**框架**

尚硅谷java研究院

版本：V 1.0

# 简单了解框架

框架，即framework。其实就是某种应用的半成品，就是一组组件，供你选用完成你自己的系统。简单说就是使用别人搭好的舞台，你来做表演。而且，框架一般是成熟的，不断升级的软件。

框架是对特定应用领域中的应用系统的部分设计和实现的整体结构。

因为软件系统发展到今天已经很复杂了，特别是服务器端软件，涉及到的知识，内容，问题太多。在某些方面使用别人成熟的框架，就相当于让别人帮你完成一些基础工作，你只需要集中精力完成系统的业务[逻辑设计](https://baike.baidu.com/item/%E9%80%BB%E8%BE%91%E8%AE%BE%E8%AE%A1" \t "_blank)。而且框架一般是成熟，稳健的，他可以处理系统很多细节问题，比如，事务处理，安全性，数据流控制等问题。还有框架一般都经过很多人使用，所以结构很好，所以扩展性也很好，而且它是不断升级的，你可以直接享受别人升级代码带来的好处。

# 第1章 Spring

## 1.1 Spring 概述

1. Spring是一个开源框架
2. Spring为简化企业级开发而生，使用Spring，JavaBesan就可以实现很多以前要靠EJB才能实现的功能。同样的功能，在EJB中要通过繁琐的配置和复杂的代码才能够实现，而在Spring中却非常的优雅和简洁。
3. Spring是一个**IOC**(DI)和**AOP**容器框架。
4. Spring的优良特性

① **非侵入式**：基于Spring开发的应用中的对象可以不依赖于Spring的API

② **依赖注入**：DI——Dependency Injection，反转控制(IOC)最经典的实现。

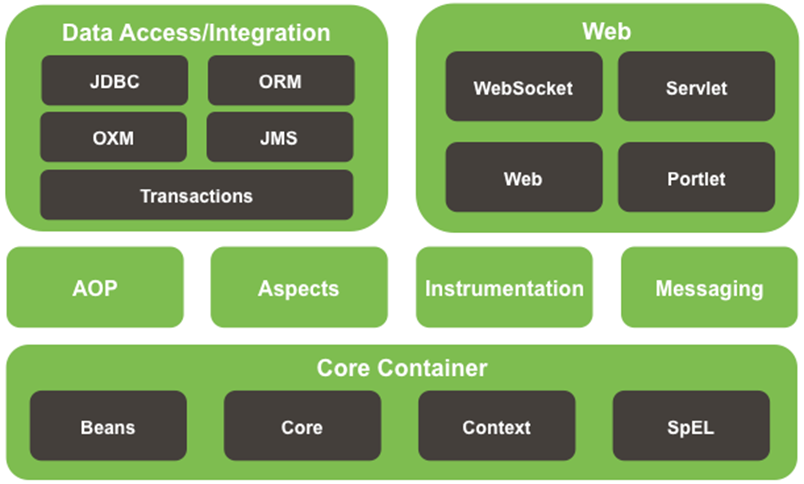
③ **面向切面编程**：Aspect Oriented Programming——AOP

④ **容器**：Spring是一个容器，因为它包含并且管理应用对象的生命周期

**⑤ 组件化**：Spring实现了使用简单的组件配置组合成一个复杂的应用。在 Spring 中可以使用XML和Java注解组合这些对象。

**⑥ 一站式**：在IOC和AOP的基础上可以整合各种企业应用的开源框架和优秀的第三方类库（实际上Spring 自身也提供了表述层的SpringMVC和持久层的Spring JDBC）。

1. Spring模块



## 1.2 Spring HelloWorld

1. 创建一个Maven版的Java工程
2. 在pom.xml中加入对Spring的依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.0.0.RELEASE</version>  </dependency> |

1. 创建Spring的核心配置文件

* File->New->Spring Bean Configuration File
* 为文件取名字 例如：applicationContext.xml

1. 基于xml

略。见课堂演示。

1. 基于注解

常用注解标识组件

* 普通组件：@Component

标识一个受Spring IOC容器管理的组件

* 持久化层组件：@Repository

标识一个受Spring IOC容器管理的持久化层组件

* 业务逻辑层组件：@Service

标识一个受Spring IOC容器管理的业务逻辑层组件

* 表述层控制器组件：@Controller

标识一个受Spring IOC容器管理的表述层控制器组件

组件命名规则

* 默认情况：使用组件的简单类名首字母小写后得到的字符串作为bean的id
* 使用组件注解的value属性指定bean的id
* 注意：事实上Spring并没有能力识别一个组件到底是不是它所标记的类型，即使将@Respository注解用在一个表述层控制器组件上面也不会产生任何错误，所以 @Respository、@Service、@Controller这几个注解仅仅是为了让开发人员自己 明确当前的组件扮演的角色。

