**Spring Boot框架探研**

# Spring Boot 介绍

Spring Boot是由Pivotal团队提供的全新框架，***其设计目的是用来简化新Spring应用的初始搭建以及开发过程***。

该框架使用了特定的方式(继承starter，约定优先于配置)来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式，Boot致力于在蓬勃发展的快速应用开发领域（rapid application development）成为领导者。

Spring Boot并不是一个框架，从根本上将，**它就是一些库的集合**，maven或者gradle项目导入相应依赖即可使用Spring Boot，而且无需自行管理这些库的版本。

## Spring Boot的特点

1：创建独立的spring应用。

2：**嵌入Tomcat, Jetty Undertow 而且不需要部署他们**。

3：提供的“starters”poms来简化Maven配置

4：**尽可能自动配置spring应用**。

5：提供生产指标,健壮检查和外部化配置

6：**绝对没有代码生成和XML配置要求**

。。。。。。

## Spring Boot运行环境

Spring Boot因为是一个最新开发的框架，所以只支持java6以上，Java7最好，官方推荐Java8。

## Spring Boot启动器

Starter POMs是可以包含到应用中的一个方便的依赖关系描述符集合。你可以获取所有spring及相关技术的一站式服务，而不需要翻阅示例代码，拷贝粘贴大量的依赖描述符。例如，如果你想使用Spring和JPA进行数据库访问，只需要在你的项目中包含spring-boot-starter-data-jpa依赖，然后你就可以开始了。

该starters包含很多你搭建项目，快速运行所需的依赖，并提供一致的，管理的传递依赖集。

名字有什么含义：所有的starters遵循一个相似的命名模式：spring-boot-starter-\*，在这里\*是一种特殊类型的应用程序。该命名结构旨在帮你找到需要的starter。很多IDEs集成的Maven允许你通过名称搜索依赖。例如，使用相应的Eclipse或STS插件，你可以简单地在POM编辑器中点击ctrl-space，然后输入”spring-boot-starter”可以获取一个完整列表。

例举几个主要的Starters。

|  |  |
| --- | --- |
| 名称 | 描述 |
| spring-boot-starter | 核心Spring Boot starter，包括自动配置支持，日志和YAML |
| spring-boot-starter-data-jpa | 对”Java持久化API”的支持，包括spring-data-jpa，spring-orm和Hibernate |
| 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-jdbc | 对JDBC数据库的支持 |
| spring-boot-starter-security | 对spring-security的支持 |
| spring-boot-starter-test | 对常用测试依赖的支持，包括JUnit, Hamcrest和Mockito，还有spring-test模块 |
| spring-boot-starter-thymeleaf | 对Thymeleaf模板引擎的支持，包括和Spring的集成 |
| spring-boot-starter-web | 对全栈web开发的支持，包括Tomcat和spring-webmvc |
|  |  |

## 项目搭建前奏

在主流的项目开发过程中，经常用到一些构建管理工具，譬如耳熟能详的Maven。

我们在开发Spring Boot 应用程式时，也常用到相关的构建管理工具。在这里我们先了解一些主流的构建管理工具。

### 使用Maven

### 什么是Maven

Maven 是一个顶级的 Apache Software Foundation 开源项目，创建它最初是为了管理 Jakarta Turbine 项目复杂的构建过程。从那以后，不论是开源开发项目还是私有开发项目都选择 Maven 作为项目构建系统。Maven 快速地发展着，如今已是第二版，Maven 已经从针对单个复杂项目的定制构建工具成长为广泛使用的**构建管理系统**，其丰富的功能可以应用于大多数的软件开发场景。

#### Maven 配置

* 下载页地址:

http://maven.apache.org/download.cgi

* 安装配置:

1、解压包;

2、添加新的系统环境变量MAVEN\_HOME， 并设置其值为你安装的目录

MAVEN\_HOME= D:\xxxxxx\apache-maven-3.2.2

3、更新系统PATH 变量， 添加;%MAVEN\_HOME%\bin;到尾部

4、测试一下 mvn -v

* 修改Maven仓库位置

目标:

修改Maven仓库为Maven安装目录的repository目录

操作:

找到，Maven安装目录\conf\settings.xml这个文件。打开,

添加

<localRepository>

D:/xxxxxx/apache-maven-3.5.0/repository

</localRepository>

* Eclipse IDE配置

1、Windows-->Prefrences,点击Maven的右边的三角符号，以展开Maven的配置界面

2、之后，点击Maven下面的Installations，选择Maven的安装目录，并点击确定。

3、然后在User Settings里面的Global选择刚才修改过的settings.xml文件，看看Local Repository项是否变成了刚才修改过的路径

### 使用Gradle

待补充......

