# 快捷键

|  |
| --- |
| Alt+insert: 插入代码 |

Idea设置编码

|  |
| --- |
| 防止console打印出乱码 |

# 简介

|  |
| --- |
| 使用嵌入式的Servlet容器，应用无需打成WAR包 |

# 打包及导入

|  |
| --- |
| java -jar 包名    重新导入 |

# myEclipse configuration working sets

|  |
| --- |
| <https://jingyan.baidu.com/article/ab0b5630609e85c15bfa7d10.html> |

# @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，可理解为用spring的时候xml里面的<beans>标签；  2、@Bean 作用于方法上，相当于xml配置中的<bean>；  @Bean给容器添加一个组件 |

# @Component 注册组件 取值@Autowired

|  |
| --- |
|  |

# @Controller、@ResponseBody、@RequestMapping("/hello")

|  |
| --- |
| //这个类的所有方法返回的数据直接写给浏览器，（如果是对象转为json数据）  /\*  @ResponseBody  @Controller\*/  @RestController  public class HelloController {  @RequestMapping("/hello")  public String hello(){  return "hello world quick!";  }  // RESTAPI的方式  } |

# 什么是Spring boot

|  |
| --- |
| 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。 |

# Spring Boot特性

|  |
| --- |
| 1. 创建独立的Spring应用程序 2. 嵌入的Tomcat，无需部署WAR文件 3. 简化Maven配置 4. 自动配置Spring 5. 提供生产就绪型功能，如指标，健康检查和外部配置 6.开箱即用，没有代码生成，也无需XML配置。 |

# Spring Boot特性理解

|  |
| --- |
| 1.为基于Spring的开发提供更快的入门体验 2.开箱即用，没有代码生成，也无需XML配置。同时也可以修改默认值来满足特定的需求。 3.提供了一些大型项目中常见的非功能特性，如嵌入式服务器、安全、指标，健康检测、外部配置等。 4.Spring Boot并不是对Spring功能上的增强，而是提供了一种快速使用Spring的方式。 |

# 开发准备

|  |
| --- |
| 开发环境JDK 1.8 开发工具(Eclipse) 项目管理工具( Maven) |

# idea创建springbooot

|  |
| --- |
| <https://blog.csdn.net/qq_34205356/article/details/81098354> |

# pom.xml

|  |
| --- |
| <modelVersion>4.0.0</modelVersion>  <groupId>com.kfit</groupId> <artifactId>spring-boot-hello</artifactId> <version>0.0.1-SNAPSHOT</version> <packaging>jar</packaging>  <name>spring-boot-hello</name> <url>http://maven.apache.org</url>  <!-- spring boot 父节点依赖,引入这个之后相关的引入就不需要添加version配置，spring boot会自动选择最合适的版本进行添加。  --> <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.4.1.RELEASE</version> </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <!-- 指定一下jdk的版本 ，这里我们使用jdk 1.8 ,默认是1.6 -->  <java.version>1.8</java.version> </properties>  <dependencies>  <!--spring-boot-starter-web: MVC,AOP的依赖包....-->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <!--  <version></version>  由于我们在上面指定了 parent(spring boot)-->  </dependency>   <!-- 添加fastjson 依赖包. -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.15</version>  </dependency>   <!-- spring boot devtools 依赖包. -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <optional>true</optional>  <scope>true</scope>  </dependency> </dependencies>  <!-- 构建节点. --> <build>  <plugins>  <!-- 在这里添加springloader plugin -->  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin </artifactId>  <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>springloaded</artifactId>  <version>1.2.4.RELEASE</version>  </dependency>  </dependencies>  <executions>  <execution>  <goals>  <goal>repackage</goal>  </goals>  <configuration>  <classifier>exec</classifier>  </configuration>  </execution>  </executions>  </plugin>   <!-- 这是spring boot devtool plugin -->  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <!--fork : 如果没有该项配置，肯呢个devtools不会起作用，即应用不会restart -->  <fork>true</fork>  </configuration>  </plugin>  </plugins> </build> |

# fastjson

|  |
| --- |
| spring boot有自己的json框架(jackson)，也可以集成其他json框架。  1.引入fastjson  方文档说的1.2.10以后，会有两个方法支HttpMessageconvert，  一个是FastJsonHttpMessageConverter，支持4.2以下的版本，  一个是FastJsonHttpMessageConverter4支持4.2以上的版本，  具体有什么区别暂时没有深入研究。这里也就是说：低版本的就不支持了，所以这里最低要求就是1.2.10+。  <dependency> <groupId>com.alibaba</groupId> <artifactId>fastjson</artifactId> <version>1.2.15</version>  </dependency>  2.配置fastjon(支持两种方法)  第一种方法就是：  （1）启动类继承extends WebMvcConfigurerAdapter  （2）覆盖方法configureMessageConverters  第二种方法  （1）在App.java启动类中，  注入Bean : HttpMessageConverters |

