**Spring Boot使用@SpringBootApplication注解**

很多Spring Boot开发者总是使用 @Configuration ， @EnableAutoConfiguration 和 @ComponentScan 注解他们的main类。由于这些注解被如此频繁地一块使用，Spring Boot提供一个方便的 @SpringBootApplication 选择。该 @SpringBootApplication 注解等价于以默认属性使用 @Configuration ，@EnableAutoConfiguration 和 @ComponentScan 。

**pom.xml中相关的配置详解：**

1、Spring Boot是一个特殊的starter，它提供了有用的Maven默认设置。它也提供了一个 dependency-management 节点，这样对于”blessed“依赖你可以省略version标记。

pom.xml Maven用户可以继承 spring-boot-starter-parent 项目来获取合适的默认设置。该父项目提供以下特性：

①、默认编译级别为Java 1.6 ②、源码编码为UTF-8 ③、一个依赖管理节点，允许你省略普通依赖的<version>标签，继承自spring-boot-dependencies POM。 ④、合适的资源过滤 ⑤、合适的插件配置（exec插件，surefire，Git commit ID，shade）针对application.properties和application.xml的资源过 滤。

想配置你的项目继承spring-boot-starter-parent只需要简单地设置parent为：

<!-- Inherit defaults from Spring Boot -->

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>1.3.0.BUILD-SNAPSHOT</version>

</parent>

注：你应该只需要在该依赖上指定Spring Boot版本。如果导入其他的starters，你可以放心的省略版本号。

**注：如果原本项目中有自己的parent,Spring Boot还提供了另外一种方式，不使用spring-boot-starter-parent，通过使用一个scope=import的依赖，仍能获取到依赖管理的好处。**

<dependencyManagement>

<dependencies>

<dependency>

<!-- Import dependency management from Spring Boot --> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-dependencies</artifactId> <version>1.3.0.BUILD-SNAPSHOT</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

2、Spring Boot的启动器starter详解：

Spring Boot的应用启动器共有44种：

spring-boot-starter 核心Spring Boot starter，包括自动配置支持，日志和YAML；

spring-boot-starteractuator 生产准备的特性，用于帮你监控和管理应用；

spring-boot-starteramqp 对"高级消息队列协议"的支持，通过 spring-rabbit 实现；

spring-boot-starteraop 对面向切面编程的支持，包括 spring-aop 和AspectJ；

spring-boot-starter-artemis 通过Apache Artemis支持JMS的API（Java Message Service API）；

spring-boot-starterbatch 对Spring Batch的支持，包括HSQLDB数据库；

spring-boot-starter-cache 支持Spring的Cache抽象；

spring-boot-startercloud-connectors 对Spring Cloud Connectors的支持，简化在云平台下（ 例如，

Cloud Foundry 和Heroku） 服务的连接；

spring-boot-starterdata-elasticsearch 对Elasticsearch搜索和分析引擎的支持，包括 spring-dataelasticsearch；

spring-boot-starterdata-gemfire 对GemFire分布式数据存储的支持，包括 spring-data-gemfire；

spring-boot-starterdata-jpa 对"Java持久化API"的支持，包括 spring-data-jpa ， springorm 和Hibernate。

spring-boot-starterdata-mongodb 对MongoDB NOSQL数据库的支持，包括 spring-data-mongodb；

spring-boot-starterdata-rest 对通过REST暴露Spring Data仓库的支持，通过 spring-datarest-webmvc 实现；