### 私服仓库.

在这里主要介绍的是Sonatype.org 出品的开源私服仓库 **Nexus Repository Manager** ，该软件可支持Maven、Gradle等主流构建管理软件，

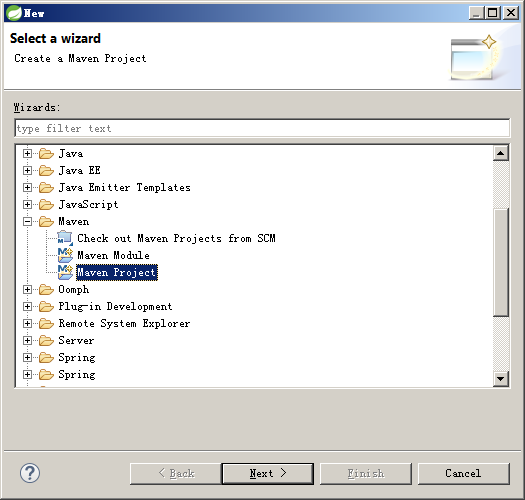
下载地址:

https://www.sonatype.com/download-oss-sonatype

## Spring Boot 应用搭建

使用Spring Boot能简化Spring配置的特性，在这里简单介绍一下利用Spring Boot微框架如何搭建起一个Spring应用。

### 创建一个Maven 项目



### 配置pom.xml 文件

(1) 继承父项目,获取Spring Boot必要依赖包

<parent>

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

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

<version>1.5.2.RELEASE</version>

<relativePath />

</parent>

(2) 加入其他依赖包

<dependencies>

<!-- Spring Boot -->

<dependency>

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

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

</dependency>

<!-- -->

<dependency>

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

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

</dependency>

<!-- -->

<dependency>

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

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

<scope>provided</scope>

</dependency>

<!-- 只需引入spring-boot-devtools 即可实现热部署 -->

<dependency>

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

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

</dependency>

<!-- -->

<dependency>

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

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

<scope>test</scope>

</dependency>

<!-- Servlet -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

</dependency>

<!-- 数据库连接驱动 -->

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.26</version>

</dependency>

<dependency>

<groupId>com.microsoft.sqlserver</groupId>

<artifactId>sqljdbc4</artifactId>

<version>4.0</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>1.0.25</version>

</dependency>

</dependencies>

### 添加应用配置类Application Class



为 Application 类添加@SpringBootApplication注解，即注明该类为Spring Boot应用。

该类继承于SpringBootServletInitializer ,即该类继承Servlet启动器。覆盖SpringBootServletInitializer 中configure方法引用SpringBoot启动器类。

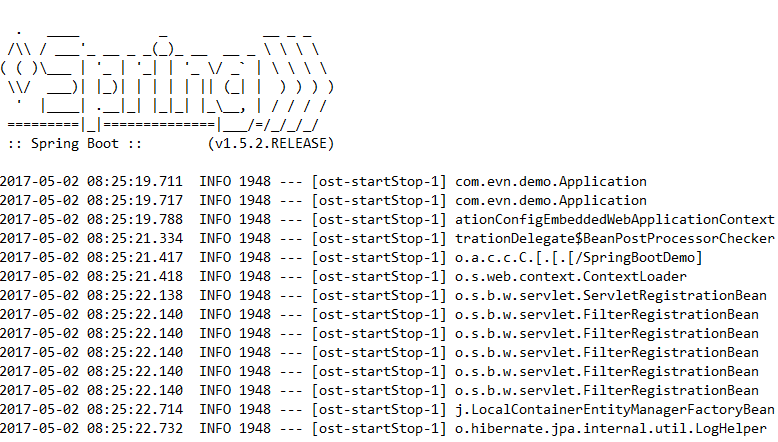
### 添加应用配置文件 Application Properties

在src/main/resources目录中添加一个application.properties配置文件，在配置文件内添加Spring Boot相关配置属性。

### 测试启动

启动Spring Boot应用基本上有两种。Main方法启动 和 在Tomcat中启动。

Spring Boot应用启动效果如下:



## Spring Boot 的部署

### 多环境配置

在Spring Boot 配置选项中有一个特殊参数#spring.profiles.active ,通过这个参数可以指定有效的配置文件，利用该参数可以选用不同配置文件。

实用案例:

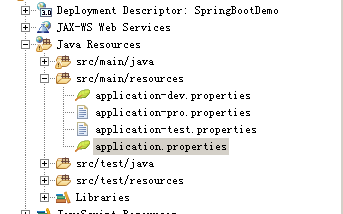
当我们项目有多个环境时，包括:开发环境(dev)、测试环境(test)、生产环境(pro)。

命名各个环境的配置文件为:

开发环境配置文件 : application-dev.properties

测试环境配置文件 : application-test.properties

生产环境配置文件 : application-pro.properties



在主配置文件件 application.properties 中，先配置好可公用部分的配置属性，然后加入spring.profiles.active 属性指定再加载的配置文件。

注意：

*在指定了spring.profiles.active属性后，指定的配置文件的属性会覆盖默认配置文件的属性。*

# Spring Framework 介绍

## Spring Framework常用注解

### @Autowired 自动装配

@Autowired可以对成员变量、方法和构造函数进行标注，来完成自动装配的工作。

这里必须明确：@Autowired是**根据类型进行自动装配**的，如果需要按名称进行装配，则需要配合@Qualifier使用；

@Autowired标注可以放在成员变量上，也可以放在成员变量的set方法上。前者，Spring会直接将UserDao类型的唯一一个bean赋值给userDao这个成员变量；后者，Spring会调用setUserDao方法来将UserDao类型的唯一一个bean装配到userDao这个属性。

### @Component 命名组件

当Spring应用配置了AspectJ切面风格编程时，可以通过@Compenent注解标识其为Spring管理Bean，而@Aspect注解不能被Spring自动识别并注册为Bean，必须通过@Component注解来完成。

### @Service 业务逻辑组件

标注业务层组件，即SERVICE组件。

### @Transactional 事务控制

### @Repository 数据访问

标注数据访问组件，即DAO组件。

# Spring MVC 介绍

Spring MVC框架是有一个MVC框架，通过实现Model-View-Controller模式来很好地将数据、业务与展现进行分离。从这样一个角度来说，Spring MVC和Struts、Struts2非常类似。Spring MVC的设计是围绕DispatcherServlet展开的，DispatcherServlet负责将请求派发到特定的handler。通过可配置的handler mappings、view resolution、locale以及theme resolution来处理请求并且转到对应的视图。

## Spring MVC 常用注解

在日常的 SpringMVC 开发中，会经常用到一些注解声明类、方法等。下面对一部分常用的注解详细说明一下。

### @Controller 控制器

在SpringMVC 中，控制器Controller 负责处理由DispatcherServlet 分发的请求，它把用户请求的数据经过业务处理层处理之后封装成一个Model ，然后再把该Model 返回给对应的View 进行展示。在SpringMVC 中提供了一个非常简便的定义Controller 的方法，你无需继承特定的类或实现特定的接口，只需使用@Controller 标记一个类是Controller ，然后使用@RequestMapping 和@RequestParam 等一些注解用以定义URL 请求和Controller 方法之间的映射，这样的Controller 就能被外界访问到。此外Controller 不会直接依赖于HttpServletRequest 和HttpServletResponse 等HttpServlet 对象，它们可以通过Controller 的方法参数灵活的获取到。

@Controller 用于标记在一个类上，使用它标记的类就是一个SpringMVC Controller 对象。分发处理器将会扫描使用了该注解的类的方法，并检测该方法是否使用了@RequestMapping 注解。@Controller 只是定义了一个控制器类，而使用@RequestMapping 注解的方法才是真正处理请求的处理器。单单使用@Controller 标记在一个类上还不能真正意义上的说它就是SpringMVC 的一个控制器类，因为这个时候Spring 还不认识它。那么要如何做Spring 才能认识它呢？这个时候就需要我们把这个控制器类交给Spring 来管理。有两种方式：

