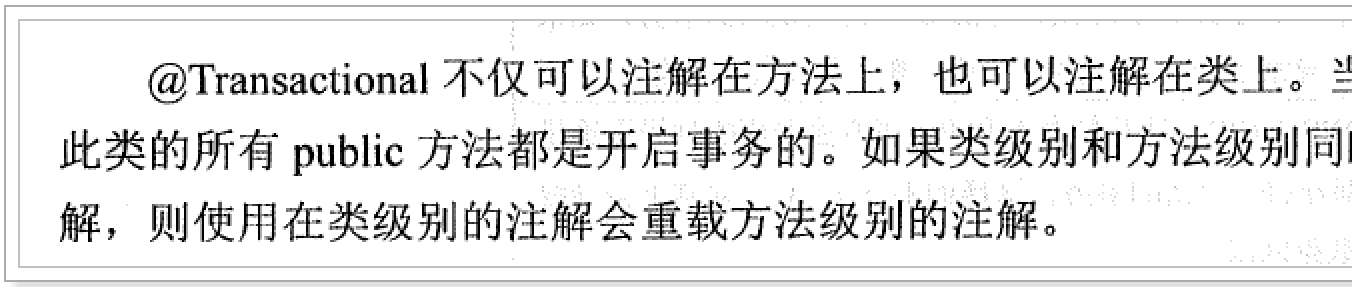
```
<dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-
jdbc</artifactId>
      </dependency>
```

当引入 jdbc 依赖之后，Spring Boot 会自动默认分别注入 DataSourceTransactionManager 或 JpaTransactionManager，所以我们不需要任何额外配置就可以用@Transactional 注解进行事务的使用。

在 Service 中添加@Transactional 注解：



```java
@Service
@Transactional
public class CartService {

    @Autowired
    private CartMapper cartMapper;

    @Autowired
    private ItemService itemService;
```



@Transactional 不仅可以注解在方法上，也可以注解在类上。
此类的所有 public 方法都是开启事务的。如果类级别和方法级别同
解，则使用在类级别的注解会重载方法级别的注解。

6.3.8. 设置 Redis 和 Spring 的整合

在 Spring Boot 中提供了 RedisTempplate 的操作，我们暂时不做学习，先按照我们之前的实现来完成。

代码：

```java
import java.util.ArrayList;
import java.util.List;

import
org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.context.annotation.PropertySource;

import redis.clients.jedis.JedisPoolConfig;
import redis.clients.jedis.JedisShardInfo;
import redis.clients.jedis.ShardedJedisPool;

@Configuration
```

```
@PropertySource(value = "classpath:redis.properties")
public class RedisSpringConfig {

    @Value("${redis.maxTotal}")
    private Integer redisMaxTotal;

    @Value("${redis.node1.host}")
    private String redisNode1Host;

    @Value("${redis.node1.port}")
    private Integer redisNode1Port;

    private JedisPoolConfig jedisPoolConfig() {
        JedisPoolConfig jedisPoolConfig = new
JedisPoolConfig();
        jedisPoolConfig.setMaxTotal(redisMaxTotal);
        return jedisPoolConfig;
    }

    @Bean
    public ShardedJedisPool shardedJedisPool() {
        List<JedisShardInfo> jedisShardInfos = new
ArrayList<JedisShardInfo>();
        jedisShardInfos.add(new
JedisShardInfo(redisNode1Host, redisNode1Port));
        return new ShardedJedisPool(jedisPoolConfig(),
jedisShardInfos);
    }
}
```

6.3.9. 设置 Httpclient 和 Spring 的整合

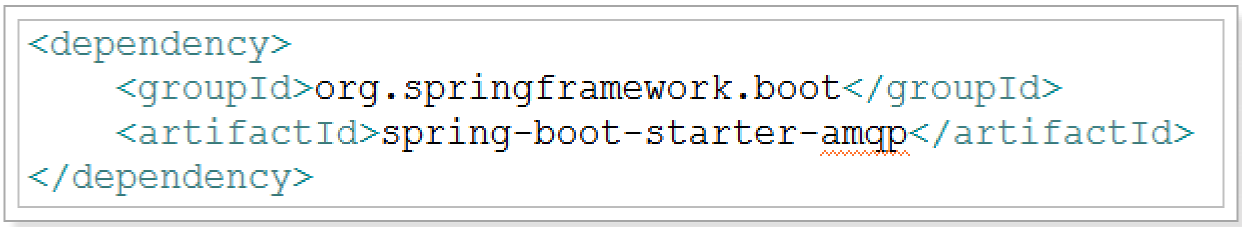
```
import org.apache.http.client.config.RequestConfig;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import
org.apache.http.impl.conn.PoolingHttpClientConnectionMana
ger;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import
```

```
org.springframework.context.annotation.Configuration;
import
org.springframework.context.annotation.PropertySource;
import org.springframework.context.annotation.Scope;

import
com.taotao.common.httpclient.IdleConnectionEvictor;

@Configuration
@PropertySource(value =
"classpath:httpclient.properties")
public class HttpclientSpringConfig {

    @Value("${http.maxTotal}")
    private Integer httpMaxTotal;

    @Value("${http.defaultMaxPerRoute}")
    private Integer httpDefaultMaxPerRoute;

    @Value("${http.connectTimeout}")
    private Integer httpConnectTimeout;

    @Value("${http.connectionRequestTimeout}")
    private Integer httpConnectionRequestTimeout;

    @Value("${http.socketTimeout}")
    private Integer httpSocketTimeout;

    @Value("${http.staleConnectionCheckEnabled}")
    private Boolean httpStaleConnectionCheckEnabled;

    @Autowired
    private PoolingHttpClientConnectionManager manager;

    @Bean
    public PoolingHttpClientConnectionManager
poolingHttpClientConnectionManager() {
        PoolingHttpClientConnectionManager
poolingHttpClientConnectionManager = new
PoolingHttpClientConnectionManager();
        // 最大连接数

poolingHttpClientConnectionManager.setMaxTotal(httpMaxTot
al);
        // 每个主机的最大并发数

poolingHttpClientConnectionManager.setDefaultMaxPerRoute(
httpDefaultMaxPerRoute);
        return poolingHttpClientConnectionManager;
    }

    // 定期关闭无效连接
    @Bean
    public IdleConnectionEvictor idleConnectionEvictor() {
        return new IdleConnectionEvictor(manager);
    }

    // 定义Httpclient对
    @Bean
    @Scope("prototype")
    public CloseableHttpClient closeableHttpClient() {
        return
HttpClients.custom().setConnectionManager(this.manager).b
uild();
    }

    // 请求配置
    @Bean
    public RequestConfig requestConfig() {
        return
RequestConfig.custom().setConnectTimeout(httpConnectTimeo
ut) // 创建连接的最长时间
                .setConnectionRequestTimeout(httpConnectionR
equestTimeout) // 从连接池中获取到连接的最长时间
                .setSocketTimeout(httpSocketTimeout) // 数据
传输的最长时间
                .~~setStaleConnectionCheckEnabled~~(httpStaleCon
nectionCheckEnabled) // 提交请求前测试连接是否可用
                .build();
    }

}
```

6.3.10. 设置 RabbitMQ 和 Spring 的整合

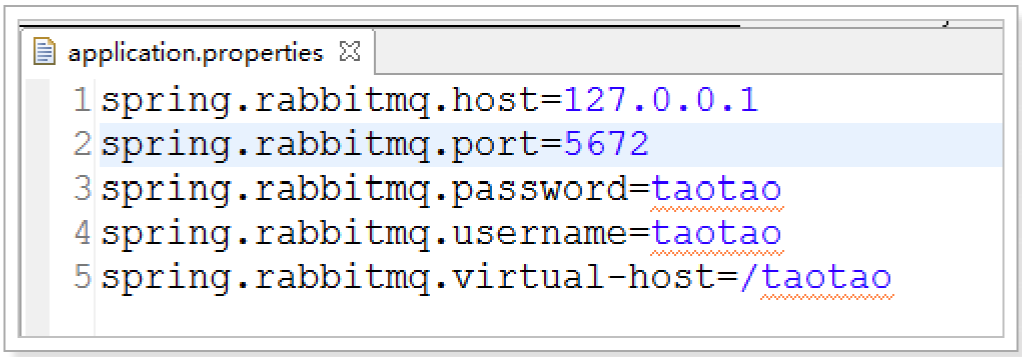
我们之前使用的 Spring-Rabbit 的 xml 方式，现在我们要改造成 java 方式，并且 Spring Boot 对 RabbitMQ 的使用做了自动配置，更加的简化了我们的使用。