## 第一种方法

Demo

|  |
| --- |
| package com.kfit;  import java.util.Date;  import com.alibaba.fastjson.annotation.JSONField;  public class Demo {    private *int* id;    private *String* name;      //com.alibaba.fastjson.annotation.JSONField    @*JSONField*(format="yyyy-MM-dd HH:mm")    private *Date* createTime;    /\*     \* 我们不想返回remarks?     \* serialize:是否需要序列化属性.     \*/    @*JSONField*(serialize=false)    private *String* remarks;      public *String* getRemarks() {      return remarks;    }    public *void* setRemarks(*String* *remarks*) {      this.remarks = remarks;    }    public *Date* getCreateTime() {      return createTime;    }    public *void* setCreateTime(*Date* *createTime*) {      this.createTime = createTime;    }    public *int* getId() {      return id;    }    public *void* setId(*int* *id*) {      this.id = id;    }    public *String* getName() {      return name;    }    public *void* setName(*String* *name*) {      this.name = name;    }  } |

HelloController

|  |
| --- |
| package com.kfit;  import java.util.Date;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  @*RestController*  public class HelloController {    /\*\*     \* Spring Boot默认使用的json解析框架是jackson     \* @return     \*/    @*RequestMapping*("/getDemo")    public *Demo* getDemo(){  *Demo* demo = new Demo();      demo.setId(1);      demo.setName("张三");      demo.setCreateTime(new Date());      demo.setRemarks("这是备注信息");      return demo;    }  } |

App

|  |
| --- |
| package com.kfit;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.boot.autoconfigure.web.HttpMessageConverters;  import org.springframework.context.annotation.Bean;  import org.springframework.http.converter.HttpMessageConverter;  import com.alibaba.fastjson.serializer.SerializerFeature;  import com.alibaba.fastjson.support.config.FastJsonConfig;  import com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter;  /\*\*@SpringBootApplication指定这是一个 spring boot的应用程序. \*/  @*SpringBootApplication*  public class App extends *WebMvcConfigurerAdapter* {      @*Override*    public *void* configureMessageConverters(*List*<*HttpMessageConverter*<*?*>> *converters*) {      super.configureMessageConverters(converters);        // 1、需要先定义一个 convert 转换消息的对象;  *FastJsonHttpMessageConverter* fastConverter = new FastJsonHttpMessageConverter();      //2、添加fastJson 的配置信息，比如：是否要格式化返回的json数据;  *FastJsonConfig* fastJsonConfig = new FastJsonConfig();  fastJsonConfig.setSerializerFeatures(    //是否需要格式化                 SerializerFeature.PrettyFormat         );         //3、在convert中添加配置信息.       fastConverter.setFastJsonConfig(fastJsonConfig);         //4、将convert添加到converters当中.       converters.add(fastConverter);    }      public static *void* main(*String*[] *args*) {      SpringApplication.run(App.class, args);    }  } |

## 第二种方法

App

|  |
| --- |
| package com.kfit;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.boot.autoconfigure.web.HttpMessageConverters;  import org.springframework.context.annotation.Bean;  import org.springframework.http.converter.HttpMessageConverter;  import com.alibaba.fastjson.serializer.SerializerFeature;  import com.alibaba.fastjson.support.config.FastJsonConfig;  import com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter;  @*SpringBootApplication*  public class App {      /\*\*     \* 在这里我们使用 @Bean注入 fastJsonHttpMessageConvert     \* @return     \*/    @*Bean*    public *HttpMessageConverters* fastJsonHttpMessageConverters() {      // 1、需要先定义一个 convert 转换消息的对象;  *FastJsonHttpMessageConverter* fastConverter = new FastJsonHttpMessageConverter();      //2、添加fastJson 的配置信息，比如：是否要格式化返回的json数据;  *FastJsonConfig* fastJsonConfig = new FastJsonConfig();      fastJsonConfig.setSerializerFeatures(SerializerFeature.PrettyFormat);        //3、在convert中添加配置信息.      fastConverter.setFastJsonConfig(fastJsonConfig);        //4、将convert添加到converters当中.  *HttpMessageConverter*<*?*> converter = fastConverter;      return new HttpMessageConverters(converter);    }      /\*\*     \* 这是springloader的配置方式：-javaagent:.\lib\springloaded-1.2.4.RELEASE.jar -noverify     \* @param *args*     \*/    public static *void* main(*String*[] *args*) {      SpringApplication.run(App.class, args);    }  } |