spring-boot-starter-data-solr 对Apache Solr搜索平台的支持，包括 spring-data-solr；

spring-boot-starterfreemarker 对FreeMarker模板引擎的支持；

spring-boot-startergroovy-templates 对Groovy模板引擎的支持；

spring-boot-starterhateoas 对基于HATEOAS的RESTful服务的支持，通过 springhateoas 实现；

spring-boot-starterhornetq 对"Java消息服务API"的支持，通过HornetQ实现；

spring-boot-starterintegration 对普通 spring-integration 模块的支持；

spring-boot-starterjdbc 对JDBC数据库的支持；

spring-boot-starterjersey 对Jersey RESTful Web服务框架的支持；

spring-boot-starter-jtaatomikos 对JTA分布式事务的支持，通过Atomikos实现；

spring-boot-starter-jtabitronix 对JTA分布式事务的支持，通过Bitronix实现；

spring-boot-startermail 对 javax.mail 的支持；

spring-boot-startermobile 对 spring-mobile 的支持；

spring-boot-startermustache 对Mustache模板引擎的支持；

spring-boot-starterredis 对REDIS键值数据存储的支持，包括 spring-redis；

spring-boot-startersecurity 对 spring-security 的支持；

spring-boot-startersocial-facebook 对 spring-social-facebook 的支持；

spring-boot-startersocial-linkedin 对 spring-social-linkedin 的支持；

spring-boot-startersocial-twitter 对 spring-social-twitter 的支持；

spring-boot-startertest 对常用测试依赖的支持，包括JUnit, Hamcrest和Mockito，还

有 spring-test 模块；

spring-boot-starterthymeleaf 对Thymeleaf模板引擎的支持，包括和Spring的集成；

spring-boot-startervelocity 对Velocity模板引擎的支持；

spring-boot-starterweb 对全栈web开发的支持，包括Tomcat和 spring-webmvc；

spring-boot-starterwebsocket 对WebSocket开发的支持；

spring-boot-starter-ws 对Spring Web服务的支持；

spring-boot-starter-actuator 添加生产准备特性，比如指标和监控；

spring-boot-starter-remote-shell 添加远程 ssh shell支持；

spring-boot-starter-jetty 导入Jetty HTTP引擎（ 作为Tomcat的替代）；

spring-boot-starter-log4j 对Log4J日志系统的支持；

spring-boot-starter-logging 导入Spring Boot的默认日志系统（ Logback）；

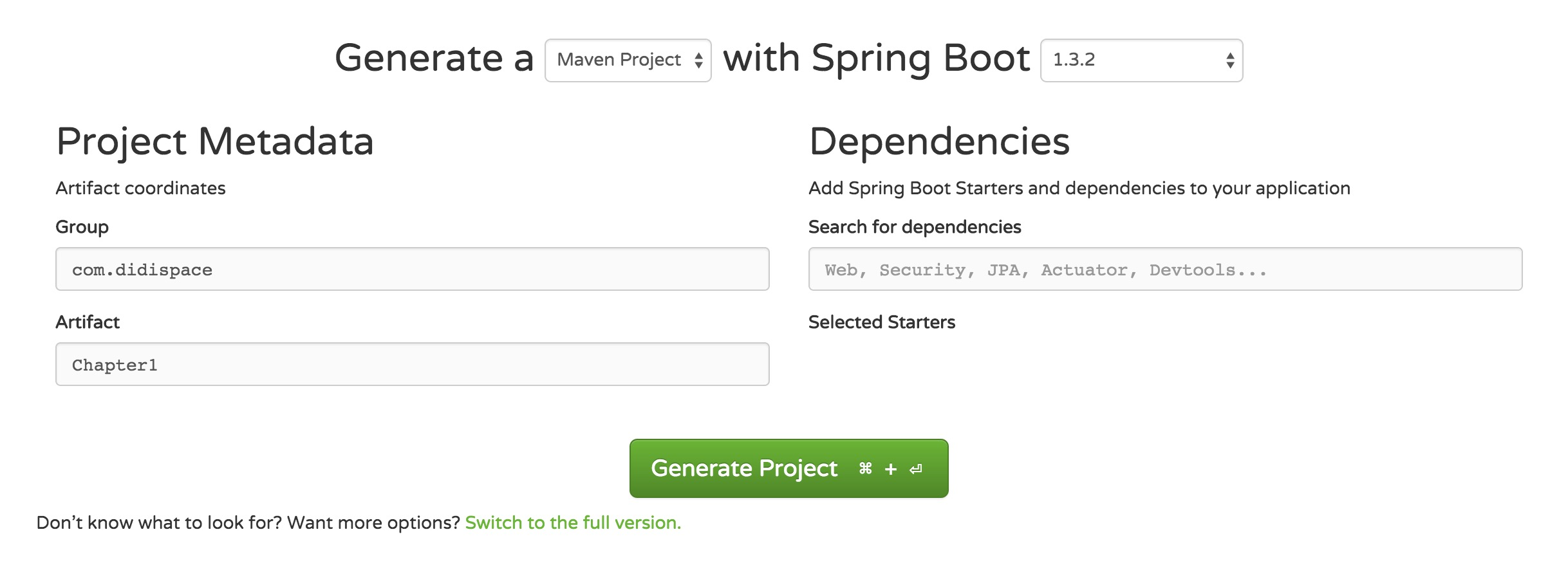
spring-boot-starter-tomcat 导入Spring Boot的默认HTTP引擎（ Tomcat）；