1、 在导入 spring-boot-starter-amqp 的依赖；



```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

2、 在 application.properties 文件中配置 RabbitMQ 的连接信息

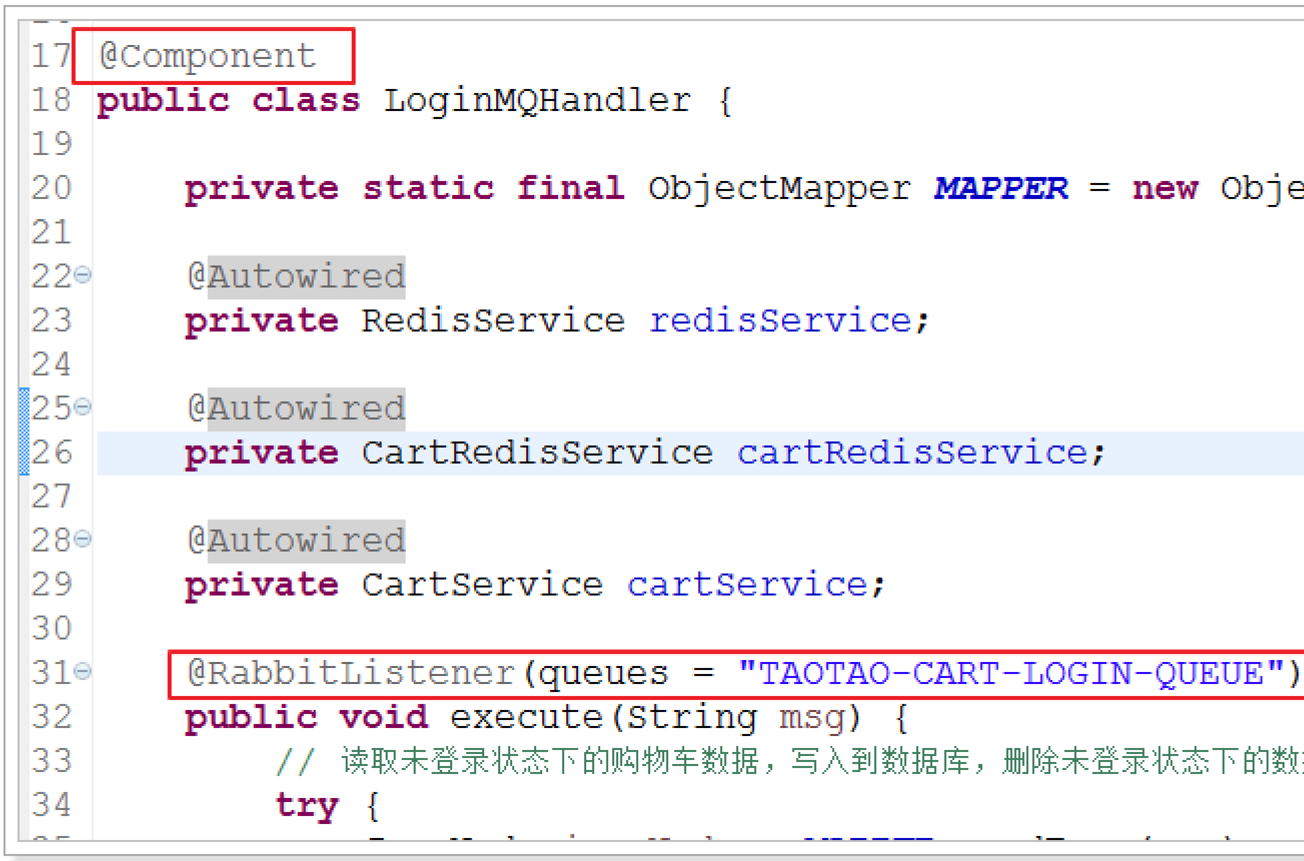


application.properties

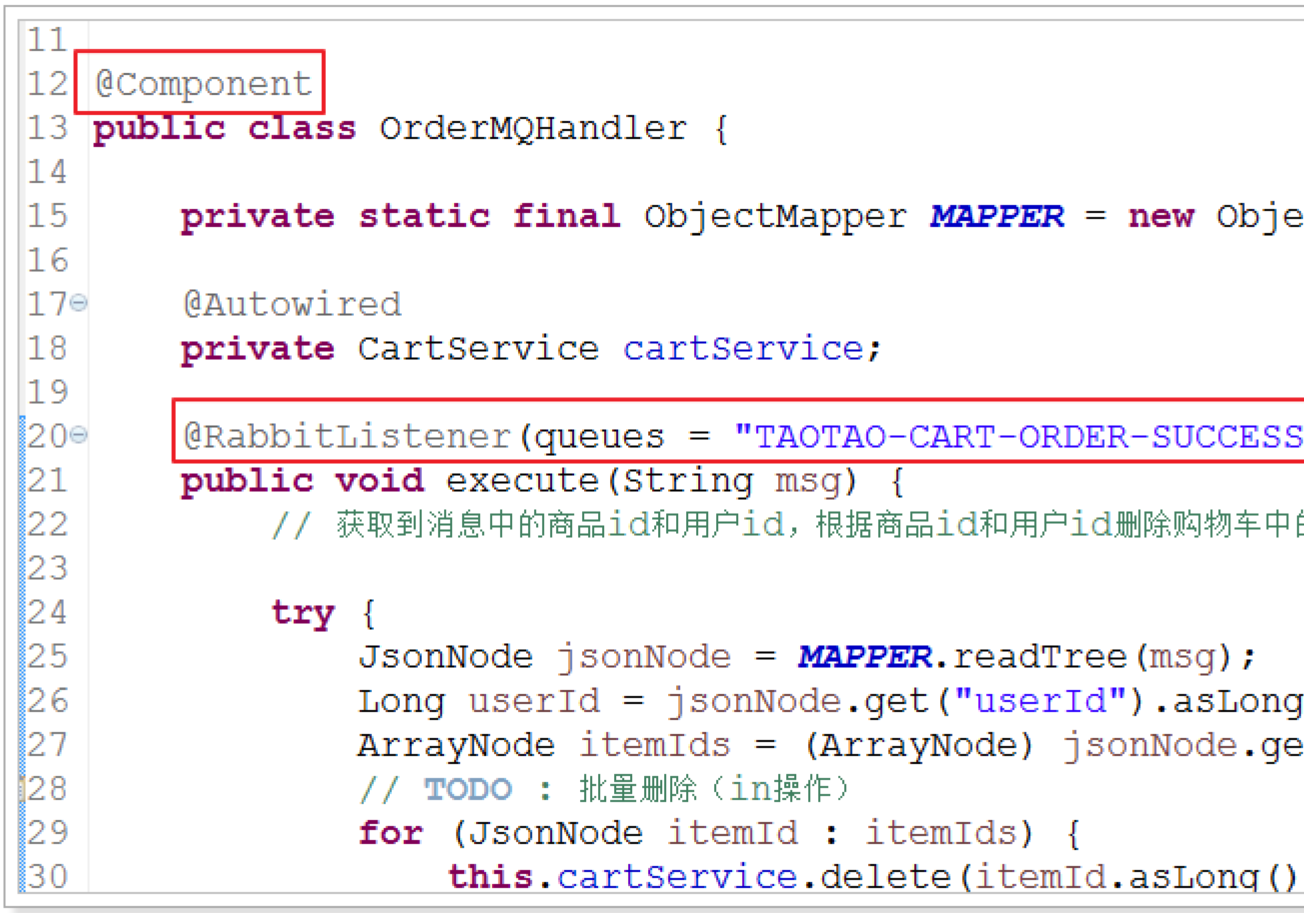
```
spring.rabbitmq.host=127.0.0.1
spring.rabbitmq.port=5672
spring.rabbitmq.password=taotao
spring.rabbitmq.username=taotao
spring.rabbitmq.virtual-host=/taotao
```

3、 编写Rabbit的Spring配置类

```java
import org.springframework.amqp.core.Queue;
import org.springframework.amqp.rabbit.connection.ConnectionFactory;
import org.springframework.amqp.rabbit.core.RabbitAdmin;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitMQSpringConfig {

    @Autowired
    private ConnectionFactory connectionFactory;

    // 管理
    @Bean
    public RabbitAdmin rabbitAdmin() {
        return new RabbitAdmin(connectionFactory);
    }

    // 声明队列
    @Bean
    public Queue taotaoCartLoginQueue() {
        // 默认就是自动声明的
        return new Queue("TAOTAO-CART-LOGIN-QUEUE",
true);
    }

    // 声明队列
    @Bean
    public Queue taotaoCartOrderSuccessQueue() {
        // 默认就是自动声明的
        return new Queue("TAOTAO-CART-ORDER-SUCCESS-
QUEUE", true);
    }

}
```
4、 设置监听



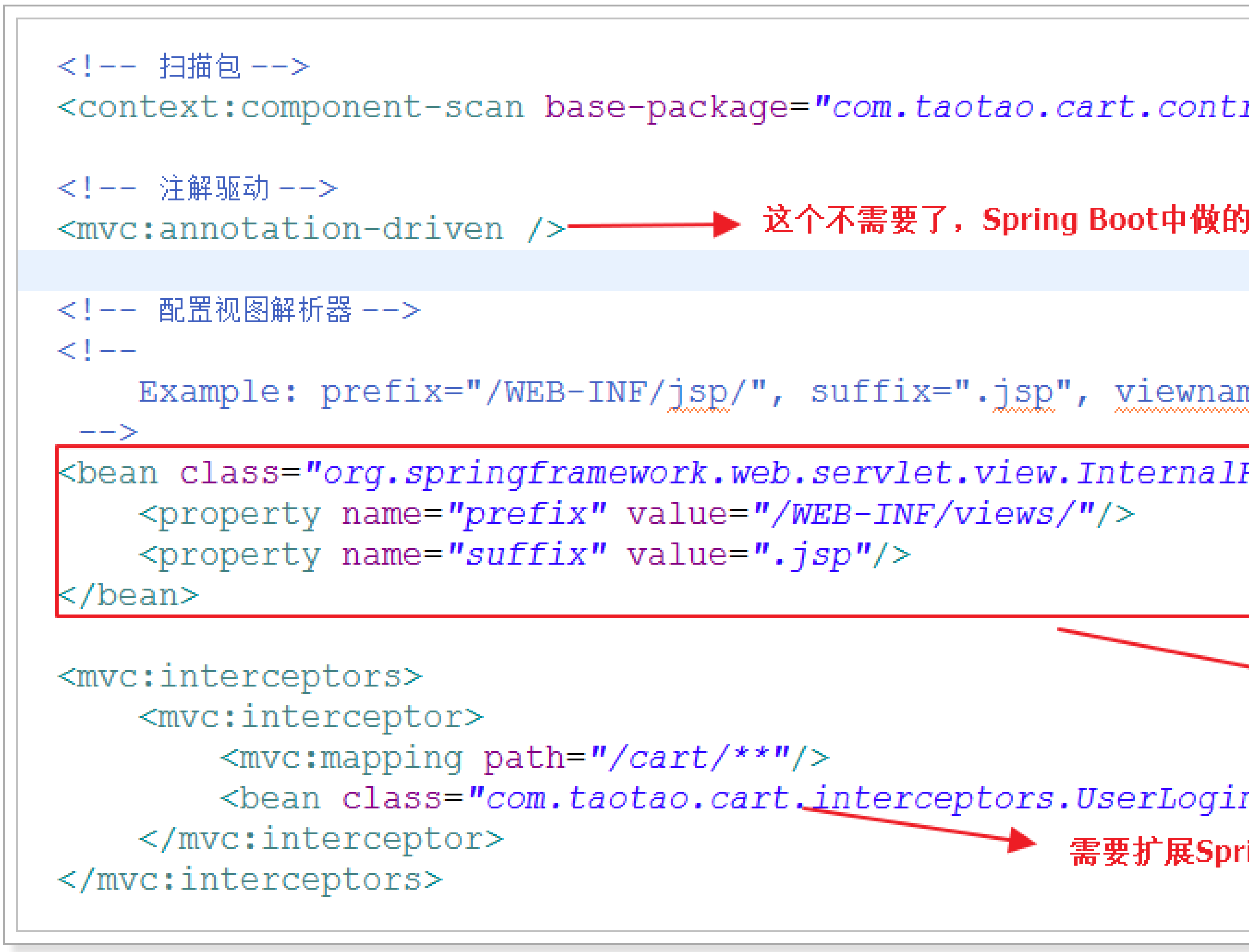
```java
@Component
public class LoginMQHandler {

    private static final ObjectMapper MAPPER = new Obje

    @Autowired
    private RedisService redisService;

    @Autowired
    private CartRedisService cartRedisService;

    @Autowired
    private CartService cartService;

    @RabbitListener(queues = "TAOTAO-CART-LOGIN-QUEUE")
    public void execute(String msg) {
        // 读取未登录状态下的购物车数据，写入到数据库，删除未登录状态下的数
        try {
```



```java
@Component
public class OrderMQHandler {

    private static final ObjectMapper MAPPER = new Obje

    @Autowired
    private CartService cartService;

    @RabbitListener(queues = "TAOTAO-CART-ORDER-SUCCESS
    public void execute(String msg) {
        // 获取到消息中的商品id和用户id，根据商品id和用户id删除购物车中的

        try {
            JsonNode jsonNode = MAPPER.readTree(msg);
            Long userId = jsonNode.get("userId").asLong
            ArrayNode itemIds = (ArrayNode) jsonNode.ge
            // TODO : 批量删除（in操作）
            for (JsonNode itemId : itemIds) {
                this.cartService.delete(itemId.asLong()
```

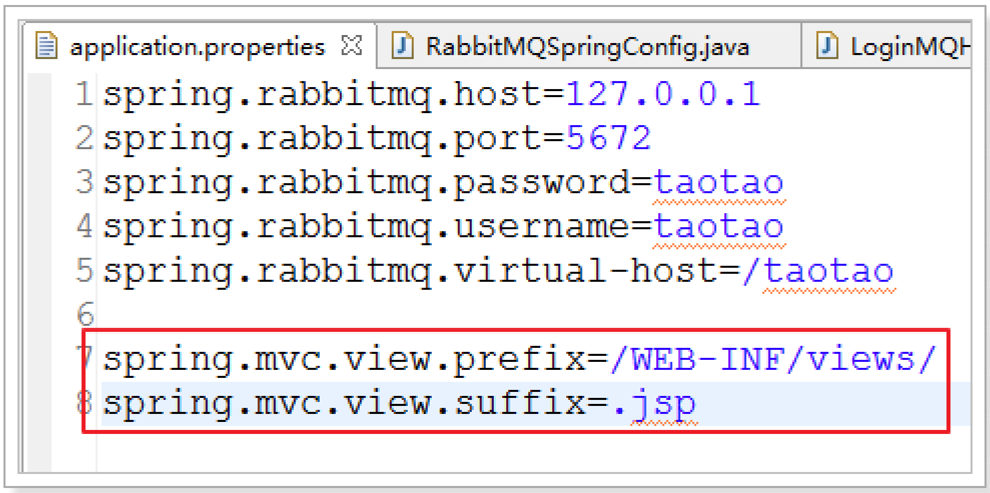
6.3.11. 设置 SpringMVC 的配置

原有配置：



```
<!-- 扫描包 -->
<context:component-scan base-package="com.taotao.cart.contr

<!-- 注解驱动 -->
<mvc:annotation-driven />

<!-- 配置视图解析器 -->
<!--
    Example: prefix="/WEB-INF/jsp/", suffix=".jsp", viewnam
 -->
<bean class="org.springframework.web.servlet.view.InternalR
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
</bean>

