**http://www.cnblogs.com/ityouknow/p/5662753.html**

**章节一**

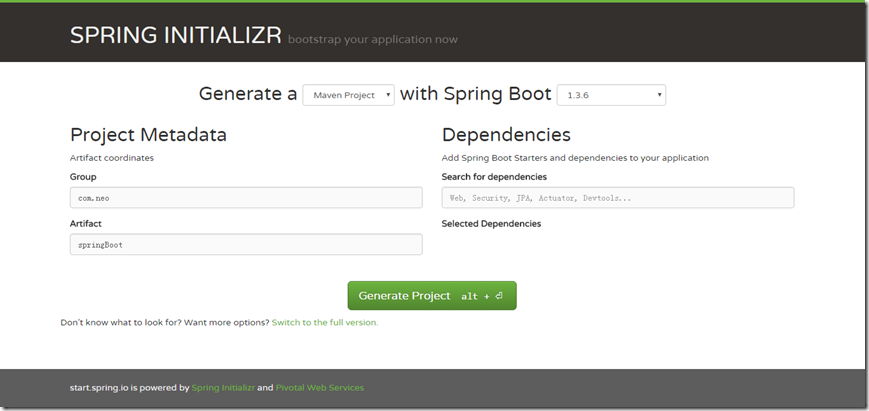
快速入门

说了那么多，手痒痒的很，马上来一发试试!

**maven构建项目**

1、访问http://start.spring.io/

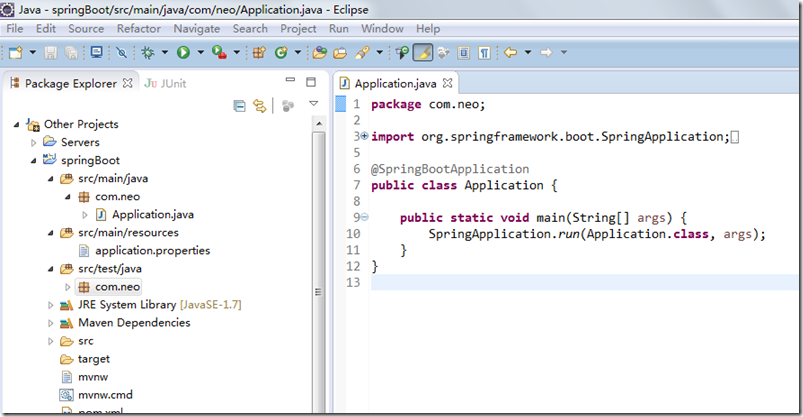
2、选择构建工具Maven Project、Spring Boot版本1.3.6以及一些工程基本信息，点击“[Switch to the full version.](http://start.spring.io/)”java版本选择1.7，可参考下图所示：

[](http://images2015.cnblogs.com/blog/331425/201607/331425-20160712105134607-1615281203.png)

3、点击Generate Project下载项目压缩包

4、解压后，使用eclipse，Import -> Existing Maven Projects -> Next ->选择解压后的文件夹-> Finsh，OK done!

**项目结构介绍**

[](http://images2015.cnblogs.com/blog/331425/201607/331425-20160712105137014-669859839.png)

如上图所示，Spring Boot的基础结构共三个文件:

src/main/java  程序开发以及主程序入口

src/main/resources 配置文件

src/test/java  测试程序

另外，spingboot建议的目录结果如下：

root package结构：com.example.myproject

[复制代码](javascript:void(0);)

com

+- example

+- myproject

+- Application.java

|

+- domain

| +- Customer.java

| +- CustomerRepository.java

|

+- service

| +- CustomerService.java

|

+- controller

| +- CustomerController.java

|

[复制代码](javascript:void(0);)

1、Application.java 建议放到跟目录下面,主要用于做一些框架配置

2、domain目录主要用于实体（Entity）与数据访问层（Repository）

3、service 层主要是业务类代码

4、controller 负责页面访问控制

采用默认配置可以省去很多配置，当然也可以根据自己的喜欢来进行更改

最后，启动Application main方法，至此一个java项目搭建好了！

**引入web模块**

1、pom.xml中添加支持web的模块：

<dependency>

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

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

</dependency>

pom.xml文件中默认有两个模块：

spring-boot-starter：核心模块，包括自动配置支持、日志和YAML；

spring-boot-starter-test：测试模块，包括JUnit、Hamcrest、Mockito。

2、编写controller内容

[复制代码](javascript:void(0);)