（1）在SpringMVC 的配置文件中定义MyController 的bean 对象。

（2）在SpringMVC 的配置文件中告诉Spring 该到哪里去找标记为@Controller 的Controller 控制器。

<!--方式一-->

<bean class="com.host.app.web.controller.MyController"/>

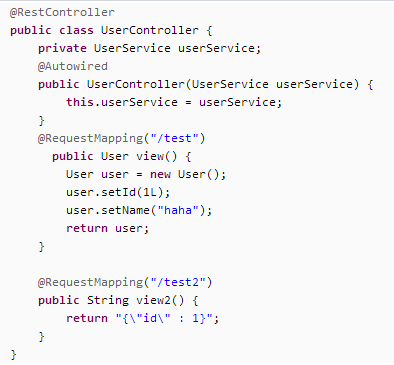
<!--方式二-->

< context:component-scan base-package = "com.host.app.web" />//路径写到controller的上一层(扫描包详解见下面浅析)

### @RestController 控制器(RESTFull)

该注解整合了 **@Controller** 和 **@ResponseBody** 。使用了这个注解的类会被看作一个controller。controller中使用@RequestMapping的方法有一个默认的@ResponseBody注解。@ResponseBody也可以加到类一级，通过继承方法一级不需要添加。

示例代码:



### @RequestMapping 地址映射

RequestMapping是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径。

RequestMapping注解有六个属性，下面我们把她分成三类进行说明（下面有相应示例）。

1、 value， method；

value： 指定请求的实际地址，指定的地址可以是URI Template 模式（后面将会说明）；

method： 指定请求的method类型， GET、POST、PUT、DELETE等；

2、consumes，produces

consumes： 指定处理请求的提交内容类型（Content-Type），例如application/json, text/html;

produces: 指定返回的内容类型，仅当request请求头中的(Accept)类型中包含该指定类型才返回；

3、params，headers

params： 指定request中必须包含某些参数值是，才让该方法处理。

headers： 指定request中必须包含某些指定的header值，才能让该方法处理请求。

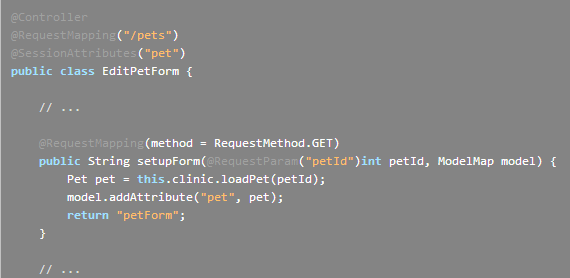
### @RequestParam 传值参数

A） 常用来处理简单类型的绑定，通过Request.getParameter() 获取的String可直接转换为简单类型的情况（ String--> 简单类型的转换操作由ConversionService配置的转换器来完成）；因为使用request.getParameter()方式获取参数，所以可以处理get 方式中queryString的值，也可以处理post方式中 body data的值；

B）用来处理Content-Type: 为 application/x-www-form-urlencoded编码的内容，提交方式GET、POST；

C) 该注解有两个属性： value、required； value用来指定要传入值的id名称，required用来指示参数是否必须绑定；

示例代码-1：



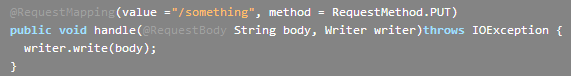
### @RequestBody

该注解常用来处理Content-Type: 不是application/x-www-form-urlencoded编码的内容，例如application/json, application/xml等；

它是通过使用HandlerAdapter 配置的HttpMessageConverters来解析post data body，然后绑定到相应的bean上的。

因为配置有FormHttpMessageConverter，所以也可以用来处理 application/x-www-form-urlencoded的内容，处理完的结果放在一个MultiValueMap<String, String>里，这种情况在某些特殊需求下使用，详情查看FormHttpMessageConverter api;