<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/cart/**"/>
        <bean class="com.taotao.cart.interceptors.UserLogin
    </mvc:interceptor>
</mvc:interceptors>
```

具体实现：
视图解析器配置：



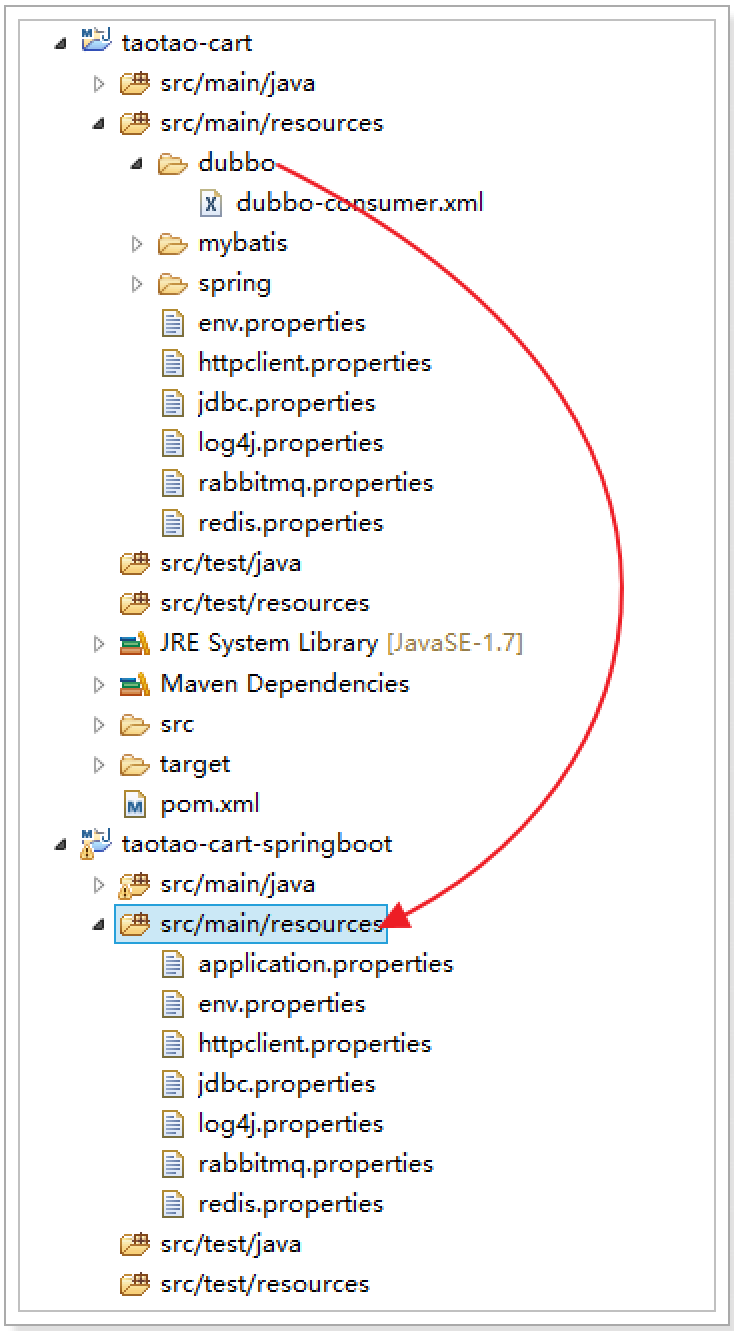
```
spring.rabbitmq.host=127.0.0.1
spring.rabbitmq.port=5672
spring.rabbitmq.password=taotao
spring.rabbitmq.username=taotao
spring.rabbitmq.virtual-host=/taotao

spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

自定义拦截器：
```
import
org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.Interce
ptorRegistry;
import
org.springframework.web.servlet.config.annotation.WebMvcC
onfigurerAdapter;

import
com.taotao.cart.interceptors.UserLoginHandlerInterceptor;

@Configuration
public class SpringMVCConfig extends
WebMvcConfigurerAdapter {

    @Override
    public void addInterceptors(InterceptorRegistry
registry) {
        // 判断用户是否登录的拦截器
        registry.addInterceptor(new
UserLoginHandlerInterceptor()).addPathPatterns("/cart/**"
);
    }

}
```

6.3.12. 设置 dubbo 的配置

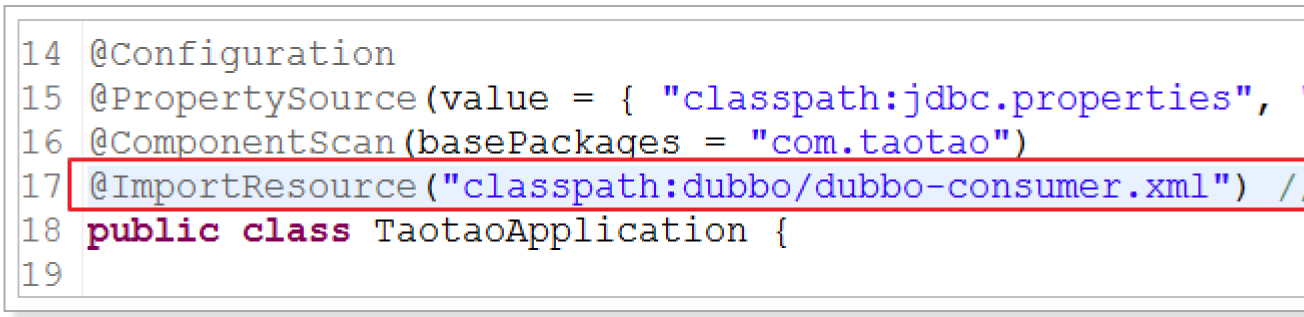
Dubbo 目前只能使用 xml 配置的方式，所以我们需要保留 xml，并且需要将该 xml 加入到现有的 Spring 容器中才能生效。

1、 将 dubbo 目录以及下面的 xml 配置文件拷贝到 taotao-cat-springboot 中



```
taotao-cart
    src/main/java
    src/main/resources
        dubbo
            dubbo-consumer.xml
        mybatis
        spring
        env.properties
        httpclient.properties
        jdbc.properties
        log4j.properties
        rabbitmq.properties
        redis.properties
    src/test/java
    src/test/resources
    JRE System Library [JavaSE-1.7]
    Maven Dependencies
    src
    target
    pom.xml
taotao-cart-springboot
    src/main/java
    src/main/resources
        application.properties
        env.properties
        httpclient.properties
        jdbc.properties
        log4j.properties
        rabbitmq.properties
        redis.properties
    src/test/java
    src/test/resources
```

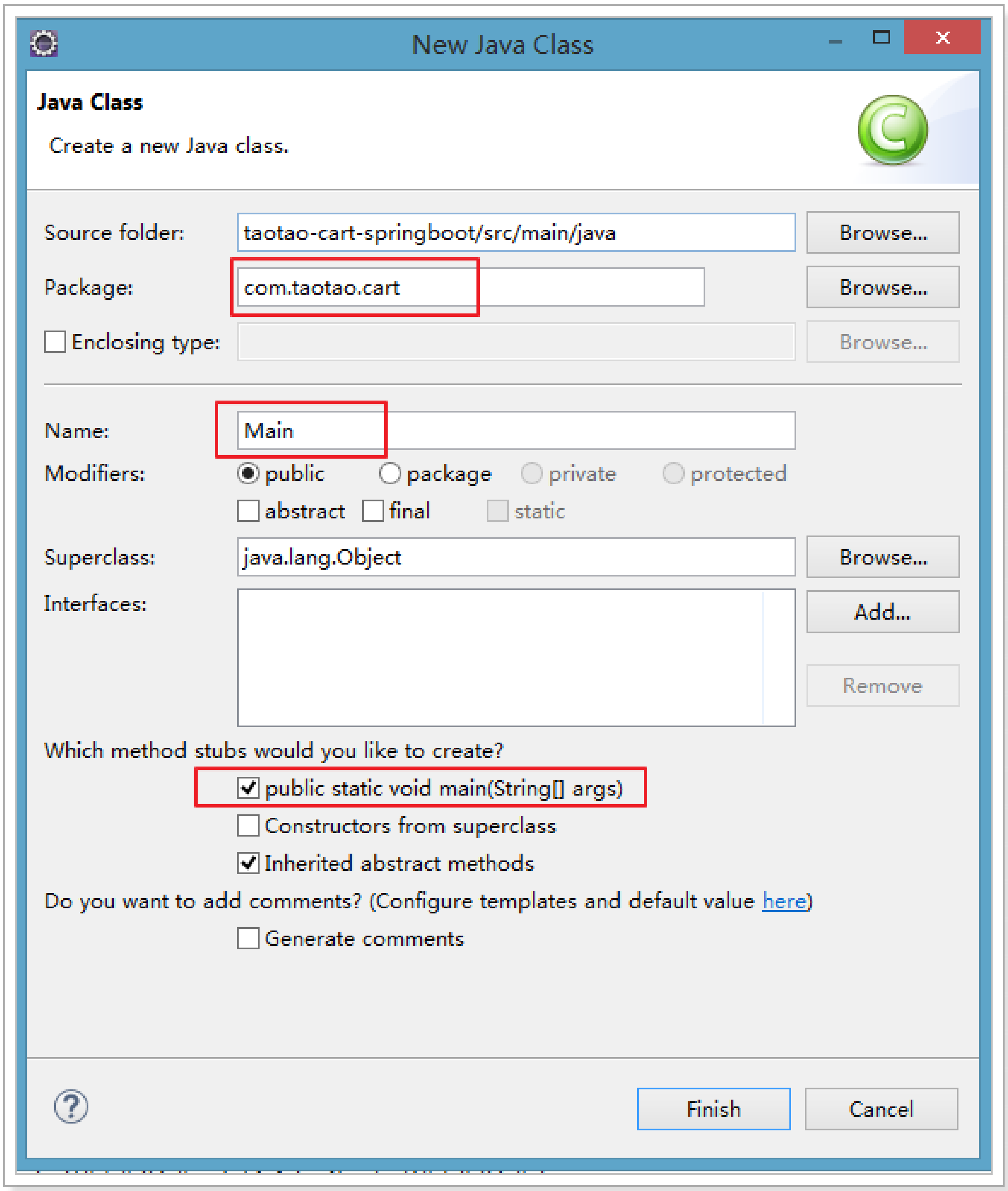
2、 将 dubbo 的 xml 文件加入到 spring 容器



```java
@Configuration
@PropertySource(value = { "classpath:jdbc.properties", 
@ComponentScan(basePackages = "com.taotao")
@ImportResource("classpath:dubbo/dubbo-consumer.xml") /
public class TaotaoApplication {

```

3、

6.4. 编写入口类


New Java Class

Java Class

Create a new Java class.