@RestController

public class HelloWorldController {

@RequestMapping("/hello")

public String index() {

return "Hello World";

}

}

[复制代码](javascript:void(0);)

@RestController的意思就是controller里面的方法都以json格式输出，不用再写什么jackjson配置的了！

3、启动主程序，打开浏览器访问http://localhost:8080/hello，就可以看到效果了，有木有很简单！

**如何做单元测试**

打开的src/test/下的测试入口，编写简单的http请求来测试；使用mockmvc进行，利用MockMvcResultHandlers.print()打印出执行结果。

[复制代码](javascript:void(0);)

@RunWith(SpringJUnit4ClassRunner.class)

@SpringApplicationConfiguration(classes = MockServletContext.class)

@WebAppConfiguration

public class HelloWorldControlerTests {

private MockMvc mvc;

@Before

public void setUp() throws Exception {

mvc = MockMvcBuilders.standaloneSetup(new HelloWorldController()).build();

}

@Test

public void getHello() throws Exception {

mvc.perform(MockMvcRequestBuilders.get("/hello").accept(MediaType.APPLICATION\_JSON))

.andExpect(MockMvcResultMatchers.status().isOk())

.andDo(MockMvcResultHandlers.print())

.andReturn();

}

}

[复制代码](javascript:void(0);)

**开发环境的调试**

热启动在正常开发项目中已经很常见了吧，虽然平时开发web项目过程中，改动项目启重启总是报错；但springBoot对调试支持很好，修改之后可以实时生效，需要添加以下的配置：

[复制代码](javascript:void(0);)

<dependencies>

    <dependency>

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

        <artifactId>spring-boot-devtools</artifactId>

        <optional>true</optional>

   </dependency>

</dependencies>

<build>

    <plugins>

        <plugin>

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

            <artifactId>spring-boot-maven-plugin</artifactId>

            <configuration>

                <fork>true</fork>

            </configuration>

        </plugin>

   </plugins>

</build>

[复制代码](javascript:void(0);)

该模块在完整的打包环境下运行的时候会被禁用。如果你使用java -jar启动应用或者用一个特定的classloader启动，它会认为这是一个“生产环境”。

**总结**

使用spring boot可以非常方便、快速搭建项目，使我们不用关心框架之间的兼容性，适用版本等各种问题，我们想使用任何东西，仅仅添加一个配置就可以，所以使用sping boot非常适合构建微服务。

# [spring boot(二)：web综合开发](http://www.cnblogs.com/ityouknow/p/5730412.html)

## web开发

spring boot web开发非常的简单，其中包括常用的json输出、filters、property、log等

### json 接口开发

在以前的spring 开发的时候需要我们提供json接口的时候需要做那些配置呢

1. 添加 jackjson 等相关jar包
2. 配置spring controller扫描
3. [对接的方法添加@ResponseBody](mailto:%E5%AF%B9%E6%8E%A5%E7%9A%84%E6%96%B9%E6%B3%95%E6%B7%BB%E5%8A%A0@ResponseBody)

就这样我们会经常由于配置错误，导致406错误等等，spring boot如何做呢，只需要类添加 @RestController 即可，默认类中的方法都会以json的格式返回

@RestController

public class HelloWorldController {

@RequestMapping("/getUser")

public User getUser() {

User user=new User();

user.setUserName("小明");

user.setPassWord("xxxx");

return user;

}

}

如果我们需要使用页面开发只要使用@Controller ，下面会结合模板来说明

### 自定义Filter

我们常常在项目中会使用filters用于录调用日志、排除有XSS威胁的字符、执行权限验证等等。Spring Boot自动添加了OrderedCharacterEncodingFilter和HiddenHttpMethodFilter，并且我们可以自定义Filter。

两个步骤：

1. 实现Filter接口，实现Filter方法
2. 添加@Configurationz 注解，将自定义Filter加入过滤链

好吧，直接上代码

@Configuration

public class WebConfiguration {

@Bean

public RemoteIpFilter remoteIpFilter() {

return new RemoteIpFilter();

}

@Bean

public FilterRegistrationBean testFilterRegistration() {

FilterRegistrationBean registration = new FilterRegistrationBean();

registration.setFilter(new MyFilter());

registration.addUrlPatterns("/\*");

registration.addInitParameter("paramName", "paramValue");

registration.setName("MyFilter");

registration.setOrder(1);

return registration;