spring-boot-starter-undertow 导入Undertow HTTP引擎（ 作为Tomcat的替代）。

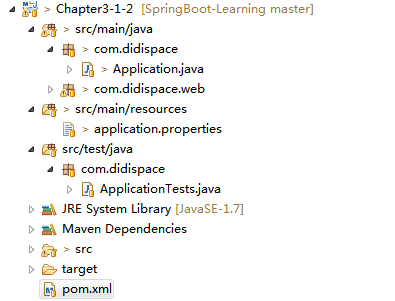
运用Spring Boot构建一个简单的web项目：

①、使用Maven构建项目

1. 通过SPRING INITIALIZR工具产生基础项目
   1. 访问：http://start.spring.io/
   2. 选择构建工具Maven Project、Spring Boot版本1.3.2以及一些工程基本信息，可参考下图所示：



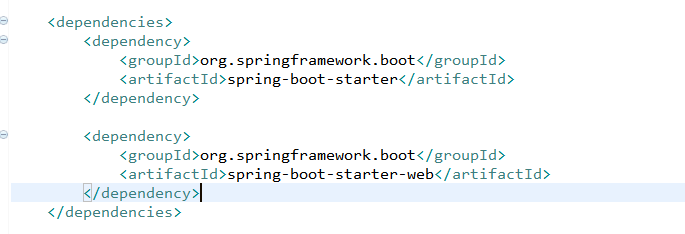
②、项目结构解析：



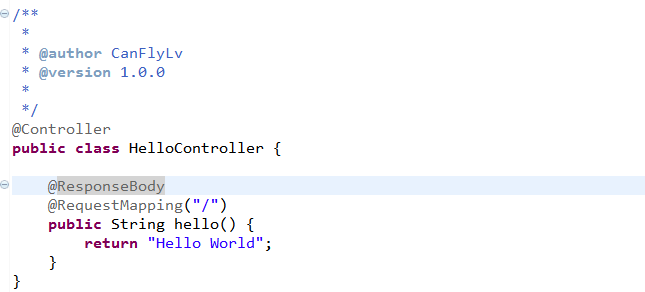
* src/main/java下的程序入口：Chapter1Application
* src/main/resources下的配置文件：application.properties
* src/test/下的测试入口：Chapter1ApplicationTests

③、引入Web模块

当前的pom.xml内容如下，仅引入了两个模块：



④、编写HelloWorld服务，如下图所示：



启动主程序，打开浏览器访问http://localhost:8080/hello，可以看到页面输出Hello World。

通过HelloWorld可以明显的看出，使用Spring Boot大大的简化了我们项目的配置（\*.xml）。

**数据访问**

①、使用Spring-data-jpa简化数据访问层。

1.在pom中添加相关依赖（以MYSQL数据库为例）：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.21</version>

</dependency>

2.在application.properties中配置数据库的相关信息:

spring.datasource.url=jdbc:mysql://localhost:3306/bcms

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

3.创建数据访问接口：

在项目中可以针对不同的实体创建不同的Repository接口实现对该实体的数据访问。

②、使用NoSQL数据库：MongoDB

1.在pom中添加相关的依赖：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-mongodb</artifactId>

</dependency>

2.在application.properties中配置MongoDB数据库相关的信息：

# spring.data.mongodb.uri=mongodb://username:password@host address:port/database

spring.data.mongodb.uri=mongodb://devDBAdmin:E4yunMon90@192.168.8.26:27017/dev

3.使用MongoTemplate实现对mongodb数据库的访问：

在需要访问mongodb的类中注入MongoTemplete,

@Autowired

private MongoTemplate mongoTemplate;