Source folder: taotao-cart-springboot/src/main/java

Package: com.taotao.cart

☐ Enclosing type:

Name: Main

Modifiers: ● public ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☑ public static void main(String[] args)

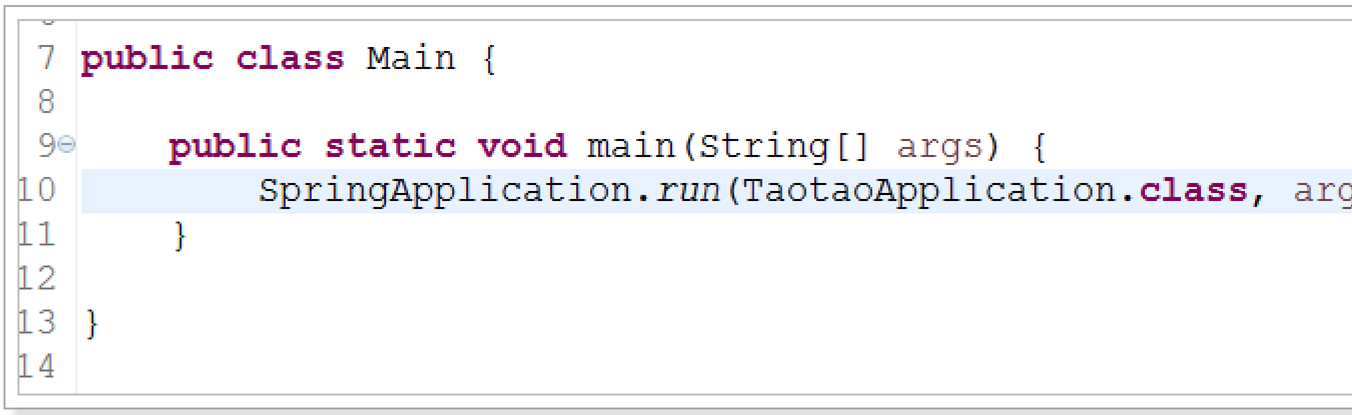
☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

编写 main 方法：



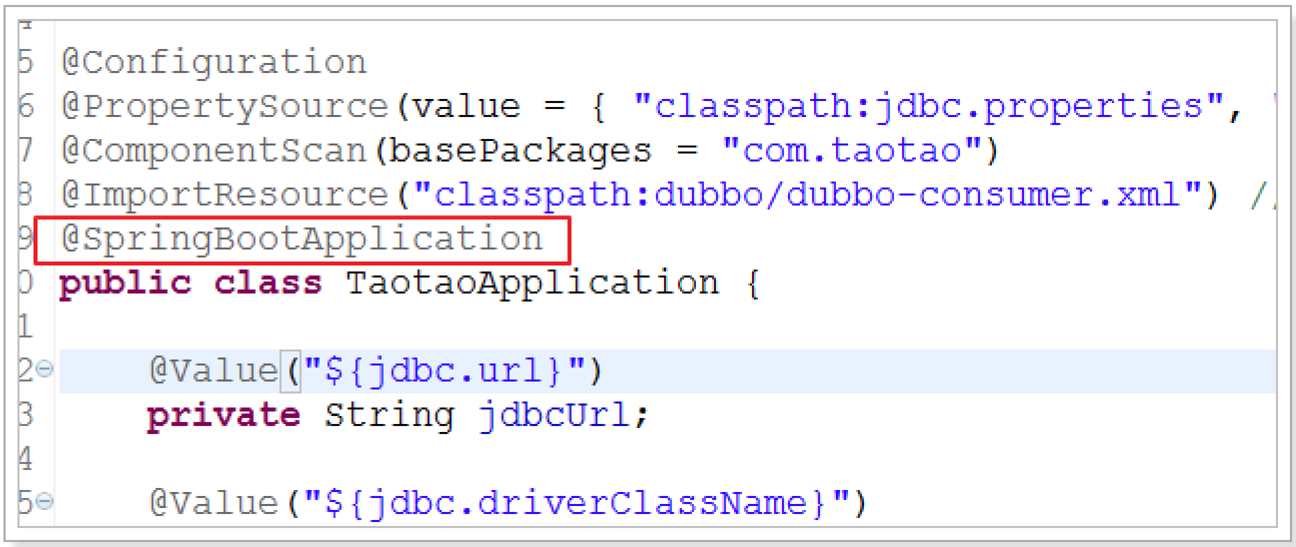
```
public class Main {

    public static void main(String[] args) {
        SpringApplication.run(TaotaoApplication.class, arg
    }

}
```

6.4.1. 启动错误 1

关键错误（丢失了 web 容器的工厂，也就是说我们并没有把它作为一个 web 应用来启动）：

```
org.springframework.context.ApplicationContextException:
Unable to start embedded container; nested exception is
org.springframework.context.ApplicationContextException:
Unable to start EmbeddedWebApplicationContext due to
missing EmbeddedServletContainerFactory bean.
```

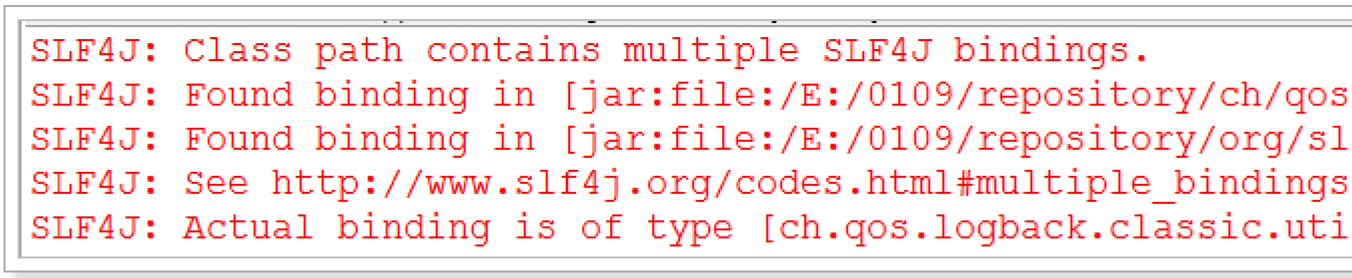
解决：



```
@Configuration
@PropertySource(value = { "classpath:jdbc.properties",
@ComponentScan(basePackages = "com.taotao")
@ImportResource("classpath:dubbo/dubbo-consumer.xml") /
@SpringBootApplication
public class TaotaoApplication {

    @Value("${jdbc.url}")
    private String jdbcUrl;

    @Value("${jdbc.driverClassName}")
```

让 Spring Boot 来自动选择并且完成 web 的相关加载工作。

6.4.2. Slf4j 日志警告

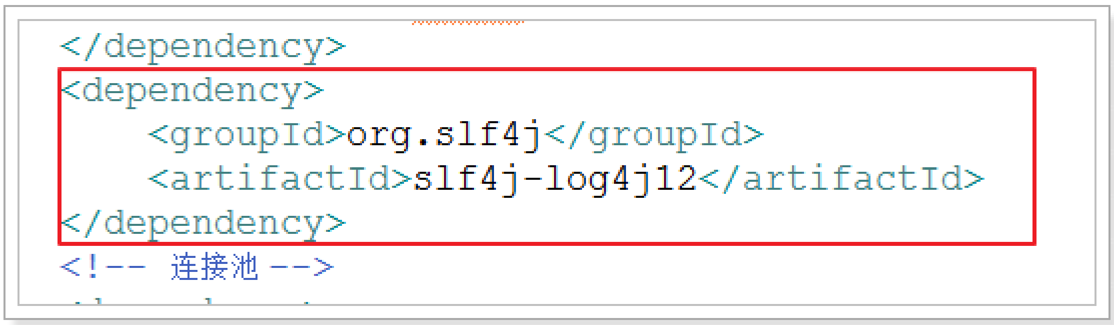


```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/E:/0109/repository/ch/qos
SLF4J: Found binding in [jar:file:/E:/0109/repository/org/sl
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings
SLF4J: Actual binding is of type [ch.qos.logback.classic.uti
```

提示我们当前的项目中 slf4j 引入了 2 个，导致了 jar 冲突。

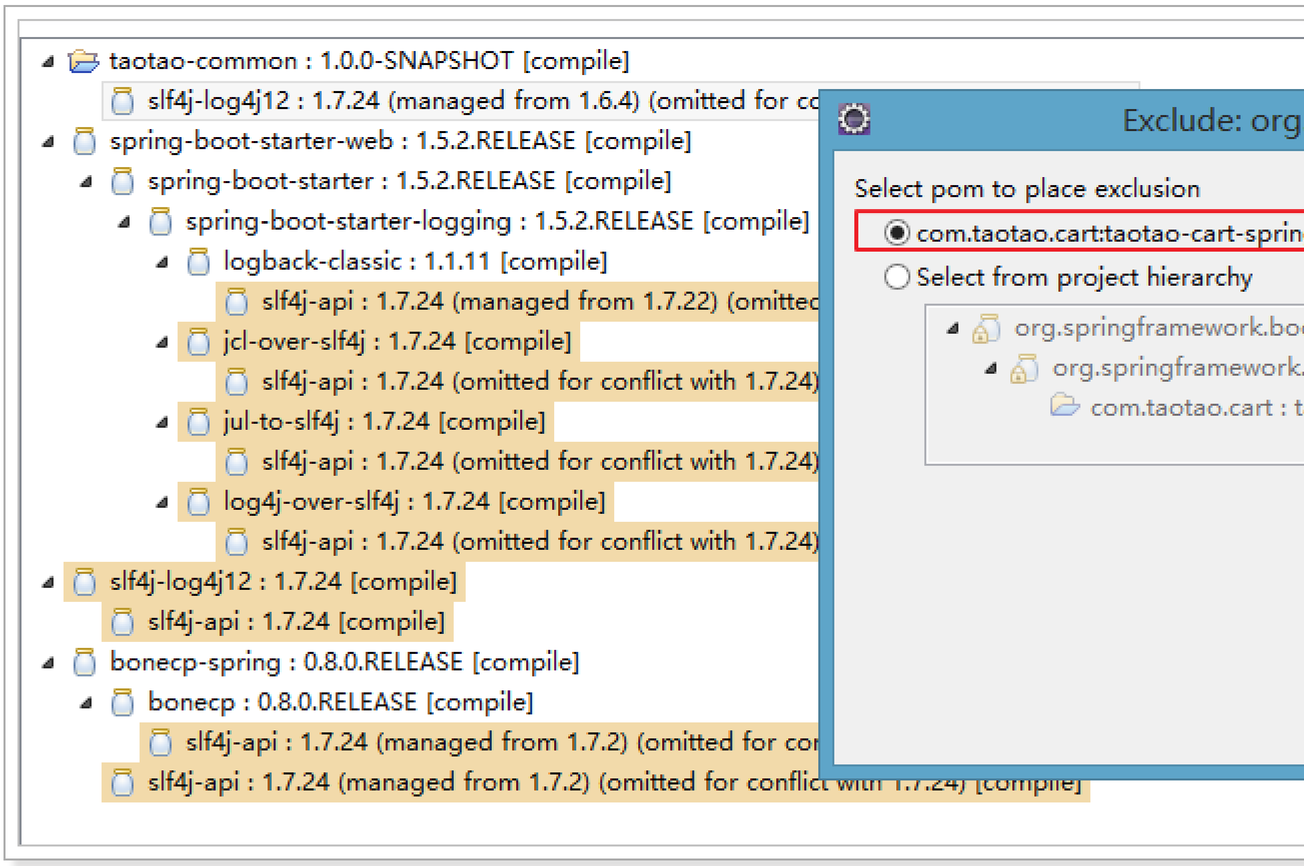
解决：

1、 删除自己引入到 slf4j 的依赖



```
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
</dependency>
<!-- 连接池 -->
```

2、 将 taotao-common 中传递的依赖排除掉



```
taotao-common : 1.0.0-SNAPSHOT [compile]
    slf4j-log4j12 : 1.7.24 (managed from 1.6.4) (omitted for co
spring-boot-starter-web : 1.5.2.RELEASE [compile]
    spring-boot-starter : 1.5.2.RELEASE [compile]
        spring-boot-starter-logging : 1.5.2.RELEASE [compile]
            logback-classic : 1.1.11 [compile]
                slf4j-api : 1.7.24 (managed from 1.7.22) (omitted
            jcl-over-slf4j : 1.7.24 [compile]
                slf4j-api : 1.7.24 (omitted for conflict with 1.7.24)
            jul-to-slf4j : 1.7.24 [compile]
                slf4j-api : 1.7.24 (omitted for conflict with 1.7.24)
            log4j-over-slf4j : 1.7.24 [compile]
                slf4j-api : 1.7.24 (omitted for conflict with 1.7.24)
slf4j-log4j12 : 1.7.24 [compile]
    slf4j-api : 1.7.24 [compile]
bonecp-spring : 0.8.0.RELEASE [compile]
    bonecp : 0.8.0.RELEASE [compile]
        slf4j-api : 1.7.24 (managed from 1.7.2) (omitted for co
    slf4j-api : 1.7.24 (managed from 1.7.2) (omitted for conflict with 1.7.24) [compile]
```