}

public class MyFilter implements Filter {

@Override

public void destroy() {

// TODO Auto-generated method stub

}

@Override

public void doFilter(ServletRequest srequest, ServletResponse sresponse, FilterChain filterChain)

throws IOException, ServletException {

// TODO Auto-generated method stub

HttpServletRequest request = (HttpServletRequest) srequest;

System.out.println("this is MyFilter,url :"+request.getRequestURI());

filterChain.doFilter(srequest, sresponse);

}

@Override

public void init(FilterConfig arg0) throws ServletException {

// TODO Auto-generated method stub

}

}

}

### 自定义Property

在web开发的过程中，我经常需要自定义一些配置文件，如何使用呢

### 配置在application.properties中

com.neo.title=纯洁的微笑

com.neo.description=分享生活和技术

自定义配置类

@Component

public class NeoProperties {

@Value("${com.neo.title}")

private String title;

@Value("${com.neo.description}")

private String description;

//省略getter settet方法

}

### log配置

配置输出的地址和输出级别

logging.path=/user/local/log

logging.level.com.favorites=DEBUG

logging.level.org.springframework.web=INFO

logging.level.org.hibernate=ERROR

path为本机的log地址，logging.level 后面可以根据包路径配置不同资源的log级别

## 数据库操作

在这里我重点讲述mysql、spring data jpa的使用，其中mysql 就不用说了大家很熟悉，jpa是利用Hibernate生成各种自动化的sql，如果只是简单的增删改查，基本上不用手写了，spring内部已经帮大家封装实现了。

下面简单介绍一下如何在spring boot中使用

### 1、添加相jar包

<dependency>

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

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

</dependency>

<dependency>

<groupId>mysql</groupId>

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

</dependency>

### 2、添加配置文件

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

spring.datasource.username=root

spring.datasource.password=root

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

spring.jpa.properties.hibernate.hbm2ddl.auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect

spring.jpa.show-sql= true

其实这个hibernate.hbm2ddl.auto参数的作用主要用于：自动创建|更新|验证数据库表结构,有四个值：

1. create： 每次加载hibernate时都会删除上一次的生成的表，然后根据你的model类再重新来生成新表，哪怕两次没有任何改变也要这样执行，这就是导致数据库表数据丢失的一个重要原因。
2. create-drop ：每次加载hibernate时根据model类生成表，但是sessionFactory一关闭,表就自动删除。
3. update：最常用的属性，第一次加载hibernate时根据model类会自动建立起表的结构（前提是先建立好数据库），以后加载hibernate时根据 model类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除以前的行。要注意的是当部署到服务器后，表结构是不会被马上建立起来的，是要等 应用第一次运行起来后才会。
4. validate ：每次加载hibernate时，验证创建数据库表结构，只会和数据库中的表进行比较，不会创建新表，但是会插入新值。

dialect 主要是指定生成表名的存储引擎为InneoDB  
show-sql 是否打印出自动生产的SQL，方便调试的时候查看

### 3、添加实体类和Dao

@Entity

public class User implements Serializable {

private static final long serialVersionUID = 1L;

@Id

@GeneratedValue

private Long id;

@Column(nullable = false, unique = true)

private String userName;

@Column(nullable = false)

private String passWord;

@Column(nullable = false, unique = true)

private String email;

@Column(nullable = true, unique = true)

private String nickName;

@Column(nullable = false)

private String regTime;

//省略getter settet方法、构造方法

}

dao只要继承JpaRepository类就可以，几乎可以不用写方法，还有一个特别有尿性的功能非常赞，就是可以根据方法名来自动的生产SQL，比如findByUserName 会自动生产一个以 userName 为参数的查询方法，比如 findAlll 自动会查询表里面的所有数据，比如自动分页等等。。

[\*\*Entity中不映射成列的字段得加@Transient](mailto:**Entity%E4%B8%AD%E4%B8%8D%E6%98%A0%E5%B0%84%E6%88%90%E5%88%97%E7%9A%84%E5%AD%97%E6%AE%B5%E5%BE%97%E5%8A%A0@Transient) 注解，不加注解也会映射成列\*\*

public interface UserRepository extends JpaRepository<User, Long> {

User findByUserName(String userName);

User findByUserNameOrEmail(String username, String email);

### 4、测试

@RunWith(SpringJUnit4ClassRunner.class)

@SpringApplicationConfiguration(Application.class)

public class UserRepositoryTests {

@Autowired

private UserRepository userRepository;

@Test

public void test() throws Exception {

Date date = new Date();

DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG);