就可以对mongodb数据库操作。

③、使用NoSQL数据库：Redis

1.在pom中添加相关的依赖：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-redis</artifactId>

</dependency>

2.在application.properties中配置redis数据库相关的信息：

# REDIS (RedisProperties)

# Redis数据库索引（默认为0）

spring.redis.database=0

# Redis服务器地址

spring.redis.host=localhost

# Redis服务器连接端口

spring.redis.port=6379

# Redis服务器连接密码（默认为空）

spring.redis.password=

# 连接池最大连接数（使用负值表示没有限制）

spring.redis.pool.max-active=8

# 连接池最大阻塞等待时间（使用负值表示没有限制）

spring.redis.pool.max-wait=-1

# 连接池中的最大空闲连接

spring.redis.pool.max-idle=8

# 连接池中的最小空闲连接

spring.redis.pool.min-idle=0

# 连接超时时间（毫秒）

spring.redis.timeout=0

3.可以通过实现RedisTemplate<K, V>接口来实现对redis数据库的操作。

以上是对三种常用数据库的访问，只是提供了简单的应用，主要的是感受Spring Boot带来的便利。对于具体的使用细节，可以在相关的官网中了解。

**Spring Boot使用Druid和监控配置**

DRUID是阿里巴巴开源平台上一个数据库连接池实现，它结合了C3P0、DBCP、PROXOOL等DB 池的优点，同时加入了日志监控，可以很好的监控DB池连接和SQL的执行情况，可以说是针对监控而生的DB连接池(据说是目前最好的连接池,不知道速度有没有BoneCP快)。业界把 Druid 和 HikariCP 做对比后，虽说 HikariCP 的性能比 Druid 高，但是因为 Druid 包括很多维度的统计和分析功能，所以这也是大家都选择使用它的原因。

接下来说明如何在Spring Boot中使用Druid。

①在pom.xml中添加关于Druid的依赖：

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.0.16</version>

</dependency>

②在application.properties文件中配置相关的配置信息：

# druid -- 数据库连接池

# 主数据源，默认的

# 1.2版本不支持，1.3版本支持，1.4版本不支持(1.4.0不支持，1.4.1支持)

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.url=jdbc:mysql://host address:port/database

spring.datasource.username=

spring.datasource.password=

# 下面为连接池的补充设置，应用到上面所有数据源中

# 初始化大小，最小，最大

spring.datasource.initialSize=5

spring.datasource.minIdle=5

spring.datasource.maxActive=20

# 配置获取连接等待超时的时间

spring.datasource.maxWait=60000

# 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒

spring.datasource.timeBetweenEvictionRunsMillis=60000

# 配置一个连接在池中最小生存的时间，单位是毫秒

spring.datasource.minEvictableIdleTimeMillis=300000

# MySql

spring.datasource.validationQuery=SELECT 1

# ORACLE spring.datasource.validationQuery=SELECT 1 FROM DUAL

spring.datasource.testOnBorrow=false

spring.datasource.testWhileIdle=true

spring.datasource.testOnReturn=false

# 配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙

spring.datasource.filters=stat,wall,log4j

③配置监控功能：

配置Servlet

@WebServlet(urlPatterns="/druid/\*",

initParams={

@WebInitParam(name="allow",value="192.168.16.64,127.0.0.1"),// IP白名单 (没有配置或者为空，则允许所有访问)

@WebInitParam(name="deny",value="192.168.16.103"),// IP黑名单 (存在共同时，deny优先于allow)

@WebInitParam(name="loginUsername",value="admin"),// 用户名

@WebInitParam(name="loginPassword",value="admin"),// 密码

@WebInitParam(name="resetEnable",value="false")// 禁用HTML页面上的“Reset All”功能

})

public class DruidStatViewServlet extends StatViewServlet{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

}

配置filter

/\*\*

\* druid过滤器.