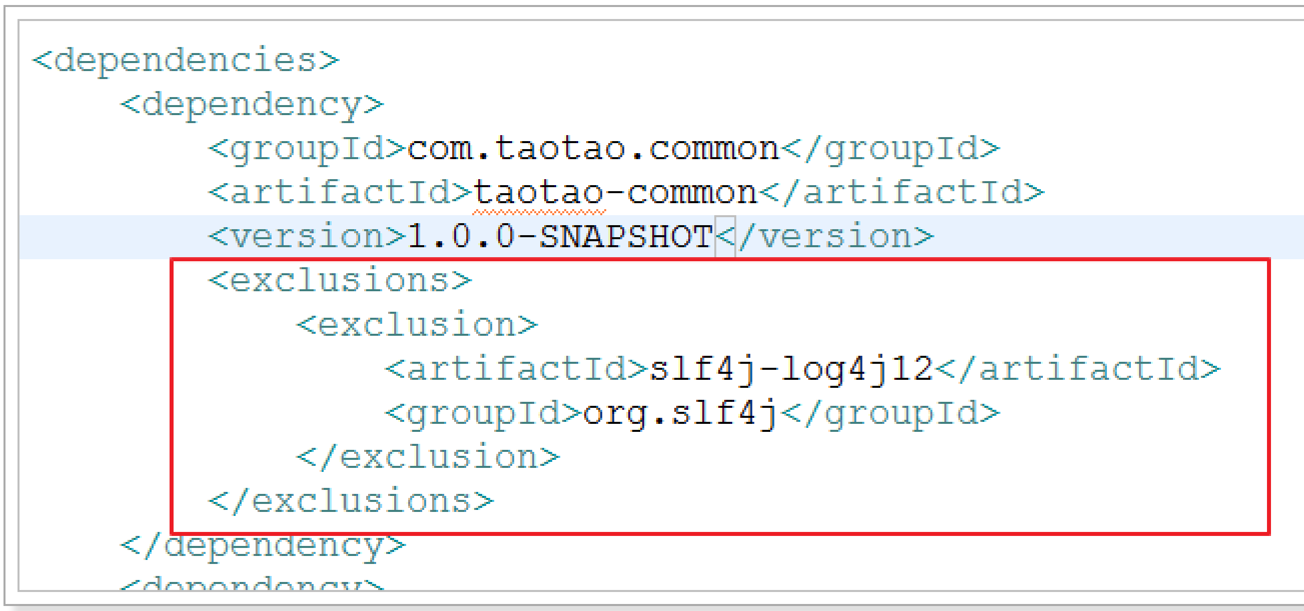
Exclude: org

Select pom to place exclusion

[● com.taotao.cart:taotao-cart-sprin]

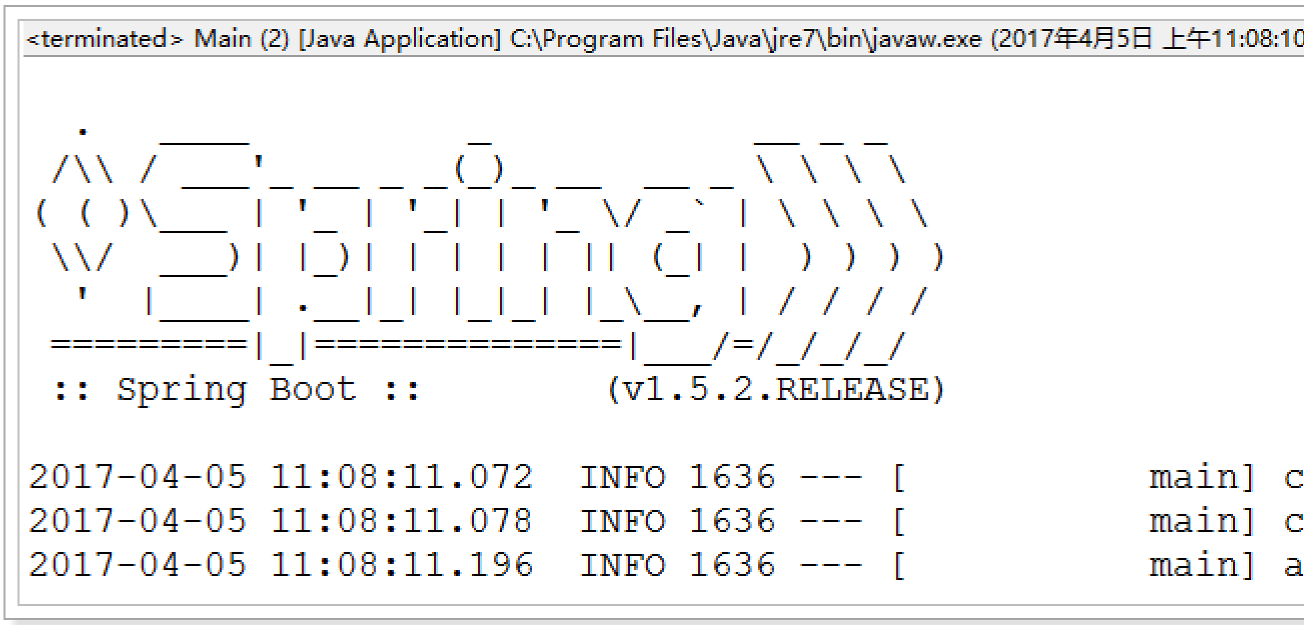
○ Select from project hierarchy

org.springframework.bo
org.springframework.
com.taotao.cart : t



```
<dependencies>
    <dependency>
        <groupId>com.taotao.common</groupId>
        <artifactId>taotao-common</artifactId>
        <version>1.0.0-SNAPSHOT</version>
        <exclusions>
            <exclusion>
                <artifactId>slf4j-log4j12</artifactId>
                <groupId>org.slf4j</groupId>
            </exclusion>
        </exclusions>
    </dependency>
```

再次启动，发现警告没了：



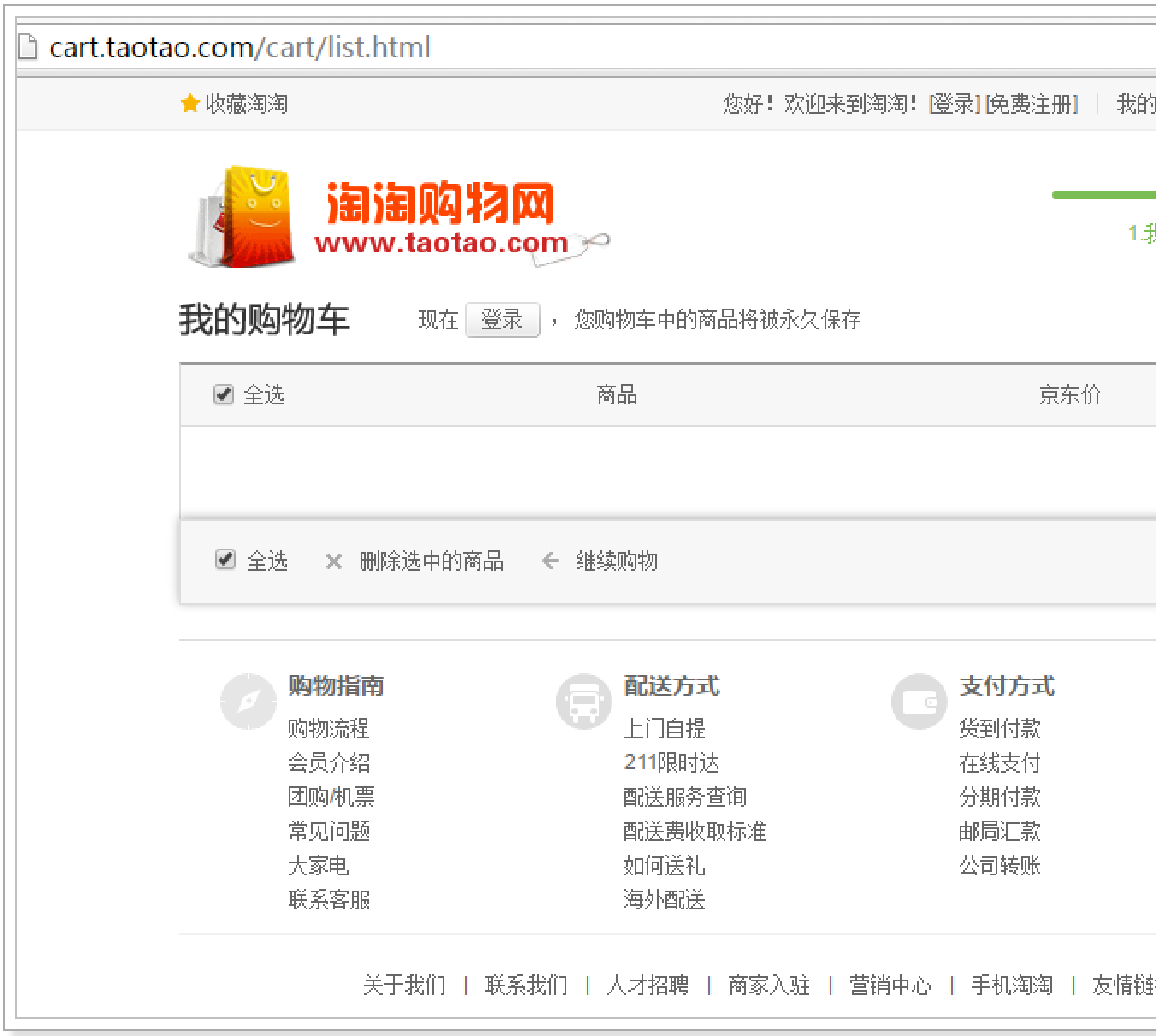
```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (2017年4月5日 上午11:08:10

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.2.RELEASE)

2017-04-05 11:08:11.072  INFO 1636 --- [           main] c
2017-04-05 11:08:11.078  INFO 1636 --- [           main] c
2017-04-05 11:08:11.196  INFO 1636 --- [           main] a
```

6.4.3. 解决 jsp 访问 404 的问题

由于 Spring boot 使用的内嵌的 tomcat，而内嵌的 tamcat 是不支持 jsp 页面的，所有需要导入额外的包才能解决。

```
<dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
            <scope>provided</scope>
        </dependency>
```