@Autowired注解

* 根据类型实现自动装配。
* 构造器、普通字段(即使是非public)、一切具有参数的方法都可以应用@Autowired 注解
* 默认情况下，所有使用@Autowired注解的属性都需要被设置。当Spring找不到匹 配的bean装配属性时，会抛出异常。
* 若某一属性允许不被设置，可以设置@Autowired注解的required属性为 false
* 默认情况下，当IOC容器里存在多个类型兼容的bean时，Spring会尝试匹配bean 的id值是否与变量名相同，如果相同则进行装配。如果bean的id值不相同，通过类型的自动装配将无法工作。此时可以在@Qualifier注解里提供bean的名称。

Spring甚至允许在方法的形参上标注@Qualifiter注解以指定注入bean的名称。

# 第2章 Spring WEB MVC

## 2.1 Spring Web MVC 概述

1. 一种轻量级的、基于MVC的Web层应用框架。偏前端而不是基于业务逻辑层。Spring框架的一个后续产品。Spring 为展现层提供的基于 MVC 设计理念的优秀的 Web 框架，是目前最主流的MVC 框架之一
2. Spring3.0 后全面超越 Struts2，成为最优秀的 MVC 框架
3. Spring MVC 通过一套 MVC 注解，让 POJO 成为处理请求的控制器，而无须实现任何接口。
4. 支持 REST 风格的 URL 请求。
5. 采用了松散耦合可插拔组件结构，比其他 MVC 框架更具扩展性和灵活性。

## 2.2 Spring Web MVC HelloWorld

1. 创建一个Maven版的WEB工程
2. 在pom.xml中加入对Springmvc的依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>4.0.0.RELEASE</version>  </dependency> |

1. 在web.xml中配置前端控制器DispatcherServlet

|  |
| --- |
| <servlet>  <servlet-name>springDispatcherServlet</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>springDispatcherServlet</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping> |

1. 创建Springmvc的核心配置文件

|  |
| --- |
| <!-- 组件扫描 -->  <context:component-scan base-package=*"基本包 "*></context:component-scan>    <!-- 视图解析器 -->  <bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/views/"*></property>  <property name=*"suffix"* value=*".jsp"*></property>  </bean> |

1. 编写index.jsp页面，通过浏览器端发送请求

|  |
| --- |
| <a href=*"hello"*>Hello Springmvc </a> |

1. 编写请求处理器

|  |
| --- |
| /\*\*  \* 请求处理器  \*/  @Controller  **public** **class** HelloWorldHandler {  } |

1. 编写请求处理方法，处理客户端的请求

|  |
| --- |
| /\*\*  \* 请求处理方法  \* 浏览器端: http://localhost:8080/Springmvc01/hello  \* **@RequestMapping**: 请求映射. 指定哪个请求交给哪个方法处理.  \*/  @RequestMapping(value="/hello")  **public** String handleHello() {  System.***out***.println("Hello Springmvc .");  **return** "success";  } |

1. 编写视图，呈现结果

根据视图解析器中prefix中的配置，在WEB-INF目录下创建views目录，并在views目录中创建success.jsp页面

1. 将工程部署到Tomcat中，并启动Tomcat
2. 浏览器端发送请求进行访问

## 2.3 请求参数&响应数据

1. @RequestParam 处理请求参数
2. ModelAndView Map Model 处理响应数据

## 2.4 REST

1. RESTful URL
2. @PathVariable

# 第3章 MyBatis

## 3.1 MyBatis 概述

1. MyBatis是Apache的一个开源项目iBatis, 2010年6月这个项目由Apache Software Foundation 迁移到了Google Code，随着开发团队转投Google Code旗下， iBatis3.x正式更名为MyBatis ，代码于2013年11月迁移到Github
2. MyBatis 是支持定制化 SQL、存储过程以及高级映射的优秀的持久层框架
3. MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集
4. MyBatis可以使用简单的XML或注解用于配置和原始映射，将接口和Java的POJO（Plain Old Java Objects，普通的Java对象）映射成数据库中的记录
5. Mybatis 是一个 半自动的ORM（Object Relation Mapping）框架
6. MyBatis 下载网址 https://github.com/mybatis/mybatis-3/