示例代码-1：

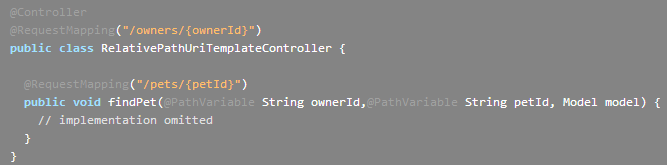


### @ResponseBody

### @PathVariable

当使用@RequestMapping URI template 样式映射时， 即 someUrl/{paramId}, 这时的paramId可通过 @Pathvariable注解绑定它传过来的值到方法的参数上。

示例代码：



# Spring Data JPA

## JPA

JPA全称 Java Persistence API .JPA通过JDK 5.0注解或XML描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中。

## JPA 常用注解

### @Entity

说明:

设置Pojo为实体。

参数:

name : 指定实体Bean的名称,默认值为 bean class 的非限定类名

### @Table

说明:

设置实体对应的表名。

参数:

name:指定表的名称

catalog:指定数据库名称

schema:指定数据库的用户名

uniqueConstraints:指定唯一性字段约束,如为personid 和name 字段指定唯一性约束

### @Id

说明:

设置主键，映射到数据库表的主键的属性，一个实体只能有一个属性被映射为主键。

### @GeneratedValue

说明:

设置主键的生成策略。

参数说明:

strategy:表示主键生成策略,有AUTO (让ORM框架自动选择) , INDENTITY (根据数据库的Identity字段生成 ) ,SEQUENCE (根据数据库表的Sequence字段生成) 和 TABLE (以有根据一个额外的表生成主键) 4种 ,默认为AUTO

generator:表示主键生成器的名称,这个属性通常和ORM框架相关,例如,Hibernate可以指定uuid等主键生成方式.

### @Column

说明:

设置字段类型。

参数说明:

通过@Column注解设置，包含的设置如下

name：字段名

unique：是否唯一

nullable：是否可以为空

inserttable：是否可以插入

updateable：是否可以更新

columnDefinition: 定义建表时创建此列的DDL

secondaryTable: 从表名。如果此列不建在主表上（默认建在主表），该属性定义该列所在从表的名字。

### @Temporal

说明:

设置时间类型。

参数说明:

TemporalType.DATE : 日期

TemporalType.TIME : 时间

TemporalType.TIMESTAMP : 时间戳

### @CreationTimestamp

说明:

指定数据创建时间戳。

### @UpdateTimestamp

说明:

指定数据更新时间戳。

### @JsonFormat

说明:

JSON格式化。用于属性或者方法上（最好是属性上），可以方便的把Date类型直接转化为我们想要的模式，比如@JsonFormat(pattern = "yyyy-MM-dd HH-mm-ss")

# Spring Security 介绍

## Spring Security 常用注解

待补充......

# Thymeleaf 模板框架介绍

Thymeleaf是一个**XML/XHTML/HTML5**模板引擎，可用于Web与非Web环境中的应用开发。它是一个开源的Java库，基于Apache License 2.0许可，由Daniel Fernández创建，该作者还是Java加密库Jasypt的作者。

Thymeleaf提供了一个***用于整合Spring MVC的可选模块***，在应用开发中，你可以使用Thymeleaf来完全代替JSP，或其他模板引擎，如Velocity、FreeMarker等。Thymeleaf的主要目标在于提供一种可被浏览器正确显示的、格式良好的模板创建方式，因此也可以用作静态建模。你可以使用它创建经过验证的XML与HTML模板。相对于编写逻辑或代码，开发者只需将标签属性添加到模板中即可。接下来，这些标签属性就会在DOM（文档对象模型）上执行预先制定好的逻辑。

## 在Spring Boot中使用Thymeleaf

在POM.XML中添加:

<dependency>

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

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

</dependency>

# 参考资料

*[1.Spring Boot教程](http://blog.csdn.net/qiutongyeluo/article/details/51883084)*

*[2.Spring Boot Reference Guide](http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/)*

*[3.Apache Maven 2 简介](https://www.ibm.com/developerworks/cn/education/java/j-mavenv2/j-mavenv2.html)*

*[4.Spring Boot自动配置原理](http://www.jianshu.com/p/346cac67bfcc)*

*[5.使用 Spring Boot 快速构建 Spring 框架应用](https://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/index.html)*

*[6.使用 Spring Data JPA 简化 JPA 开发](https://www.ibm.com/developerworks/cn/opensource/os-cn-spring-jpa/)*