重新启动进行测试：


cart.taotao.com/cart/list.html

收藏淘淘　　您好！欢迎来到淘淘！[登录] [免费注册]　|　我的

淘淘购物网 www.taotao.com

我的购物车　现在 登录 ，您购物车中的商品将被永久保存

全选	商品	京东价

全选　× 删除选中的商品　← 继续购物

购物指南：购物流程、会员介绍、团购/机票、常见问题、大家电、联系客服

配送方式：上门自提、211限时达、配送服务查询、配送费收取标准、如何送礼、海外配送

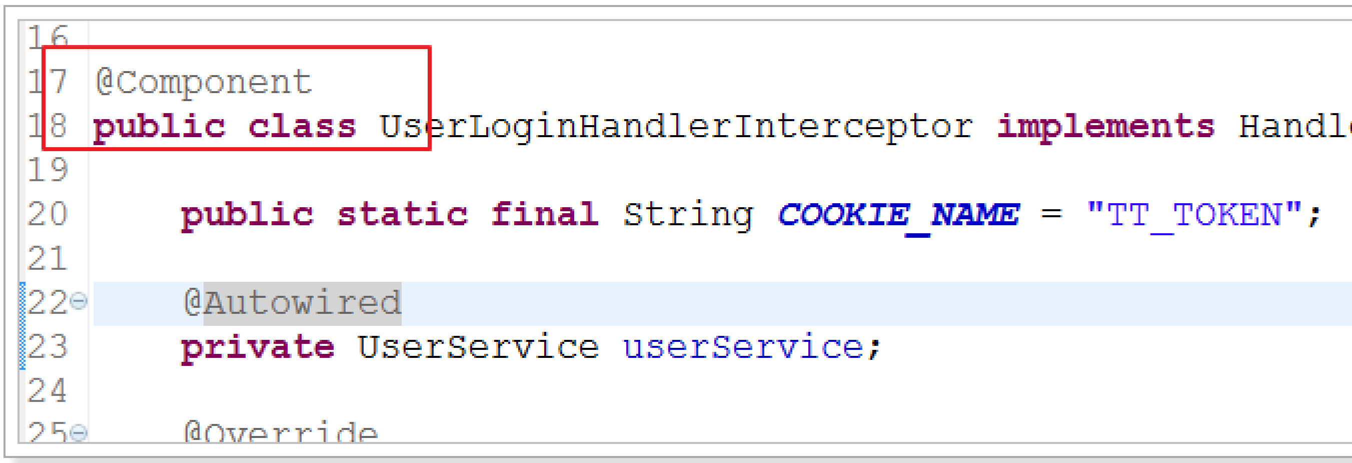
支付方式：货到付款、在线支付、分期付款、邮局汇款、公司转账

关于我们 | 联系我们 | 人才招聘 | 商家入驻 | 营销中心 | 手机淘淘 | 友情链

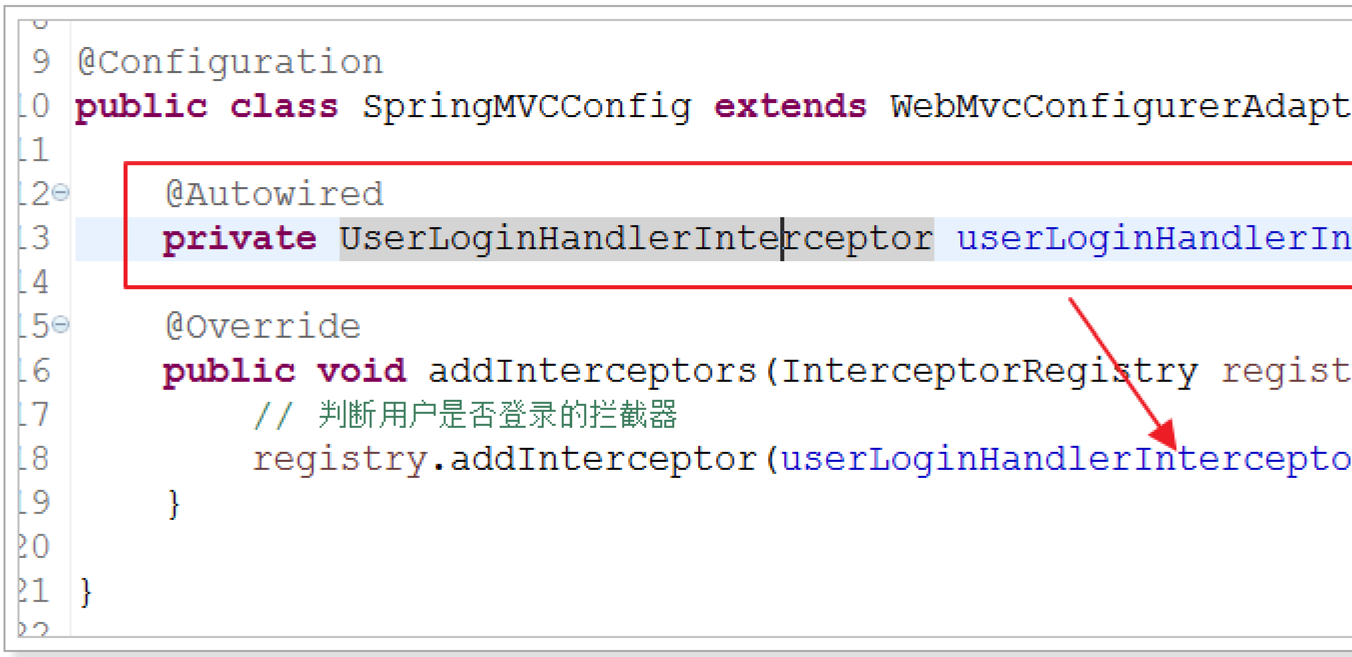
6.4.4. 拦截器中的 UserService 空指针异常

分析：由于添加拦截器时，直接对 UserLoginHandlerInterceptor 进行 new 操作，导致 UserService 无法注入，所以有空指针异常。

解决：



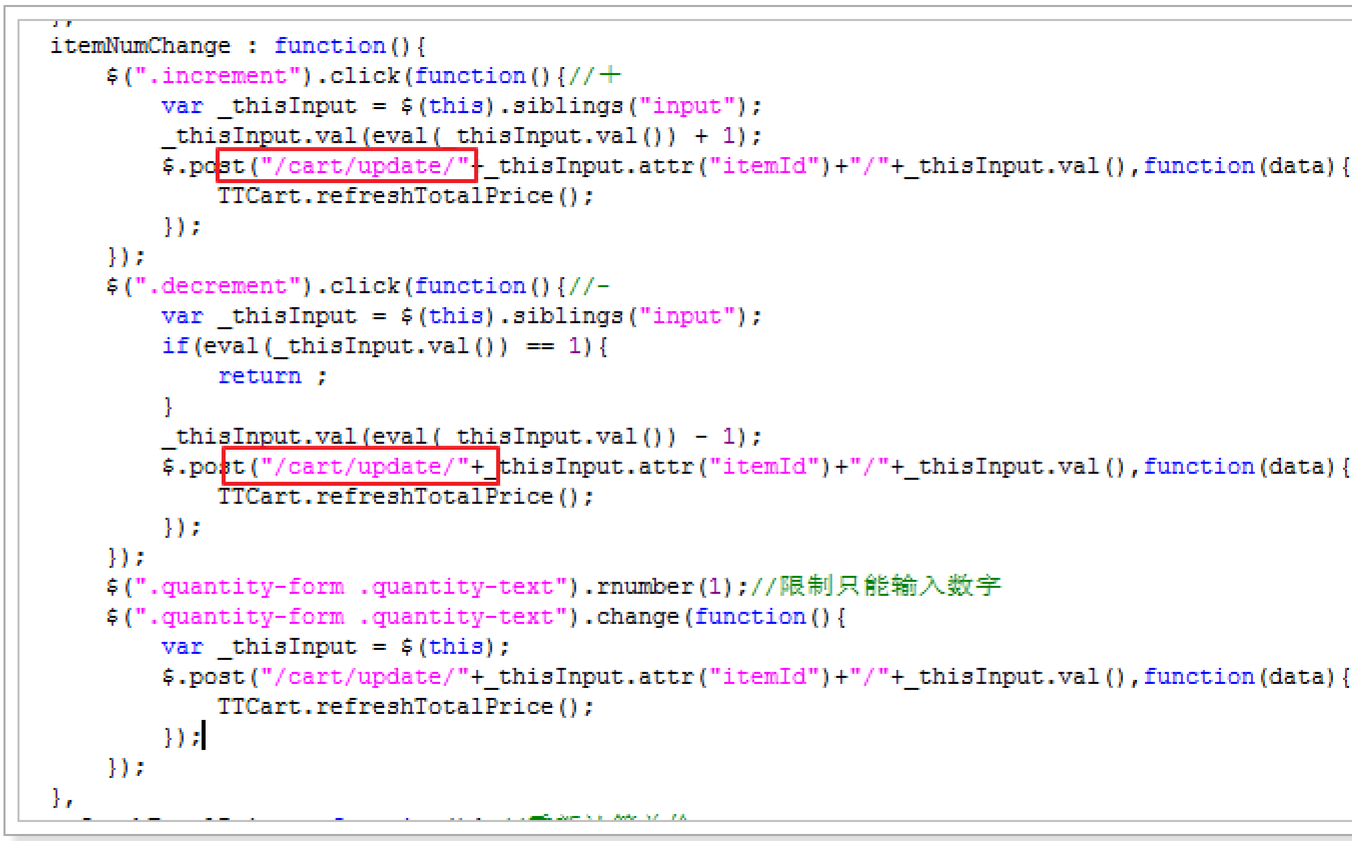
```
@Component
public class UserLoginHandlerInterceptor implements Handl

    public static final String COOKIE_NAME = "TT_TOKEN";

    @Autowired
    private UserService userService;

    @Override
```



```
@Configuration
public class SpringMVCConfig extends WebMvcConfigurerAdapt

    @Autowired
    private UserLoginHandlerInterceptor userLoginHandlerIn

    @Override
    public void addInterceptors(InterceptorRegistry regist
        // 判断用户是否登录的拦截器
        registry.addInterceptor(userLoginHandlerIntercepto
    }

}
```

6.4.5. 路径问题

现在我们进入 Servlet 的路径为"/"，访问*.html 页面没问题，但是，访问 /service/* 就会有问题，所以需要改一下 js，将原有的/service/ 改为 /



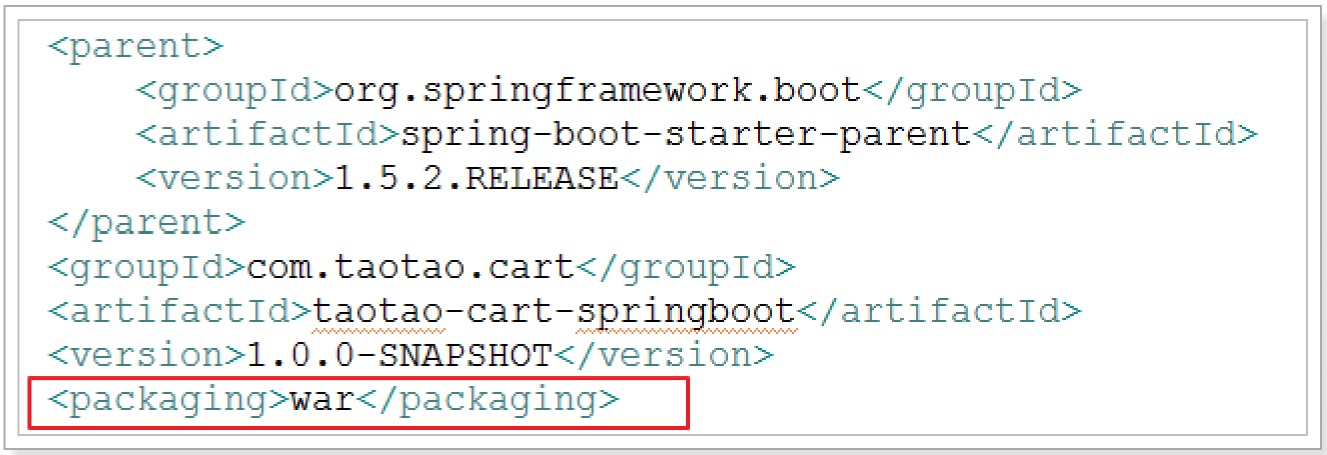
```
itemNumChange : function(){
    $(".increment").click(function(){//+
        var _thisInput = $(this).siblings("input");
        _thisInput.val(eval(_thisInput.val()) + 1);
        $.post("/cart/update/"+_thisInput.attr("itemId")+"/"+_thisInput.val(),function(data){
            TTCart.refreshTotalPrice();
        });
    });
    $(".decrement").click(function(){//-
        var _thisInput = $(this).siblings("input");
        if(eval(_thisInput.val()) == 1){
            return ;
        }
        _thisInput.val(eval(_thisInput.val()) - 1);
        $.post("/cart/update/"+_thisInput.attr("itemId")+"/"+_thisInput.val(),function(data){
            TTCart.refreshTotalPrice();
        });
    });
    $(".quantity-form .quantity-text").rnumber(1);//限制只能输入数字
    $(".quantity-form .quantity-text").change(function(){
        var _thisInput = $(this);
        $.post("/cart/update/"+_thisInput.attr("itemId")+"/"+_thisInput.val(),function(data){
            TTCart.refreshTotalPrice();
        });
    });
},
```

测试，功能一切 ok。

7. 发布到独立的 tomcat 中运行

在开发阶段我们推荐使用内嵌的 tomcat 进行开发，因为这样会方便很多，但是到生成环境，我希望在独立的 tomcat 容器中运行，因为我们需要对 tomcat 做额外的优化，这时我们需要将工程打包成 war 包发进行发布。

7.1. 工程的打包方式为 war



```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.2.RELEASE</version>
</parent>
<groupId>com.taotao.cart</groupId>
<artifactId>taotao-cart-springboot</artifactId>
<version>1.0.0-SNAPSHOT</version>
<packaging>war</packaging>
```