## 3.2 MyBatis HelloWorld

1. 创建一个Maven版的java工程
2. 在pom.xml中加入对MyBatis的依赖

|  |
| --- |
| <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.4.1</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.37</version>  </dependency>    <dependency>  <groupId>log4j</groupId>  <artifactId>log4j</artifactId>  <version>1.2.14</version>  </dependency> |

1. 导入log4j的配置文件
2. 创建测试表

|  |
| --- |
| -- 创建库  CREATE DATABASE test\_mybatis;  -- 使用库  USE test\_mybatis;  -- 创建表  CREATE TABLE tbl\_employee(  id INT(11) PRIMARY KEY AUTO\_INCREMENT,  last\_name VARCHAR(50),  email VARCHAR(50),  gender CHAR(1)  ); |

1. 创建JavaBean

|  |
| --- |
| **public** **class** Employee {  **private** Integer id ;  **private** String lastName;  **private** String email ;  **private** String gender ;  **public** Integer getId() {  **return** id;  }  **public** **void** setId(Integer id) {  **this**.id = id;  }  **public** String getLastName() {  **return** lastName;  }  **public** **void** setLastName(String lastName) {  **this**.lastName = lastName;  }  **public** String getEmail() {  **return** email;  }  **public** **void** setEmail(String email) {  **this**.email = email;  }  **public** String getGender() {  **return** gender;  }  **public** **void** setGender(String gender) {  **this**.gender = gender;  }  @Override  **public** String toString() {  **return** "Employee [id=" + id + ", lastName=" + lastName + ", email=" + email + ", gender=" + gender + "]";  } |

1. 创建Mybatis的全局配置文件，参考Mybatis的官方手册

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE configuration  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-config.dtd">  <configuration>  <!-- 数据库连接环境的配置 -->  <environments default=*"development"*>  <environment id=*"development"*>  <transactionManager type=*"JDBC"* />    <dataSource type=*"POOLED"*>  <property name=*"driver"* value=*"com.mysql.jdbc.Driver"* />  <property name=*"url"* value=*"jdbc:mysql://localhost:3306/mybatis\_1129"* />  <property name=*"username"* value=*"root"* />  <property name=*"password"* value=*"1234"* />  </dataSource>  </environment>  </environments>  <!-- 引入SQL映射文件,Mapper映射文件 -->  <mappers>  <package name=*"Mapper映射文件所在的包 "* />  </mappers>  </configuration> |

1. 创建Mapper接口

|  |
| --- |
| /\*\*  \* Mapper接口，实际上就是我们非常熟悉的Dao.  \*/  **public** **interface** EmployeeMapper {  //根据ID查询Employee  **public** Employee getEmployeeById(Integer id );  } |

1. 创建Mapper映射文件

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"* ?>  <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <!--  Mapper映射文件: 定义CRUD的SQL语句  -->  <mapper namespace=*"com.atguigu.mybatis.mapper.EmployeeMapper"*>  <!-- public Employee getEmployeeById(Integer id ); -->  <select id=*"getEmployeeById"* resultType=*"com.atguigu.mybatis.beans.Employee"* parameterType=*"java.lang.Integer"*>  select id,last\_name, email,gender from tbl\_employee where id = #{id }  </select>  </mapper> |