# springloader热部署

<https://blog.csdn.net/chachapaofan/article/details/88697452>

|  |
| --- |
| 1. 当我们修改了方法的返回值，是能够进行热部署的； 2、当我们重新创建了一个方法，是能够进行热部署的； 3、当我们重新创建了一个Class,是能够进行热部署的；   使用方式:  1.在pom.xml文件添加依赖包： <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin </artifactId> <dependencies>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>springloaded</artifactId>  <version>1.2.4.RELEASE</version>  </dependency> </dependencies> <executions>  <execution>  <goals>  <goal>repackage</goal>  </goals>  <configuration>  <classifier>exec</classifier>  </configuration>  </execution> </executions> </plugin>  运行方法一：使用spring-boot:run    运行方法二：  1.如果使用的run as – java application的话，那么还需要做一些处理。  2.把spring-loader-1.2.4.RELEASE.jar下载下来，放到项目的lib目录中，然后把IDEA的run参数里VM参数设置为：  3.-javaagent:.\lib\springloaded-1.2.4.RELEASE.jar -noverify  4.然后启动就可以了，这样在run as的时候，也能进行热部署    端口被占用，要关闭改端口  添加lib |

# devtools热部署

|  |
| --- |
| 问题的提出：通过使用springloaded进行热部署，但是些代码修改了，并不会进行热部署。  1.spring-boot-devtools 是一个为开发者服务的一个模块，其中最重要的功能就是自动应用代码更改到最新的App上面去。原理是在发现代码有更改之后，重新启动应用，但是速度比手动停止后再启动还要更快，更快指的不是节省出来的手工操作的时间。  2.其深层原理是使用了两个ClassLoader，一个Classloader加载那些不会改变的类（第三方Jar包），另一个ClassLoader加载会更改的类，称为 restart ClassLoader  ,这样在有代码更改的时候，原来的restart ClassLoader 被丢弃，重新创建一个restart ClassLoader，由于需要加载的类相比较少，所以实现了较快的重启时间（5秒以内）。  添加依赖包： <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <optional>true</optional>  <scope>true</scope> </dependency>  添加spring-boot-maven-plugin： <build> <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <!--fork : 如果没有该项配置，肯呢个devtools不会起作用，即应用不会restart -->  <fork>true</fork>  </configuration>  </plugin> </plugins> </build>  说明：  1. devtools会监听classpath下的文件变动，并且会立即重启应用（发生在保存时机），注意：因为其采用的虚拟机机制，该项重启是很快的。  2. devtools可以实现页面热部署（即页面修改后会立即生效，这个可以直接在application.properties文件中配置spring.thymeleaf.cache=false来实现(这里注意不同的模板配置不一样)  测试方法：  修改类-->保存：应用会重启  修改配置文件-->保存：应用会重启  修改页面-->保存：应用会重启，页面会刷新（原理是将spring.thymeleaf.cache设为false）  不能使用分析：  1.对应的spring-boot版本是否正确，这里使用的是1.4.1版本；  2.是否加入plugin以及属性<fork>true</fork>  3.Eclipse Project 是否开启了Build Automatically（我自己就在这里栽了坑，不知道为什么我的工具什么时候关闭了自动编译的功能）。  4.如果设置SpringApplication.setRegisterShutdownHook(false)，则自动重启将不起作用。 |

# SpringApplication

|  |
| --- |
| package com.shipeng.springboot01;  import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication;  // @SpringBootApplication指定这是一个 spring boot的应用程序. @SpringBootApplication public class Springboot01Application {   public static void main(String[] args) {  SpringApplication.run(Springboot01Application.class, args);  }  } |

# @RestController （等待于 @Controller 和 @RequestBody）

|  |
| --- |
| package com.shipeng.springboot01;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.ResponseBody;  import org.springframework.web.bind.annotation.RestController;  /\*\*   \* 在这里我们使用@RestController  （等待于 @Controller 和 @ResponseBody）   \*/  @*RestController*  public class HelloController {      /\*\*     \* 在这里我们使用@RequestMapping 建立请求映射:     \* http://127.0.0.1:8080/hello     \* @return     \*/    @*RequestMapping*("/hello")    public *String* hello(){      return "hello";    }  } |

# JPA/Hibernate/Spring Data概念

## 什么是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。  在上面只是一个JPA的定义，我们看看另外一段更能看出是什么的描述：  JPA(Java Persistence API)是Sun官方提出的Java持久化规范。它为Java开发人员提供了一种对象/关系映射工具来管理Java应用中的关系数据。  持久化（Persistence），即把数据（如内存中的对象）保存到可永久保存的存储设备中（如磁盘）。持久化的主要应用是将内存中的对象存储在的数据库中，或者存储在磁盘文件中、XML数据文件中等等。  持久化是将程序数据在持久状态和瞬时状态间转换的机制。  JDBC就是一种持久化机制。文件IO也是一种持久化机制。  “规范”： 所谓的规范意指明文规定或约定俗成的标准。如：道德规范、技术规范，公司管理规范。  那么“持久化规范”就是Sun针对持久化这一层操作指定的规范，如果没有指定JPA规范，那么新起的框架就随意按照自己的标准来了，那我们开发人员就没法把我们的经历全部集中在我们的业务层上，而是在想如何进行兼容，这种情况有点像Android开发，Android本身有官方的SDK,但是由于SDK过于开源了，结果导致很多厂商基于SDK二次开发，但是兼容性就不是很好，最好的例子就是Android的头像上传，就是一件很烦人的事情。好了，JPA就介绍到这里。 |