String formattedDate = dateFormat.format(date);

userRepository.save(new User("aa1", "aa@126.com", "aa", "aa123456",formattedDate));

userRepository.save(new User("bb2", "bb@126.com", "bb", "bb123456",formattedDate));

userRepository.save(new User("cc3", "cc@126.com", "cc", "cc123456",formattedDate));

Assert.assertEquals(9, userRepository.findAll().size());

Assert.assertEquals("bb", userRepository.findByUserNameOrEmail("bb", "cc@126.com").getNickName());

userRepository.delete(userRepository.findByUserName("aa1"));

}

}

当让 spring data jpa 还有很多功能，比如封装好的分页，可以自己定义SQL，主从分离等等，这里就不详细讲了

## thymeleaf模板

Spring boot 推荐使用来代替jsp,thymeleaf模板到底是什么来头呢，让spring大哥来推荐，下面我们来聊聊

### Thymeleaf 介绍

Thymeleaf是一款用于渲染XML/XHTML/HTML5内容的模板引擎。类似JSP，Velocity，FreeMaker等，它也可以轻易的与Spring MVC等Web框架进行集成作为Web应用的模板引擎。与其它模板引擎相比，Thymeleaf最大的特点是能够直接在浏览器中打开并正确显示模板页面，而不需要启动整个Web应用。

好了，你们说了我们已经习惯使用了什么 velocity,FreMaker，beetle之类的模版，那么到底好在哪里呢？  
比一比吧  
Thymeleaf是与众不同的，因为它使用了自然的模板技术。这意味着Thymeleaf的模板语法并不会破坏文档的结构，模板依旧是有效的XML文档。模板还可以用作工作原型，Thymeleaf会在运行期替换掉静态值。Velocity与FreeMarker则是连续的文本处理器。  
下面的代码示例分别使用Velocity、FreeMarker与Thymeleaf打印出一条消息：

Velocity: <p>$message</p>

FreeMarker: <p>${message}</p>

Thymeleaf: <p th:text="${message}">Hello World!</p>

\*\* 注意，由于Thymeleaf使用了XML DOM解析器，因此它并不适合于处理大规模的XML文件。\*\*

### URL

URL在Web应用模板中占据着十分重要的地位，[需要特别注意的是Thymeleaf对于URL的处理是通过语法@{](mailto:%E9%9C%80%E8%A6%81%E7%89%B9%E5%88%AB%E6%B3%A8%E6%84%8F%E7%9A%84%E6%98%AFThymeleaf%E5%AF%B9%E4%BA%8EURL%E7%9A%84%E5%A4%84%E7%90%86%E6%98%AF%E9%80%9A%E8%BF%87%E8%AF%AD%E6%B3%95@%7B)...}来处理的。Thymeleaf支持绝对路径URL：

<a th:href="@{http://www.thymeleaf.org}">Thymeleaf</a>

### 条件求值

<a th:href="@{/login}" th:unless=${session.user != null}>Login</a>

### for循环

<tr th:each="prod : ${prods}">

<td th:text="${prod.name}">Onions</td>

<td th:text="${prod.price}">2.41</td>

<td th:text="${prod.inStock}? #{true} : #{false}">yes</td>

</tr>

就列出这几个吧

### 页面即原型

在Web开发过程中一个绕不开的话题就是前端工程师与后端工程师的写作，在传统Java Web开发过程中，前端工程师和后端工程师一样，也需要安装一套完整的开发环境，然后各类Java IDE中修改模板、静态资源文件，启动/重启/重新加载应用服务器，刷新页面查看最终效果。

但实际上前端工程师的职责更多应该关注于页面本身而非后端，使用JSP，Velocity等传统的Java模板引擎很难做到这一点，因为它们必须在应用服务器中渲染完成后才能在浏览器中看到结果，而Thymeleaf从根本上颠覆了这一过程，通过属性进行模板渲染不会引入任何新的浏览器不能识别的标签，例如JSP中的，不会在Tag内部写表达式。整个页面直接作为HTML文件用浏览器打开，几乎就可以看到最终的效果，这大大解放了前端工程师的生产力，它们的最终交付物就是纯的HTML/CSS/JavaScript文件。

## Gradle 构建工具

spring 项目建议使用Gradle进行构建项目，相比maven来讲 Gradle更简洁，而且gradle更时候大型复杂项目的构建。gradle吸收了maven和ant的特点而来，不过目前maven仍然是Java界的主流，大家可以先了解了解。