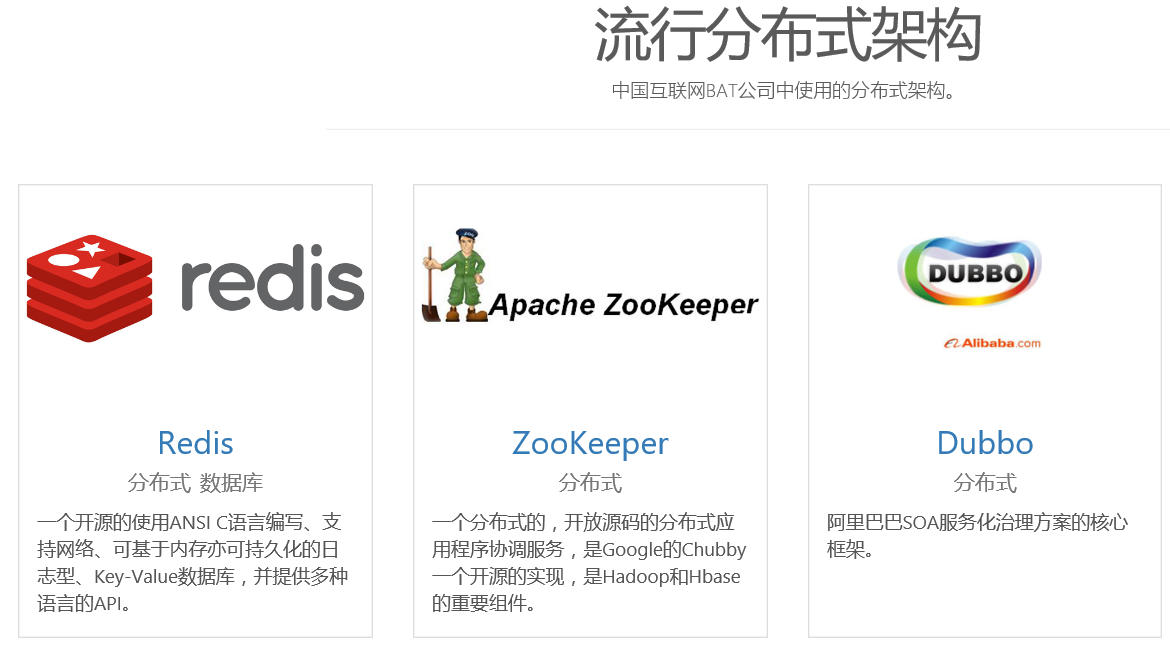
1. 测试

|  |
| --- |
| @Test  **public** **void** test() **throws** Exception{  String resource = "mybatis-config.xml";  InputStream inputStream =Resources.*getResourceAsStream*(resource);  SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder()  .build(inputStream);  SqlSession session = sqlSessionFactory.openSession();  **try** {  //Mapper接口:获取Mapper接口的 代理实现类对象  EmployeeMapper mapper =session.getMapper(EmployeeMapper.**class**);  Employee employee = mapper.getEmployeeById(1006);  System.***out***.println(employee);  } **finally** {  session.close();  }  } |

## 3.3 CRUD

# 第4章 SpringBoot

## 4.1 流行分布式架构



## 4.2 Spring分布式架构



## 4.3 SpringBoot 概述

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，Spring 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。

此课件基于Maven创建项目，如果对于Maven工具不是很了解，请自行学习相关课件

## 4.4 为什么要使用SpringBoot

说到为什么使用Spring Boot, 就不得不提到Spring框架的前世今生

Spring框架由于其繁琐的配置，一度被人认为“配置地狱”，各种XML、Annotation配置混合使用，让人眼花缭乱，而且如果出错了也很难找出原因。

通过SpringMVC框架部署和发布web程序，需要和系统外服务器进行关联，操作繁琐不方便。

Spring Boot是由Spring官方推出的一个新框架，对Spring进行了高度封装，是Spring未来的发展方向。使用Spring Boot框架后，可以帮助开发者快速搭建Spring框架，也可以帮助开发者快速启动一个Web服务，无须依赖外部Servlet容器，使编码变得简单，使配置变得简单，使部署变得简单，使监控变得简单。

## 4.5 Spring 前世今生

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。

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

1. 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 式。

|  |
| --- |
| Spring 1.X 使用基本的框架类及配置文件（.xml）实现对象的声明及对象关系的整合。 org.springframework.core.io.ClassPathResource org.springframework.beans.factory.xml.XmlBeanFactory org.springframework.context.support.ClassPathXmlApplicationContext  Spring 2.X 使用注解代替配置文件中对象的声明。简化配置。 org.springframework.stereotype.@Component org.springframework.stereotype.@Controller org.springframework.stereotype.@Service org.springframework.stereotype.@Repository org.springframework.stereotype.@Scope org.springframework.beans.factory.annotation.@Autowired  Spring 3.X 使用更强大的注解完全代替配置文件。 org.springframework.context.annotation.AnnotationConfigApplicationContext org.springframework.context.annotation.@Configuration org.springframework.context.annotation.@Bean org.springframework.context.annotation.@Value org.springframework.context.annotation.@Import  Spring 4.X 使用条件注解强化之前版本的注解。 org.springframework.context.annotation.@Conditional |