## 什么是Hibernate?

|  |
| --- |
| 这里引用百度百科的话hibernate：  Hibernate是一个开放源代码的对象关系映射框架，它对JDBC进行了非常轻量级的对象封装，它将POJO与数据库表建立映射关系，是一个全自动的orm框架，hibernate可以自动生成SQL语句，自动执行，使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。 Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序使用，也可以在Servlet/JSP的Web应用中使用，最具革命意义的是，Hibernate可以在应用EJB的J2EE架构中取代CMP，完成数据持久化的重任。  那么是ORM呢? ORM是对象关系映射的意思，英语：Object Relational Mapping简称ORM，是一种程序技术，用于实现面向对象编程语言里不同系统类型的系统之间的数据转换。好了，更多的概念需要自己去挖掘，这里只是抛装引玉下。 |

## 什么是Spring Data?

|  |
| --- |
| Spring Data是一个用于简化数据库访问，并支持云服务的开源框架。其主要目标是使得数据库的访问变得方便快捷，并支持map-reduce框架和云计算数据服务。此外，它还支持基于关系型数据库的数据服务，如Oracle RAC等。对于拥有海量数据的项目，可以用Spring Data来简化项目的开发，就如Spring Framework对JDBC、ORM的支持一样，Spring Data会让数据的访问变得更加方便。 |

## 什么是Spring Data JPA?

|  |
| --- |
| 我们先看一个描述： Spring Data JPA能干什么  可以极大的简化JPA的写法，可以在几乎不用写实现的情况下，实现对数据的访问和操作。除了CRUD外，还包括如分页、排序等一些常用的功能。  首先我们需要清楚的是Spring Data是一个开源框架，在这个框架中Spring Data JPA只是这个框架中的一个模块，所以名称才叫Spring Data JPA。如果单独使用JPA开发，你会发现这个代码量和使用JDBC开发一样有点烦人，所以Spring Data JPA的出现就是为了简化JPA的写法，让你只需要编写一个接口继承一个类就能实现CRUD操作了。 |

## JPA/Hibernate 关系？

|  |
| --- |
| 我们先看下别人的描述：  JPA是一种规范，而Hibernate是它的一种实现。除了Hibernate，还有EclipseLink(曾经的toplink)，OpenJPA等可供选择，所以使用Jpa的一个好处是，可以更换实现而不必改动太多代码。 |

# JPA-Hibernate

|  |
| --- |
| 1.在pom.xml添加mysql,spring-data-jpa依赖  <!-- 添加MySQL数据库驱动依赖包. -->  <dependency>     <groupId>mysql</groupId>     <artifactId>mysql-connector-java</artifactId>  </dependency>    <!-- 添加Spring-data-jpa依赖. -->  <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  -----------------------------------------  2.在application.properties文件中配置mysql连接配置文件  ######################################################## ###datasource ######################################################## spring.datasource.url = jdbc:mysql://localhost:3306/test spring.datasource.username = root spring.datasource.password = root spring.datasource.driverClassName = com.mysql.jdbc.Driver spring.datasource.max-active=20 spring.datasource.max-idle=8 spring.datasource.min-idle=8 spring.datasource.initial-size=10  -----------------------------------------  3.在application.properties文件中配置JPA配置信息  ######################################################## ### Java Persistence Api ######################################################## # Specify the DBMS spring.jpa.database = MYSQL # Show or not log for each sql query spring.jpa.show-sql = true # Hibernate ddl auto (create, create-drop, update) spring.jpa.hibernate.ddl-auto = update # Naming strategy #[org.hibernate.cfg.ImprovedNamingStrategy #org.hibernate.cfg.DefaultNamingStrategy] spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy # stripped before adding them to the entity manager) spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect  4.编写测试例子 |