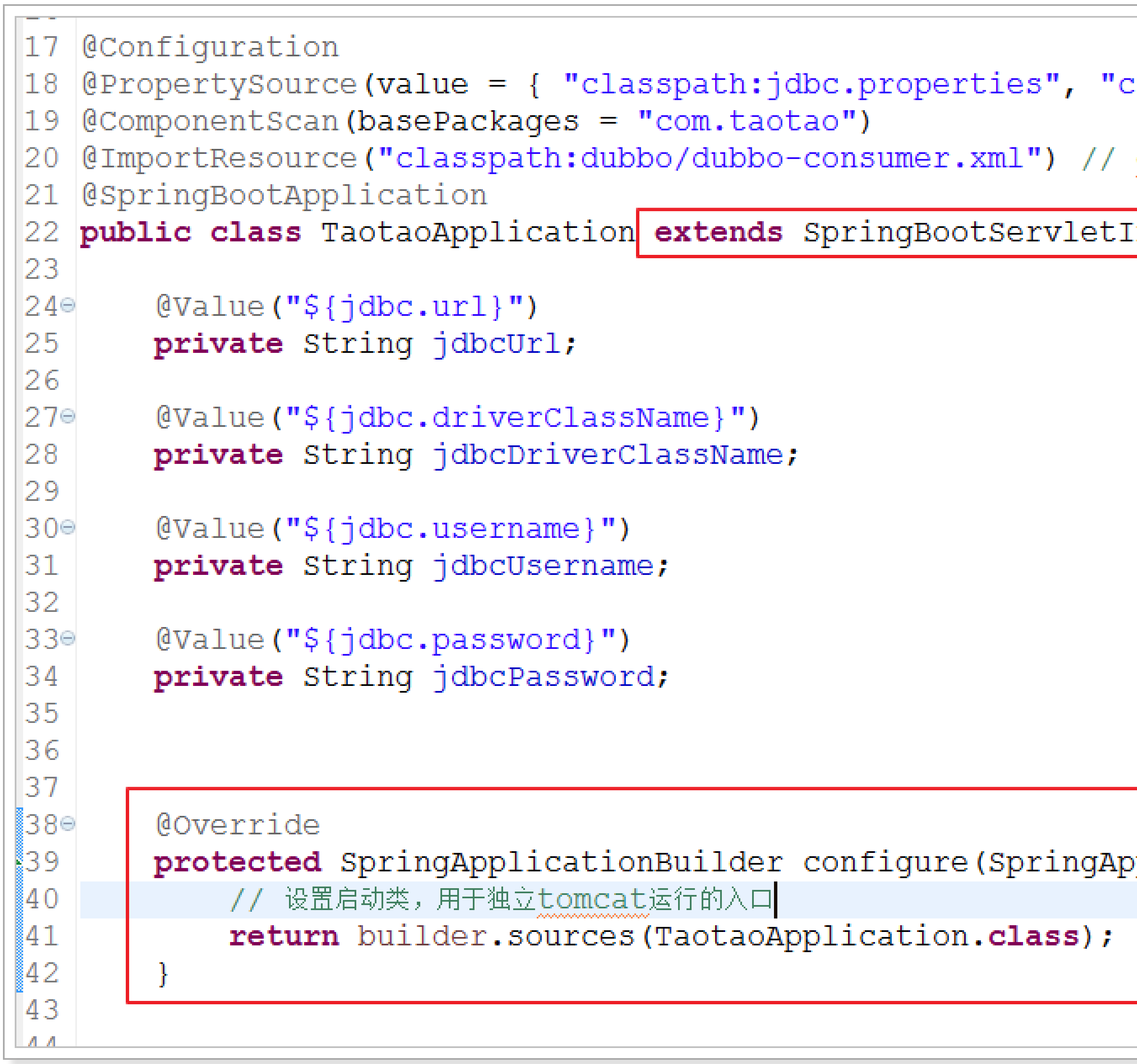
7.2. 将 spring-boot-starter-tomcat 的范围设置为 provided

设置为 provided 是在打包时会将该包排除，因为要放到独立的 tomcat 中运行，是不需要的。

```xml
<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-
tomcat</artifactId>
		<scope>provided</scope>
	</dependency>
```

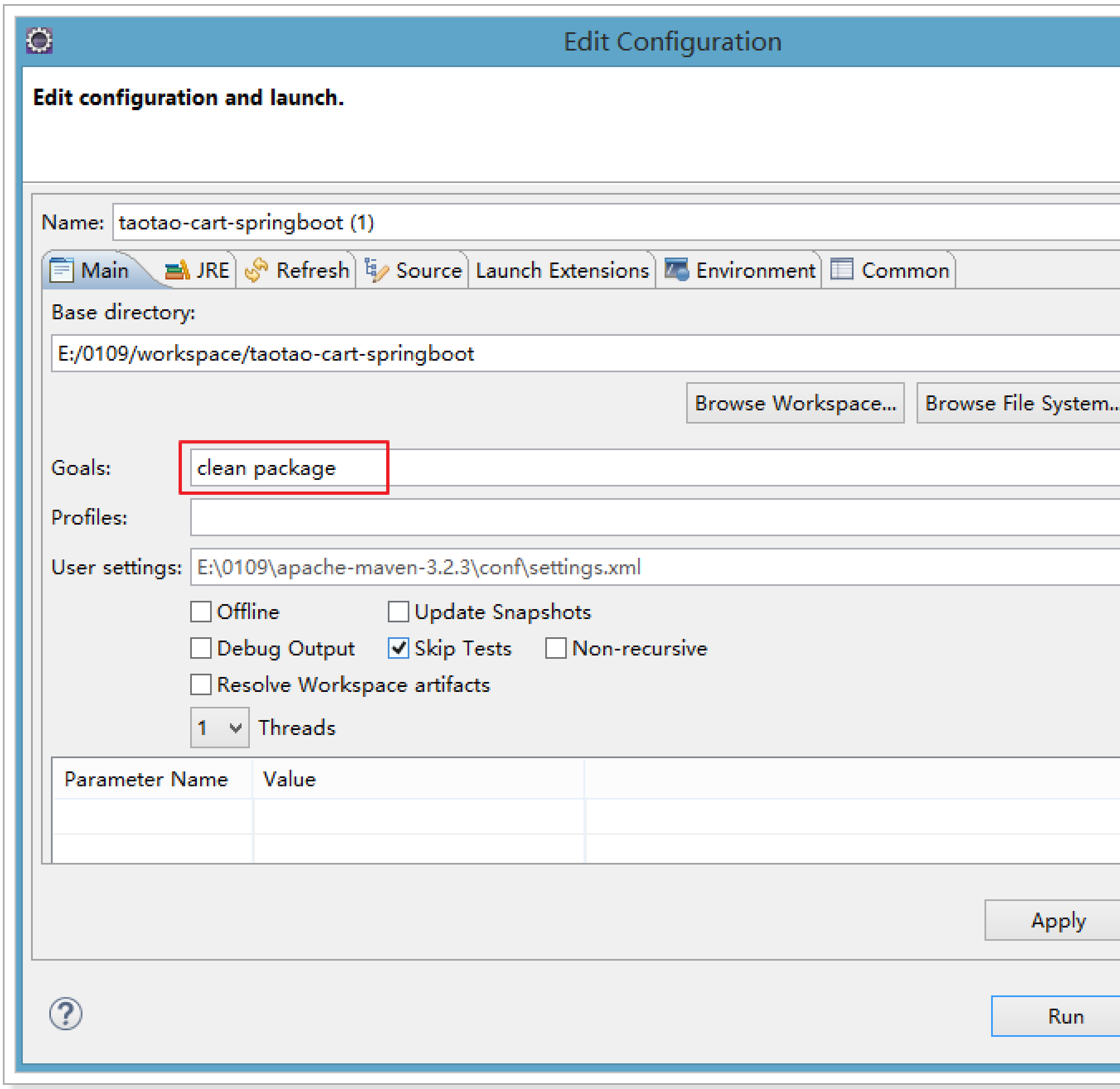
7.3. 修改代码，设置启动配置

需要集成 SpringBootServletInitializer，然后重写 configure，将 Spring Boot 的入口类设置进去。



```
@Configuration
@PropertySource(value = { "classpath:jdbc.properties", "c
@ComponentScan(basePackages = "com.taotao")
@ImportResource("classpath:dubbo/dubbo-consumer.xml") //
@SpringBootApplication
public class TaotaoApplication extends SpringBootServletI

    @Value("${jdbc.url}")
    private String jdbcUrl;

    @Value("${jdbc.driverClassName}")
    private String jdbcDriverClassName;

    @Value("${jdbc.username}")
    private String jdbcUsername;

    @Value("${jdbc.password}")
    private String jdbcPassword;



    @Override
    protected SpringApplicationBuilder configure(SpringAp
        // 设置启动类，用于独立tomcat运行的入口
        return builder.sources(TaotaoApplication.class);
    }

```

7.4. 打 war 包



Edit Configuration

Edit configuration and launch.

Name: taotao-cart-springboot (1)

Base directory:

E:/0109/workspace/taotao-cart-springboot

Goals: clean package

Profiles:

User settings: E:\0109\apache-maven-3.2.3\conf\settings.xml

☐ Offline ☐ Update Snapshots

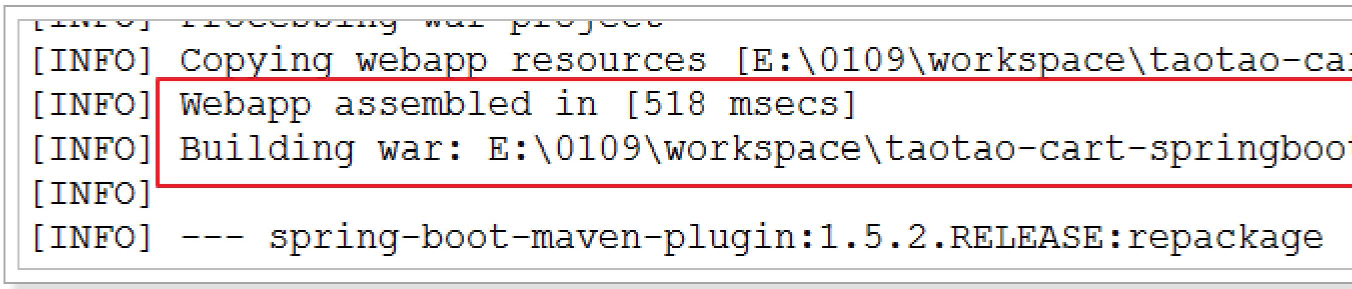
☐ Debug Output ☑ Skip Tests ☐ Non-recursive