## 4.6 自动创建一个SpringBoot项目

右键->new-> Spring starter Project

## 4.7手动创建一个SpringBoot 项目

1. 创建Maven项目

* Spring Tool Suite中创建Maven jar项目，无需配置web.xml文件。
* 修改项目中的pom.xml文件，设置JDK编译版本为1.8

|  |
| --- |
| <project>  ...  <build>  <plugins>  <!-- 修改maven默认的JRE编译版本，1.8代表JRE编译的版本，根据自己的安装版本选择1.7或1.8 -->  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  </configuration>  </plugin>  </plugins>  </build>  ... </project> |

* Spring Boot框架最低JDK版本要求1.6，但是Spring Boot官方公布的一些功能使用1.8性能会高很多，所以本章我们选用JDK1.8版本
* SSM基础架构中，需要生成web.xml文件，Spring Boot框架中为什么没有？

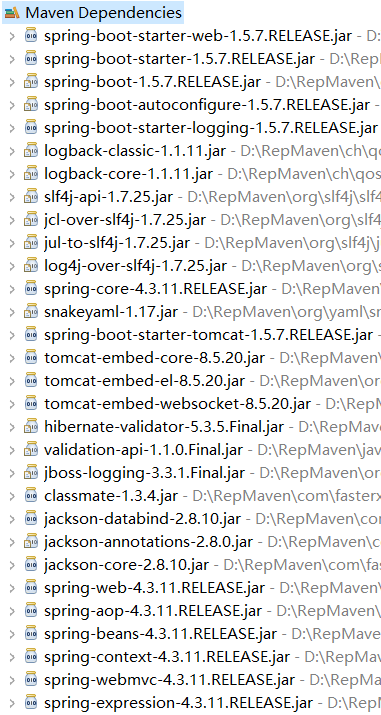
Spring Boot框架开发web系统，是基于servlet3.0或以上规范，无需web.xml文件

1. 集成Spring Boot框架

* 修改pom.xml文件，增加Spring Boot框架的依赖关系及对Web环境的支持。

|  |
| --- |
| <project>  ...  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.8.RELEASE</version>  </parent>  ...  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  </dependencies>  ... </project> |

* Spring Boot版本为官方最新正式版1.5.8.RELEASE
* 以往的项目中，所有类库的依赖关系都需要我们自己导入到pom.xml文件中，但是Spring Boot项目增加spring-boot-starter-web依赖后，会自动加载web环境配置相关依赖(SpringMVC, Tomcat)，简化了我们的操作。
* spring-boot-starter-parent：继承Spring Boot的相关参数
* spring-boot-starter-xxx：代表一个Spring Boot模块([参考附录1.Spring Boot相关模块](onenote:PN1.one" \l "参考附录1.Spring Boot相关模块&section-id={9044B196-8600-4CA5-9DCA-52597144F9F4}&page-id={A4B380C1-3C9D-4BE5-8A77-38D6E8AE3A7B}&base-path=//D:/OneNote/Teacher_atcrowdfunding_new))
* spring-boot-starter-web：代表Web模块，在这个模块中包含了许多依赖的JAR包



**附录2.项目报小红叉**

1. 增加程序代码

* 集成环境,启动服务器
* 在src/main/java目录中增加类com.atguigu.crowdfunding.AtCrowdfundingApplication，并增加相应代码。

|  |
| --- |
| package com.atguigu.crowdfunding;  import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication public class AtCrowdfundingApplication { //有网的时候可以自动创建,没有网的时候,我们可以手动创建.          public static void main(String[] args) {                 SpringApplication.run(AtCrowdfundingApplication.class, args);         }  } |

* Spring Boot项目中都会有一个以Application结尾的应用类，然后有一个标准的Java入口方法main方法。通过这个方法启动Spring Boot项目，方法中无需放入任何业务逻辑。
* @SpringBootApplication注解是Spring Boot核心注解
* 右键点击项目或项目中的AtCrowdfundingApplication类, 选择菜单Run as Spring Boot App，控制台出现下面内容表示服务启动成功。