### 编写测试例子

|  |
| --- |
| (1) 创建实体类Demo,如果已经存在，可以忽略。  (2) 创建jpa repository类操作持久化(CrudRepository)。  (3) 创建service类。  (4) 创建restful请求类。  (5) 测试;  *Spring* *Boot* JPA 总结  ---------------------  1、需要添加相应的依赖包;  2、需要在application.properties文件添加配置信息；  3、需要创建一个实体类,比如Cat;  4、需要创建一个接口继承CrudRepository;  5、需要创建一个Service;  6、需要创建一个Controller;  7、代码测试；访问链接 http://localhost:8081/save |

bean

|  |
| --- |
| package com.kfit.demo.bean;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  /\*\*   \* 创建了一个实体类。   \*   \* 如何持久化呢？   \*   \* 1、使用@Entity进行实体类的持久化操作，当JPA检测到我们的实体类当中有   \*   \* @Entity 注解的时候，会在数据库中生成对应的表结构信息。   \*   \* 如何指定主键以及主键的生成策略？   \*   \* 2、使用@Id指定主键.   \*/  @*Entity*  public class Cat {      /\*\*     \* 使用@Id指定主键.     \*     \* 使用代码@GeneratedValue(strategy=GenerationType.AUTO)     \* 指定主键的生成策略，mysql默认的是自增长。     \*     \*/    @*Id* @*GeneratedValue*(strategy=GenerationType.AUTO)    private *int* id;//主键.      private *String* catName;//姓名. cat\_name      private *int* catAge;//年龄. cat\_age;    public *int* getId() {      return id;    }    public *void* setId(*int* *id*) {      this.id = id;    }    public *String* getCatName() {      return catName;    }    public *void* setCatName(*String* *catName*) {      this.catName = catName;    }    public *int* getCatAge() {      return catAge;    }    public *void* setCatAge(*int* *catAge*) {      this.catAge = catAge;    }    } |

repository

|  |
| --- |
| package com.kfit.demo.repository;  import org.springframework.data.repository.CrudRepository;  import com.kfit.demo.bean.Cat;  public interface CatRepository extends *CrudRepository*<*Cat*, *Integer*>{  } |

service

|  |
| --- |
| package com.kfit.demo.service;  import javax.annotation.Resource;  import javax.transaction.Transactional;  import org.springframework.stereotype.Service;  import com.kfit.demo.bean.Cat;  import com.kfit.demo.repository.CatRepository;  @*Service*  public class CatService {      @*Resource*    private *CatRepository* catRepository;      /\*\*     \* save,update ,delete 方法需要绑定事务.     \*     \* 使用@Transactional进行事务的绑定.     \*     \* @param *cat*     \*/      //保存数据.    @*Transactional*    public *void* save(*Cat* *cat*){      catRepository.save(cat);    }      //删除数据》    @*Transactional*    public *void* delete(*int* *id*){      catRepository.delete(id);    }      //查询数据.    public *Iterable*<*Cat*> getAll(){      return catRepository.findAll();    }  } |

controller

|  |
| --- |
| package com.kfit.demo.controller;  import javax.annotation.Resource;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  import com.kfit.demo.bean.Cat;  import com.kfit.demo.service.CatService;  @*RestController*  @*RequestMapping*("/cat")  public class CatController {      @*Resource*    private *CatService* catService;      @*RequestMapping*("/save")    public *String* save(){  *Cat* cat = new Cat();      cat.setCatName("jack");      cat.setCatAge(3);      catService.save(cat);      return "save ok.";    }      @*RequestMapping*("/delete")    public *String* delete(){      catService.delete(1);      return "delete ok";    }      @*RequestMapping*("/getAll")    public *Iterable*<*Cat*> getAll(){      return catService.getAll();    }    } |

-------------------------

repository

|  |
| --- |
| package com.kfit.demo.repository;  import org.springframework.data.jpa.repository.Query;  import org.springframework.data.repository.PagingAndSortingRepository;  import org.springframework.data.repository.query.Param;  import com.kfit.demo.bean.Cat;  public interface Cat2Repository extends *PagingAndSortingRepository*<*Cat*, *Integer*>{    /\*\*     \* 1/ 查询方法 以 get | find | read 开头.     \* 2/ 涉及查询条件时，条件的属性用条件关键字连接，要注意的是条件属性以首字母大写。     \*/      //根据catName进行查询 : 根据catName进行查询.    public *Cat* findByCatName(*String* *catName*);      /\*\*     \* 如何编写JPQL语句，     \* Hibernate  -- HQL语句.     \* JPQL 语句 和HQL语句是类似的.     \*/    @*Query*("from Cat where catName=:cn")    public *Cat* findMyCatName(@*Param*("cn")*String* *catName*);  } |

controller

|  |
| --- |
| package com.kfit.demo.controller;  import javax.annotation.Resource;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  import com.kfit.demo.bean.Cat;  import com.kfit.demo.service.CatService;  @*RestController*  @*RequestMapping*("/cat")  public class CatController {      @*Resource*    private *CatService* catService;      @*RequestMapping*("/save")    public *String* save(){  *Cat* cat = new Cat();      cat.setCatName("jack");      cat.setCatAge(3);      catService.save(cat);      return "save ok.";    }      @*RequestMapping*("/delete")    public *String* delete(){      catService.delete(1);      return "delete ok";    }      @*RequestMapping*("/getAll")    public *Iterable*<*Cat*> getAll(){      return catService.getAll();    }      @*RequestMapping*("/findByCatName")    public *Cat* findByCatName(*String* *catName*){      return catService.findByCatName(catName);    }      @*RequestMapping*("/findByCatName2")    public *Cat* findByCatName2(*String* *catName*){      System.out.println("CatController.findByCatName2()");      return catService.findByCatName2(catName);      }      @*RequestMapping*("/selectByCatName")    public *Cat* selectByCatName(*String* *catName*){      return catService.selectByCatName(catName);    }  } |