\* @author CanFlyLv

\*/

@WebFilter(filterName="druidWebStatFilter",urlPatterns="/\*",

initParams={

@WebInitParam(name="exclusions",value="\*.js,\*.gif,\*.jpg,\*.bmp,\*.png,\*.css,\*.ico,/druid/\*")// 忽略资源

}

)

public class DruidWebStatFilter extends WebStatFilter{

}

**注：最后在主类上添加@ServletComponentScan标签，@ServletComponentScan是的spring能够扫描到我们自己编写的servlet和filter。**

④启动项目，访问http://localhost:8080/druid/index.html

按照上边的配置就可以使用Druid进行统计和分析。

**Spring Boot中的事务管理**

在Spring Boot中，当我们使用了spring-boot-starter-jdbc或spring-boot-starter-data-jpa依赖的时候，框架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。

当我们的项目中不止一个数据源时，这时候需要在声明事务时，指定不同的事务管理器。这时候在声明事务时，只需要通过指定value的值配置相应的事务管理器。比如： （@Transactional(value="transactionManagerName")）。同时我们还可以指定事务的隔离级别（@Transactional(isolation = Isolation.DEFAULT)）和传播行为（@Transactional(propagation =Propagation.REQUIRED)）。

**消息服务**

主要介绍RabbitMQ在Spring Boot中的应用。

1.在Spring Boot中使用RabbitMQ需要我们在pom.xml中依赖相关的starer,如下：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

2.在application.properties中配置关于RabbitMQ的连接和用户信息：

# RABBIT (RabbitProperties)

spring.rabbitmq.host=/

spring.rabbitmq.addresses=192.168.8.70:5672

spring.rabbitmq.username=guest

spring.rabbitmq.password=guest

3.在需要发送消息的类中依赖注入RabbitTemplate，进行消息的发送，此为消息的生产者（Sender）。然后创建消息的消费者（customer），代码如下：

@Component

public class TestConsumer {

@RabbitListener(queues="key")//启用Rabbit队列监听 key.

public void onMessage(@Payload String message){

//业务操作。

}

}

**定时任务**

Spring boot编写定时任务非常简单，只需要在pom.xml中添加一个最基本的依赖：

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

然后再主类中添加@EnableScheduling注解，启用定时任务的配置。

创建定时任务实现类，如下：

@Component

public class ScheduledTasks {

@Scheduled(corn="\*/5 \* \* \* \* \*")

public void reportCurrentTime() {

//具体的业务操作

}

}

**Spring Boot导入XML配置**

虽然Spring Boot理念就是零配置编程，但是如果绝对需要使用XML的配置，可以从一个@Configuration注解的类开始，可以使用@ImportResouce注解加载XML配置文件，给我们带来的很大的便利。

① 在src/main/resouces下编写配置文件application-bean.xml文件；

② 在确保能被Spring Boot可以扫描到的包路径下建立ConfigClass,eg:

/\*\*

\* 加载xml文件

\* classpath路径：locations={"classpath:application-bean1.xml",

\* "classpath:application-bean2.xml"} file路径： locations =

\* {"file:d:/test/application-bean1.xml"};

\*/

@Configuration

@ImportResource(locations = { "classpath:application-bean.xml" }) //部分spring版本不支持这样写可以用另一种方式

//@ImportResource("classpath\*:spring/\*.xml")

public class ConfigClass {

}

**Spring Boot自定义Servlet,Filter,Listener**

在项目当中很多情况下我们需要定义自己的Servlet，filter，interceptor，或者listener，Spring Boot给我们提供了很方便的方法，接下来就说明如何自定义Servlet，Filter，interceptor或者listener。

自定义Servlet：在自己定义的Servlet类上添加 @WebServlet(urlPatterns="/\*",

initParams={

@WebInitParam(name="",value=""),

@WebInitParam(name="deny",value="192.168.16.103")...

},

description="Servlet的说明")标签；

自定义Filter：在自己定义的Filter类上添加@WebFilter(filterName="filterName",urlPatterns="/\*")标签；

自定义Listener：同理在定义的Listener类上添加@WebListener标签。

自定义interceptor：自定义的interceptor类实现HandlerInterceptor接口，再创建一个Java类继承WebMvcConfigurerAdapter，并重写 addInterceptors 方法，实例化我们自定义的拦截器，然后将对像手动添加到拦截器链中（在addInterceptors方法中添加）。

**注：最后在主类上添加@ServletComponentScan 注解。@ServletComponentScan是的spring能够扫描到我们自己编写的servlet、filter和listener。**