|  |
| --- |
| . \_\_\_\_ \_ \_\_ \_ \_  /\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \ ( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \  [\\/](mhtml:file://D:\\OneNote\\20170925_JavaEE\\P9SpringBoot.mht!file:///\\\\) \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )  ' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /  =========|\_|==============|\_\_\_/=/\_/\_/\_/  :: Spring Boot :: (v1.5.8.RELEASE)  2017-08-21 14:47:42.686 INFO 7320 --- [ main] c.a.c.AtCrowdfundingApplication : Starting AtCrowdfundingApplication on DESKTOP-9UGKHC7 with PID 7320 (D:\KnowledgeHierarchy\workspace\sts-test\Web\target\classes started by 18801 in D:\KnowledgeHierarchy\workspace\sts-test\Web) 2017-08-21 14:47:42.688 INFO 7320 --- [ main] c.a.c.AtCrowdfundingApplication : No active profile set, falling back to default profiles: default 2017-08-21 14:47:42.727 INFO 7320 --- [ main] ationConfigEmbeddedWebApplicationContext : Refreshing  …  2017-08-21 14:47:44.525 INFO 7320 --- [ main] o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on startup 2017-08-21 14:47:44.586 INFO 7320 --- [ main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http) 2017-08-21 14:47:44.590 INFO 7320 --- [ main] c.a.c.AtCrowdfundingApplication : Started AtCrowdfundingApplication in 2.131 seconds (JVM running for 2.394) </java.util.map |

1. 集成了Tomcat服务器

* 当增加Web依赖后执行main方法，等同于启动Tomcat服务器, 默认端口号为8080。
* 如果想要修改默认的Tomcat服务器端口号，可以通过全局配置文件进行配置，在src/main/resources/目录中增加application.properties文件。

|  |
| --- |
| server.context-path=/ server.port=80 server.session.timeout=60 server.tomcat.max-threads=800 server.tomcat.uri-encoding=UTF-8 |

* Spring Boot会自动读取src/main/resources/路径或类路径下/config路径中的application.properties文件或application.yml文件。

1. 为什么还会有配置文件

Spring Boot我们称之为微框架，这里的“微”不是小和少的意思，而是“简”的意思，简单，简洁。

项目中大部分的基础配置由Spring Boot框架帮我们自动集成，简化了我们的配置，但是框架自身为了扩展性，依然需要提供配置文件。

上面的代码中只是简单的应用了Spring Boot框架，但是我们真正要做的是将Spring Boot应用到项目中，所以接下来我们增加对SpringMVC框架，Mybatis框架的集成。

## 4.7 SpringBoot 集成 Spring & Spring Web MVC

* 基本的Spring Boot环境已经构建好了，现在需要配置Spring框架及SpringMVC框架的业务环境

1. @ComponentScan注解

|  |
| --- |
| package com.atguigu.crowdfunding;  import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.context.annotation.ComponentScan;  **@ComponentScan(basePackages="com.atguigu")**  @SpringBootApplication public class AtCrowdfundingApplication {          public static void main(String[] args) {                 SpringApplication.run(AtCrowdfundingApplication.class, args);         }  } |

1. 默认扫描

默认扫描当前包com.atguigu.crowdfunding和子包com.atguigu.crowdfunding.\*

如果还需要扫描其他的包，那么需要增加@ComponentScan注解,指定包名进行扫描。

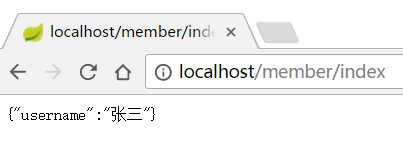
1. 增加控制器代码

在src/main/java目录中增加类com.atguigu.crowdfunding.controller.MemberController，并增加相应代码。

|  |
| --- |
| package com.atguigu.crowdfunding.controller;  import java.util.HashMap; import java.util.Map;  import org.springframework.stereotype.Controller; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.ResponseBody;  @Controller @RequestMapping("/member") public class MemberController {          @ResponseBody         @RequestMapping("/index")         public Object index() {                 Map map = new HashMap();                 map.put("username", "张三");                 return map;         } } |

1. 执行main方法启动应用

访问路径http://127.0.0.1:8080[/应用路径名称]/member/index 页面打印JSON字符串即可



1. @Controller和@RestController区别

官方文档：**@RestController** is a stereotype annotation that combines **@ResponseBody and @Controller**.