# Spring Data JPA介绍

|  |
| --- |
| Spring Data 的一个核心接口为我们提供了常用的接口，在这节我们就来简单介绍下。  Repository接口  Repository 接口是 Spring Data 的一个核心接口，它不提供任何方法，开发者需要在自己定义的接口中声明需要的方法 ：  public interface Repository<T, ID extends Serializable> { }  有这么几点需要强调下：  1. Repository是一个空接口，即是一个标记接口；  2. 若我们定义的接口继承了Repository，则该接口会被IOC容器识别为一个Repository Bean纳入到IOC容器中，进而可以在该接口中定义满足一定规范的方法。  3. 实际上也可以通过@RepositoryDefinition,注解来替代继承Repository接口。  4. 查询方法以find | read | get开头；  5. 涉及查询条件时，条件的属性用条件关键字连接，要注意的是条件属性以首字母大写。  6.使用@Query注解可以自定义JPQL语句实现更灵活的查询。  CrudRepository接口  CrudRepository 接口提供了最基本的对实体类的添删改查操作  --T save(T entity);//保存单个实体  --Iterable<T> save(Iterable<? extends T> entities);//保存集合  --T findOne(ID id);//根据id查找实体  --boolean exists(ID id);//根据id判断实体是否存在  --Iterable<T> findAll();//查询所有实体,不用或慎用!  --long count();//查询实体数量  --void delete(ID id);//根据Id删除实体  --void delete(T entity);//删除一个实体  --void delete(Iterable<? extends T> entities);//删除一个实体的集合  --void deleteAll();//删除所有实体,不用或慎用!  PagingAndSortingRepository接口  该接口提供了分页与排序功能  --Iterable<T> findAll(Sort sort); //排序  --Page<T> findAll(Pageable pageable); //分页查询（含排序功能）  其它接口  JpaRepository：查找所有实体，排序、查找所有实体，执行缓存与数据库同步  JpaSpecificationExecutor：不属于Repository体系，实现一组 JPA Criteria 查询相关的方法，封装 JPA Criteria 查询条件。通常使用匿名内部类的方式来创建该接口的对象。  自定义 Repository：可以自己定义一个MyRepository接口。 |

# JdbcTemplate

|  |
| --- |
| (1)在pom.xml加入jdbcTemplate的依赖；  (2)编写DemoDao类，声明为：@Repository，引入JdbcTemplate  (3)编写DemoService类，引入DemoDao进行使用  (4)编写Demo2Controller进行简单测试。  在pom.xml加入jdbcTemplate的依赖  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId> </dependency>  如果在JPA已经加入的话，则可以不用引入以上的配置。 <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId> </dependency>  --------------------------------------  那么只需要在需要使用的类中加入： @Resource private JdbcTemplate jdbcTemplate;  --------------------------------------  编写DemoDao类  声明为：@Repository，引入JdbcTemplate public Demo getById(long id) {  String sql = "select \*from Demo where id=?";  RowMapper<Demo> rowMapper = new BeanPropertyRowMapper<Demo>(Demo.class);  return jdbcTemplate.queryForObject(sql, rowMapper,id); }  --------------------------------------  编写DemoService类，引入DemoDao进行使用 @Resource private DemoDao demoDao;  public void save(Demo demo){  demoDao.save(demo); }  --------------------------------------  编写Demo2Controller进行简单测试  @Resource private DemoService demoService;  @RequestMapping("/getById") public Demo getById(long id){  return demoService.getById(id); } |

dao

|  |
| --- |
| package com.kfit.demo.dao;  import javax.annotation.Resource;  import org.springframework.jdbc.core.BeanPropertyRowMapper;  import org.springframework.jdbc.core.JdbcTemplate;  import org.springframework.jdbc.core.RowMapper;  import org.springframework.stereotype.Repository;  import com.kfit.demo.bean.Cat;  /\*\*   \* 使用@Repository注解，标注这是一个持久化操作对象.   \* @author Angel -- 守护天使   \* @version v.0.1   \* @date 2016年12月18日   \*/  @*Repository*  public class CatDao {      @*Resource*    private *JdbcTemplate* jdbcTemplate;      public *Cat* selectByCatName(*String* *catName*){        /\*\*       \* 1、定义一个Sql语句；       \* 2、定义一个RowMapper.       \* 3、执行查询方法.       \*/  *String* sql = "select \*from cat where cat\_name=?";  *RowMapper*<*Cat*> rowMapper = new *BeanPropertyRowMapper*<>(Cat.class);  *Cat* cat = jdbcTemplate.queryForObject(sql, new *Object*[]{catName}, rowMapper);        return cat;    }    } |