一个使用gradle配置的项目

buildscript {

repositories {

maven { url "http://repo.spring.io/libs-snapshot" }

mavenLocal()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:1.3.6.RELEASE")

}

}

apply plugin: 'java' //添加 Java 插件, 表明这是一个 Java 项目

apply plugin: 'spring-boot' //添加 Spring-boot支持

apply plugin: 'war' //添加 War 插件, 可以导出 War 包

apply plugin: 'eclipse' //添加 Eclipse 插件, 添加 Eclipse IDE 支持, Intellij Idea 为 "idea"

war {

baseName = 'favorites'

version = '0.1.0'

}

sourceCompatibility = 1.7 //最低兼容版本 JDK1.7

targetCompatibility = 1.7 //目标兼容版本 JDK1.7

repositories { // Maven 仓库

mavenLocal() //使用本地仓库

mavenCentral() //使用中央仓库

maven { url "http://repo.spring.io/libs-snapshot" } //使用远程仓库

}

dependencies { // 各种 依赖的jar包

compile("org.springframework.boot:spring-boot-starter-web:1.3.6.RELEASE")

compile("org.springframework.boot:spring-boot-starter-thymeleaf:1.3.6.RELEASE")

compile("org.springframework.boot:spring-boot-starter-data-jpa:1.3.6.RELEASE")

compile group: 'mysql', name: 'mysql-connector-java', version: '5.1.6'

compile group: 'org.apache.commons', name: 'commons-lang3', version: '3.4'

compile("org.springframework.boot:spring-boot-devtools:1.3.6.RELEASE")

compile("org.springframework.boot:spring-boot-starter-test:1.3.6.RELEASE")

compile 'org.webjars.bower:bootstrap:3.3.6'

compile 'org.webjars.bower:jquery:2.2.4'

compile("org.webjars:vue:1.0.24")

compile 'org.webjars.bower:vue-resource:0.7.0'

}

bootRun {

addResources = true

}

## WebJars

WebJars是一个很神奇的东西，可以让大家以jar包的形式来使用前端的各种框架、组件。

### 什么是WebJars

什么是WebJars？WebJars是将客户端（浏览器）资源（JavaScript，Css等）打成jar包文件，以对资源进行统一依赖管理。WebJars的jar包部署在Maven中央仓库上。

### 为什么使用

我们在开发Java web项目的时候会使用像Maven，Gradle等构建工具以实现对jar包版本依赖管理，以及项目的自动化管理，但是对于JavaScript，Css等前端资源包，我们只能采用拷贝到webapp下的方式，这样做就无法对这些资源进行依赖管理。那么WebJars就提供给我们这些前端资源的jar包形势，我们就可以进行依赖管理。

### 如何使用

1、 [WebJars主官网](http://www.webjars.org/bower) 查找对于的组件，比如Vuejs

<dependency>

<groupId>org.webjars.bower</groupId>

<artifactId>vue</artifactId>

<version>1.0.21</version>

</dependency>

2、页面引入

<link th:href="@{/webjars/bootstrap/3.3.6/dist/css/bootstrap.css}" rel="stylesheet"></link>

# **spring-data-redis序列化策略**

[**spring**](http://lib.csdn.net/base/javaee)-data-[**Redis**](http://lib.csdn.net/base/redis)提供了多种serializer策略，这对使用jedis的开发者而言，实在是非常便捷。sdr提供了4种内置的serializer：

* JdkSerializationRedisSerializer：使用JDK的序列化手段(serializable接口，ObjectInputStrean，ObjectOutputStream)，数据以字节流存储，jdk序列化和反序列化数据
* StringRedisSerializer：字符串编码，数据以string存储
* JacksonJsonRedisSerializer：json格式存储
* OxmSerializer：xml格式存储

其中JdkSerializationRedisSerializer和StringRedisSerializer是最基础的序列化策略，其中“JacksonJsonRedisSerializer”与“OxmSerializer”都是基于stirng存储，因此它们是较为“高级”的序列化(最终还是使用string解析以及构建[**Java**](http://lib.csdn.net/base/java)对象)。

RedisTemplate中需要声明4种serializer，默认为“JdkSerializationRedisSerializer”：

1) keySerializer ：对于普通K-V操作时，key采取的序列化策略  
2) valueSerializer：value采取的序列化策略  
3) hashKeySerializer： 在hash[**数据结构**](http://lib.csdn.net/base/datastructure)中，hash-key的序列化策略  
4) hashValueSerializer：hash-value的序列化策略

无论如何，建议key/hashKey采用StringRedisSerializer