表示@RestController等同于@Controller + @ResponseBody，所以上面的代码可以变为：

|  |
| --- |
| package com.atguigu.crowdfunding.controller;  import java.util.HashMap; import java.util.Map;  import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController;  **@RestController** @RequestMapping("/member") public class MemberController {  @RequestMapping("/index") public Object index() {         Map map = new HashMap();         map.put("username", "张三");         return map; //JSON => JavaScript Object Notation  }  } |

如果响应的中文字符乱码，可以在注解RequestMapping中指定编码格式：

@RequestMapping(value=””,produces = { “application/json;charset=UTF-8”})

1. 增加服务层代码

Service接口，ServiceImpl实现类的使用和SSM架构中的使用方式完全相同。

|  |
| --- |
| package com.atguigu.crowdfunding.service;    public interface MemberService {    } |
| package com.atguigu.crowdfunding.service.impl;    import org.springframework.stereotype.Service;  import com.atguigu.crowdfunding.service.MemberService;    @Service  public class MemberServiceImpl implements MemberService {    } |
| @RestController  public class MemberController {  @Autowired  private MemberService memberService ;  } |

## 4.8 SpringBoot 集成 MyBatis

Spring Boot框架在集成Mybatis框架的时候依然要扫描Dao接口，SQL映射文件以及依赖数据库连接池，但是和传统SSM框架集成时稍微有一些不同。

1. 增加Mybatis 依赖

|  |
| --- |
| <project>  ...  <dependencies>  ...  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <!-- 数据库连接池 -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.5</version>  </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.1.1</version>  </dependency>  ...  </dependencies>  ... </project> |

1. 增加application.yml配置文件，并配置连接池和Mybatis相关配置

|  |
| --- |
| ---  spring:   datasource:  name: mydb  type: com.alibaba.druid.pool.DruidDataSource  url: jdbc:mysql://127.0.0.1:3306/atcrowdfunding  username: root  password: root  driver-class-name: com.mysql.jdbc.Driver  mybatis:  mapper-locations: classpath\*:/mybatis/mapper-\*.xml  type-aliases-package: com.atguigu.\*\*.bean |

1. Properties 和 yml 区别

在 Spring Boot 中，有两种配置文件， 一种是application.properties, 另一种是application.yml,

两种都可以配置Spring Boot 项目中的一些变量的定义，参数的设置等。

application.properties 配置文件在写的时候要写完整，yml文件在写的时候层次感强，而且少写了代码。但是从严格意义上来讲，区别不大。

1. 扫描Dao接口和开启声明式事务

需要在AtCrowdfundingApplication类中增加扫描注解

@MapperScan("com.atguigu.\*\*.dao")及事务管理@EnableTransactionManagement

1. 增加Dao代码

|  |
| --- |
| package com.atguigu.crowdfunding.dao;  import org.apache.ibatis.annotations.Insert; import org.apache.ibatis.annotations.Select;  import com.atguigu.crowdfunding.bean.Member;  public interface MemberDao {  @Select("select \* from t\_member where id = #{id}") public Member queryById(Integer id);  @Insert("insert into t\_member (loginacct) values (#{loginacct})") public int insertMember(Member member);  } |

1. 增加Member实体类

|  |
| --- |
| package com.atguigu.crowdfunding.bean;  public class Member {          public Integer id;         private String name;          public String getName() {                 return name;         }          public void setName(String name) {                 this.name= name;         }          public Integer getId() {                 return id;         }          public void setId(Integer id) {                 this.id = id;         }          } |

1. 增加事务注解 @Transactional

传统的SSM架构中采用的是声明式事务，需要在配置文件中增加AOP事务配置，Spring Boot框架中简化了这种配置，可以在Service接口中增加注解@Transactional

|  |
| --- |
| package com.atguigu.crowdfunding.service.impl;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.stereotype.Service; import org.springframework.transaction.annotation.Transactional;  import com.atguigu.crowdfunding.bean.Member; import com.atguigu.crowdfunding.dao.MemberDao; import com.atguigu.crowdfunding.service.MemberService;  @Service @Transactional(readOnly=true) public class MemberServiceImpl implements MemberService {          @Autowired         private MemberDao memberDao;                  public Member queryById(Integer id) {                 return memberDao.queryById(id);         }          @Transactional         public int insertMember(Member member) {                 return memberDao.insertMember(member);         }  } |