# 全局异常捕捉

|  |
| --- |
| 在一个项目中的异常我们我们都会统一进行处理的，那么如何进行统一进行处理呢？  新建一个类GlobalDefaultExceptionHandler，  在class注解上@ControllerAdvice,  在方法上注解上@ExceptionHandler(value = Exception.class)，具体代码如下：  @ControllerAdvice public class GlobalDefaultExceptionHandler {   @ExceptionHandler(value = Exception.class)  public void defaultErrorHandler(HttpServletRequest req, Exception e) {  } }  -----------  package com.kfit.config;  import javax.servlet.http.HttpServletRequest;  import org.springframework.web.bind.annotation.ControllerAdvice;  import org.springframework.web.bind.annotation.ExceptionHandler;  import org.springframework.web.bind.annotation.ResponseBody;  /\*\*   \* 1、新建一个Class,这里取名为GlobalDefaultExceptionHandler   \* 2、在class上添加注解，@ControllerAdvice;   \* 3、在class中添加一个方法   \* 4、在方法上添加@ExcetionHandler拦截相应的异常信息；   \* 5、如果返回的是View -- 方法的返回值是ModelAndView;   \* 6、如果返回的是String或者是Json数据，那么需要在方法上添加@ResponseBody注解.   \*   \*   \* @author Angel -- 守护天使   \* @version v.0.1   \* @date 2016年12月18日   \*/  @*ControllerAdvice*  public class GlobalDefaultExceptionHandler {    @*ExceptionHandler*(Exception.class)    @*ResponseBody*    public *String* defaultExceptionHandler(*HttpServletRequest* *req*,*Exception* *e*){      //是返回的String.        //ModelAndView -- 介绍 模板引擎...?  //    ModelAndView mv = new ModelAndView();  //    mv.setViewName(viewName);        return "对不起，服务器繁忙，请稍后再试！";    }  } |

# Hello World访问404

|  |
| --- |
| （1）404 -- 确定地址是否输入正确，，此路径非彼路径 （2）404 -- 是否用对注解，此注解非彼注解 （3）404 -- 包路径是否正确，此包非彼包  （4）404 -- 确认类包是否正确，此类包非彼类包 |

# 配置server信息

|  |
| --- |
| spring boot返回json数据  1/修改端口号  2/修改context-path  3/其它配置说明  Spring boot 默认端口是8080，如果想要进行更改的话，  只需要修改applicatoin.properties文件，在配置文件中加入：  server.port=8081  配置context-path  在application.properties进行配置：  server.context-path=/spring-boot  访问地址就是http://ip:port/spring-boot  其它server配置  #server.port=8080  #server.address= # bind to a specific NIC  #server.session-timeout= # session timeout in seconds  #the context path, defaults to '/'  #server.context-path=/spring-boot  #server.servlet-path= # the servlet path, defaults to '/'  #server.tomcat.access-log-pattern= # log pattern of the access log  #server.tomcat.access-log-enabled=false # is access logging enabled  #server.tomcat.protocol-header=x-forwarded-proto # ssl forward headers  #server.tomcat.remote-ip-header=x-forwarded-for  #server.tomcat.basedir=/tmp # base dir (usually not needed, defaults to tmp)  #server.tomcat.background-processor-delay=30; # in seconds  #server.tomcat.max-threads = 0 # number of threads in protocol handler  #server.tomcat.uri-encoding = UTF-8 # character encoding to use for URL decoding |

# thymeleaf

|  |
| --- |
| （1）在pom.xml中引入thymeleaf;  （2）如何关闭thymeleaf缓存  （3）编写模板文件.html  （4）编写访问模板文件controller  在pom.xml加入thymeleaf的依赖： <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-thymeleaf</artifactId> </dependency>  如何关闭thymeleaf缓存:  ######################################################## ###THYMELEAF (ThymeleafAutoConfiguration) ######################################################## #spring.thymeleaf.prefix=classpath:/templates/ #spring.thymeleaf.suffix=.html #spring.thymeleaf.mode=HTML5 #spring.thymeleaf.encoding=UTF-8 # ;charset=<encoding> is added #spring.thymeleaf.content-type=text/html # set to false for hot refresh spring.thymeleaf.cache=false  编写模板文件.html  编写模板文件src/main/resouces/templates/hello.html:  <!DOCTYPE html>  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">  <head>  <title>Hello World!</title>  </head>  <body>  <h1 th:inline="text">Hello.v.2</h1>  <p th:text="${hello}"></p>  </body>  </html>  编写访问模板文件controller  @Controller public class TemplateController {  /\*\*  \* 返回html模板.  \*/  @RequestMapping("/helloHtml")  public String helloHtml(Map<String,Object> map){  map.put("hello","from TemplateController.helloHtml");  return "/helloHtml";  } } |