☐ Resolve Workspace artifacts

1 Threads

Parameter Name	Value

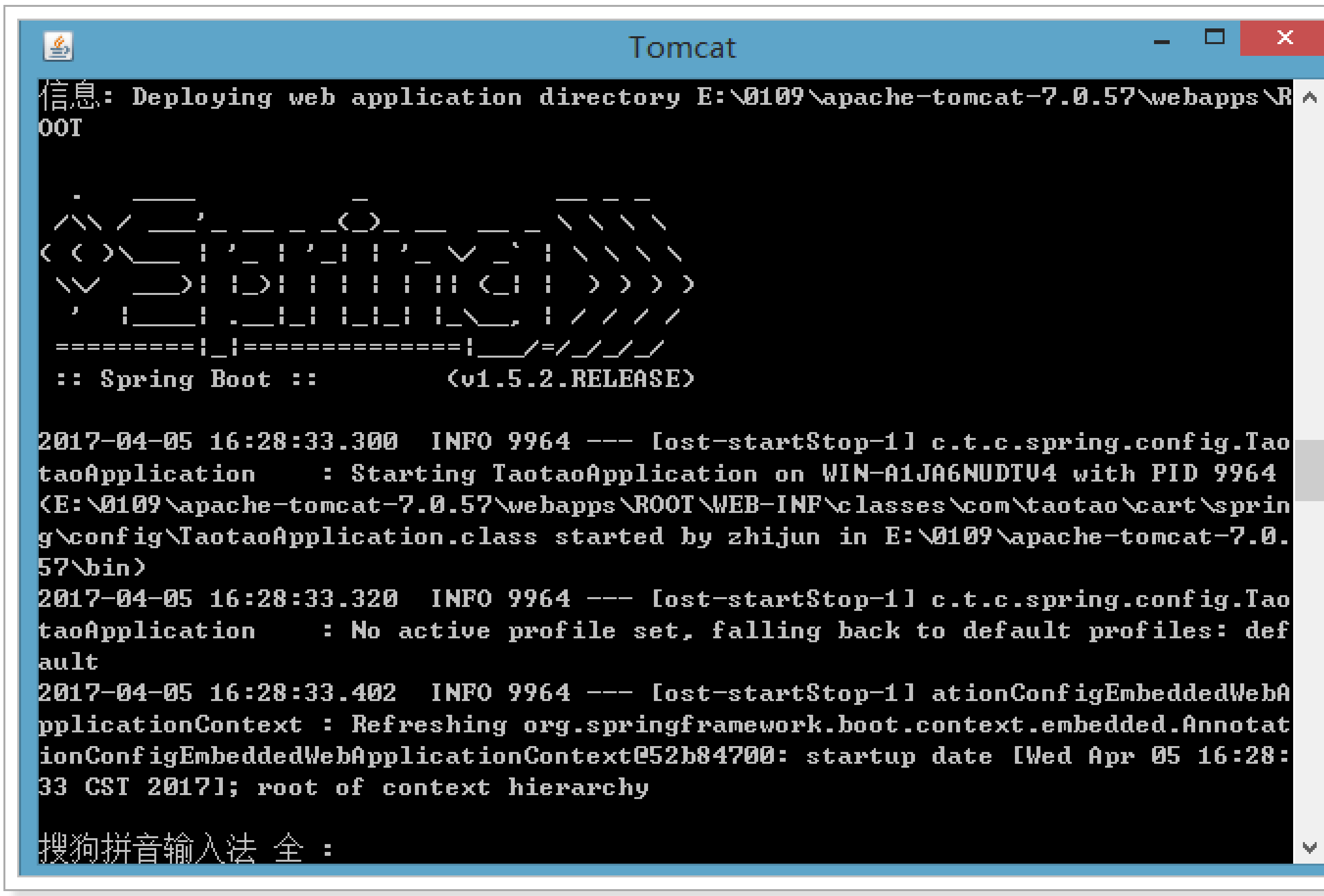
打包成功：



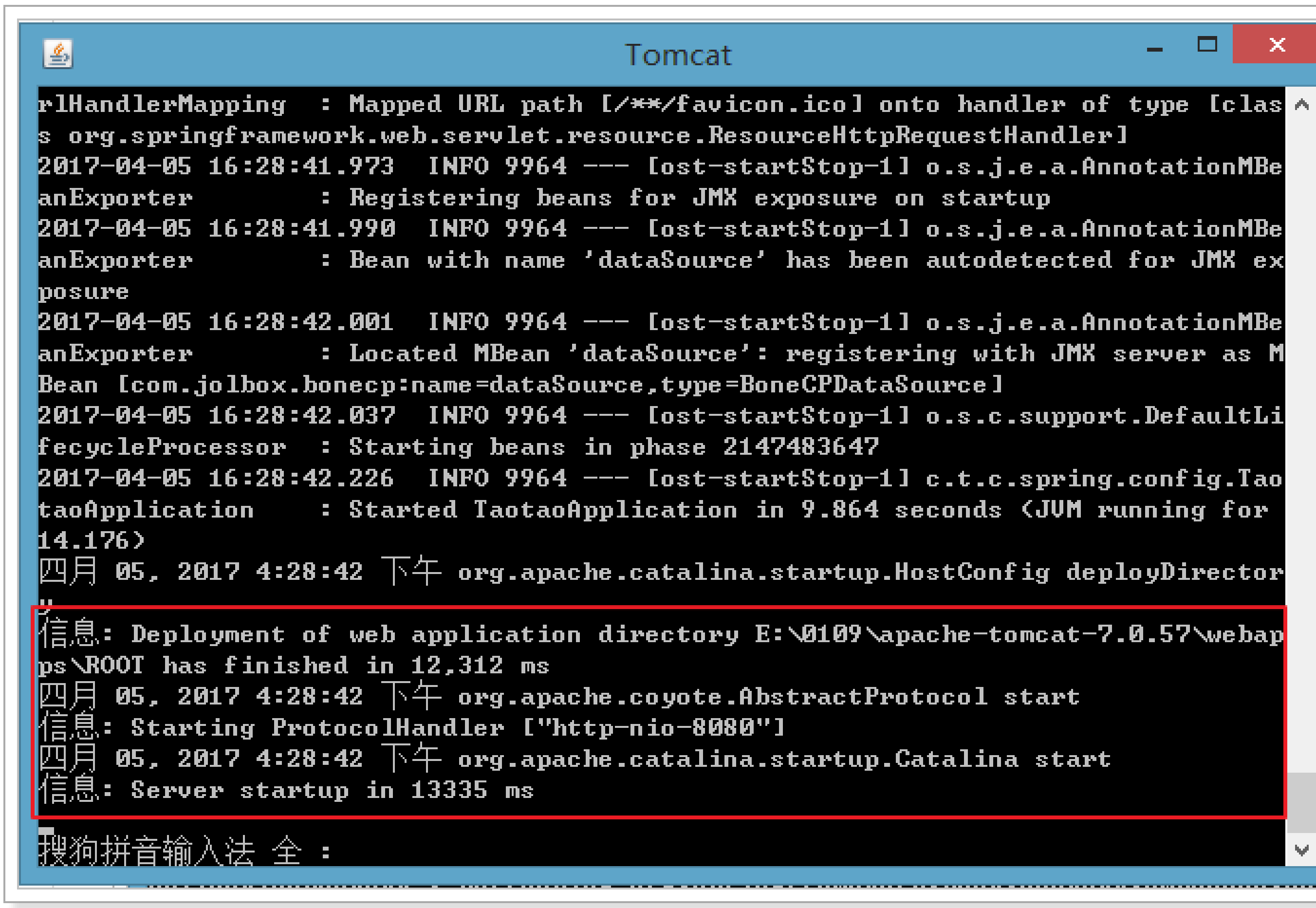
```
[INFO] Copying webapp resources [E:\0109\workspace\taotao-ca
[INFO] Webapp assembled in [518 msecs]
[INFO] Building war: E:\0109\workspace\taotao-cart-springboo
[INFO]
[INFO] --- spring-boot-maven-plugin:1.5.2.RELEASE:repackage
```

7.5. 部署到 tomcat

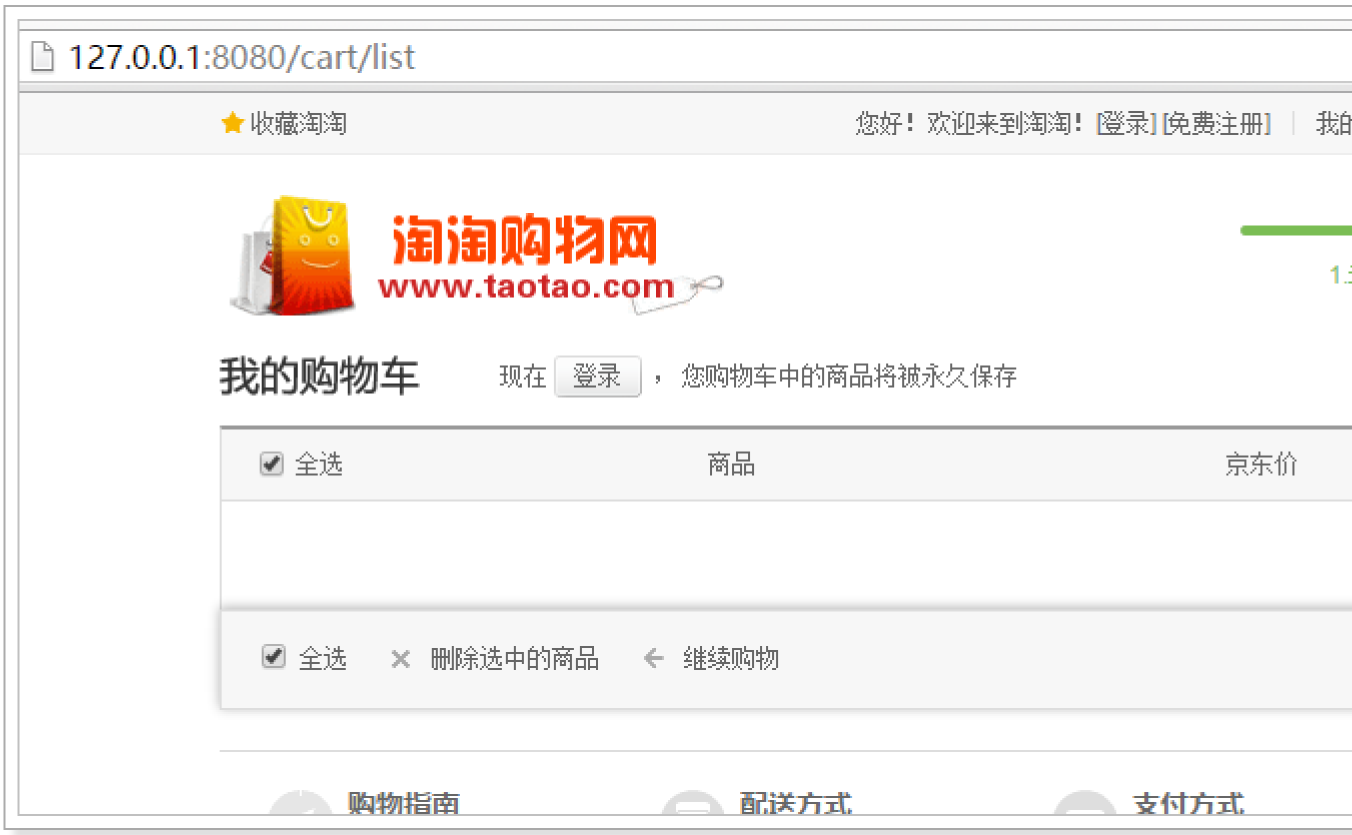
解压 apache-tomcat-7.0.57.tar.gz，将 war 包解压到 webapps 下的 ROOT 目录中，启动：



```
信息: Deploying web application directory E:\0109\apache-tomcat-7.0.57\webapps\R
OOT

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.2.RELEASE)

2017-04-05 16:28:33.300  INFO 9964 --- [ost-startStop-1] c.t.c.spring.config.Tao
taoApplication     : Starting TaotaoApplication on WIN-A1JA6NUDTV4 with PID 9964
(E:\0109\apache-tomcat-7.0.57\webapps\ROOT\WEB-INF\classes\com\taotao\cart\sprin
g\config\TaotaoApplication.class started by zhijun in E:\0109\apache-tomcat-7.0.
57\bin)
2017-04-05 16:28:33.320  INFO 9964 --- [ost-startStop-1] c.t.c.spring.config.Tao
taoApplication     : No active profile set, falling back to default profiles: def
ault
2017-04-05 16:28:33.402  INFO 9964 --- [ost-startStop-1] ationConfigEmbeddedWebA
pplicationContext : Refreshing org.springframework.boot.context.embedded.Annotat
ionConfigEmbeddedWebApplicationContext@52b84700: startup date [Wed Apr 05 16:28:
33 CST 2017]; root of context hierarchy

搜狗拼音输入法 全 :
```



```
rlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [clas
s org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2017-04-05 16:28:41.973  INFO 9964 --- [ost-startStop-1] o.s.j.e.a.AnnotationMBe
anExporter        : Registering beans for JMX exposure on startup
2017-04-05 16:28:41.990  INFO 9964 --- [ost-startStop-1] o.s.j.e.a.AnnotationMBe
anExporter        : Bean with name 'dataSource' has been autodetected for JMX ex
posure
2017-04-05 16:28:42.001  INFO 9964 --- [ost-startStop-1] o.s.j.e.a.AnnotationMBe
anExporter        : Located MBean 'dataSource': registering with JMX server as M
Bean [com.jolbox.bonecp:name=dataSource,type=BoneCPDataSource]
2017-04-05 16:28:42.037  INFO 9964 --- [ost-startStop-1] o.s.c.support.DefaultLi
fecycleProcessor  : Starting beans in phase 2147483647
2017-04-05 16:28:42.226  INFO 9964 --- [ost-startStop-1] c.t.c.spring.config.Tao
taoApplication    : Started TaotaoApplication in 9.864 seconds (JVM running for
14.176)
四月 05, 2017 4:28:42 下午 org.apache.catalina.startup.HostConfig deployDirector
y
信息: Deployment of web application directory E:\0109\apache-tomcat-7.0.57\webap
ps\ROOT has finished in 12,312 ms
四月 05, 2017 4:28:42 下午 org.apache.coyote.AbstractProtocol start
信息: Starting ProtocolHandler ["http-nio-8080"]
四月 05, 2017 4:28:42 下午 org.apache.catalina.startup.Catalina start
信息: Server startup in 13335 ms

搜狗拼音输入法 全 :
```



完美！