1. 修改MemberController进行测试

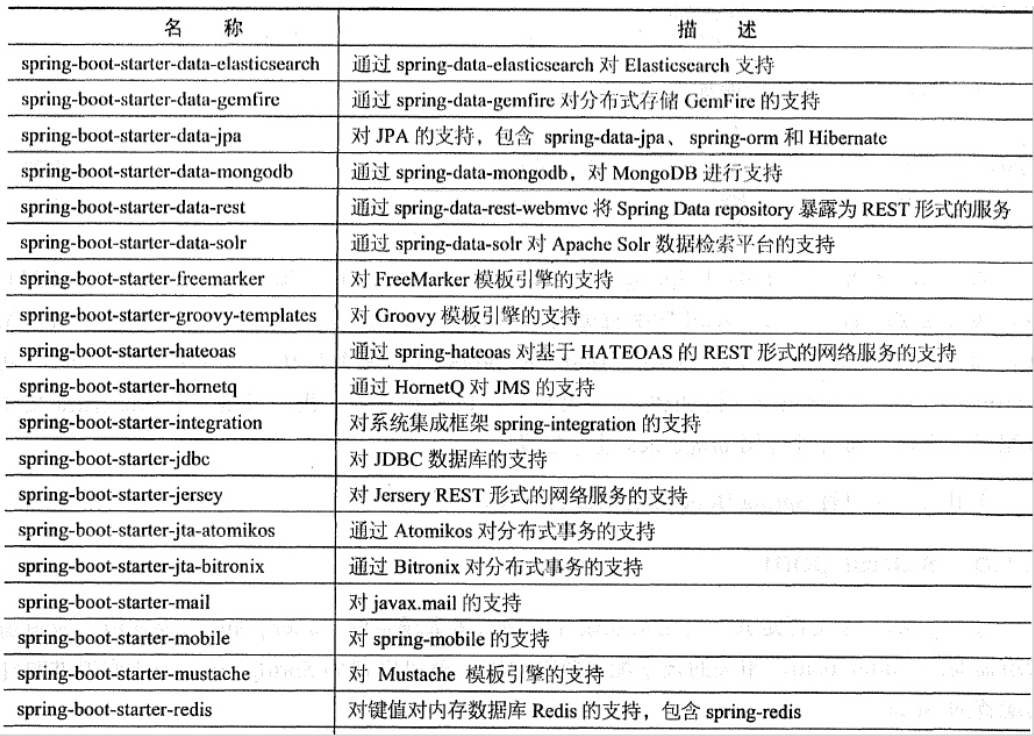
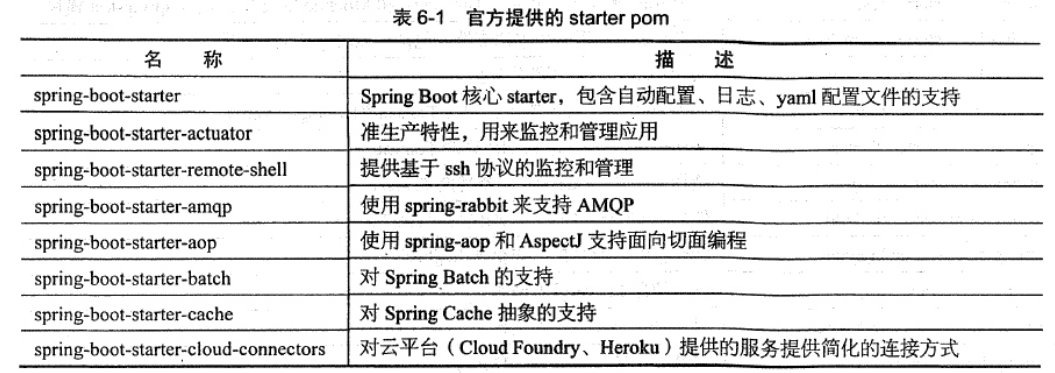
|  |
| --- |
| package com.atguigu.crowdfunding.controller; import java.util.HashMap;  import org.springframework.beans.factory.annotation.Autowired; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController; import org.springframework.transaction.annotation.EnableTransactionManagement;  import com.atguigu.crowdfunding.bean.Member; import com.atguigu.crowdfunding.service.MemberService;  @RestController @RequestMapping("/member") public class MemberController {          @Autowired         private MemberService memberService;                  @RequestMapping("/index/{id}")         public Object index(@PathVariable("id") Integer id ) {                 Member member = memberService.queryById(id);                 return member;         }           @RequestMapping("/insert")         public Object insert( Member member ) {                 memberService.insertMember(member);                 return new HashMap();         } } |

1. 测试

重启服务，访问路径http://127.0.0.1:8080[/应用路径名称]/member/index观察效果

# 附录

## 附录1 SpringBoot相关模块



## 附录2 项目报小红叉