# freemarker

|  |
| --- |
| （1）在pom.xml中引入freemarker;  （2）如何关闭freemarker缓存  （3）编写模板文件.ftl  （4）编写访问文件的controller  在pom.xml中引入freemarker  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-freemarker</artifactId> </dependency>  如何关闭freemarker缓存  ########################################################  ###FREEMARKER (FreeMarkerAutoConfiguration)  ########################################################  spring.freemarker.allow-request-override=false  spring.freemarker.cache=true  spring.freemarker.check-template-location=true  spring.freemarker.charset=UTF-8  spring.freemarker.content-type=text/html  spring.freemarker.expose-request-attributes=false  spring.freemarker.expose-session-attributes=false  spring.freemarker.expose-spring-macro-helpers=false  #spring.freemarker.prefix=  #spring.freemarker.request-context-attribute=  #spring.freemarker.settings.\*=  #spring.freemarker.suffix=.ftl  #spring.freemarker.template-loader-path=classpath:/templates/ #comma-separated list  #spring.freemarker.view-names= # whitelist of view names that can be resolved  编写模板文件.ftl  <!DOCTYPE html>  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"  xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">  <head>  <title>Hello World!</title>  </head>  <body>  <h1>Hello.v.2</h1>  <p>${hello}</p>  </body>  </html>  编写访问文件的controller  @RequestMapping("/helloFtl") public String helloFtl(Map<String,Object> map){  map.put("hello","from TemplateController.helloFtl");  return "/helloFtl"; } |

# JSP支持

|  |
| --- |
| （1） 创建Maven web project；  （2） 在pom.xml文件添加依赖；  （3） 配置application.properties支持jsp  （4） 编写测试Controller  （5） 编写JSP页面  （6） 编写启动类App.java  使用Eclipse新建一个Maven Web Project ，项目取名为：  spring-boot-jsp  在pom.xml文件添加依赖  <!-- spring boot parent节点，引入这个之后，在下面和spring boot相关的就不需要引入版本了; --> <parent> <groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-parent</artifactId> <version>1.4.1.RELEASE</version> </parent>  <!-- 指定一下jdk的版本 ，这里我们使用jdk 1.8 ,默认是1.6 --> <java.version>1.8</java.version>  ------------------------  <!-- web支持: 1、web mvc; 2、restful; 3、jackjson支持; 4、aop ........ --> <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId> </dependency>  <!-- servlet 依赖. --> <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <scope>provided</scope> </dependency>  --------------------------  JSTL（JSP Standard Tag Library，JSP标准标签库)是一个不断完善的开放源代码的JSP标签库，是由apache的jakarta小组来维护的。 <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId> </dependency>  <!-- tomcat 的支持.--> <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  <scope>provided</scope> </dependency> <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  <scope>provided</scope> </dependency>  --------------------------  application.properties配置  添加src/main/resources/application.properties：  # 页面默认前缀目录 spring.mvc.view.prefix=/WEB-INF/jsp/ # 响应页面默认后缀 spring.mvc.view.suffix=.jsp # 自定义属性，可以在Controller中读取 application.hello=Hello Angel From application  --------------------------  编写测试Controller  @Controller public class HelloController {  private String hello;   @RequestMapping("/helloJsp")  public String helloJsp(Map<String,Object> map){  System.***out***.println("HelloController.helloJsp().hello=hello");  map.put("hello", hello);  return "helloJsp";  } }  --------------------------  编写JSP页面  在 src/main 下面创建 webapp/WEB-INF/jsp 目录用来存放我们的jsp页面：helloJsp.jsp:  <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <title>Insert title here</title>  </head>  <body>  helloJsp  <hr>  ${hello}    </body>  </html>  --------------------------  编写启动类  @SpringBootApplication public class App {   public static void main(String[] args) {  SpringApplication.run(App.class, args);  } }  注意：  特别说明：针对el表达式，类似${hello} 这个对于servlet的版本是有限制的，2.4版本版本以下是不支持的，是无法进行识别的，请注意 |

# myBatis

|  |
| --- |
| 1.集成MyBatis  2.使用PageHelper分页  3.获取自增长ID  Spring Boot 集成Mybatis:  ---------------------------------------  1、新建一个新的Maven Project;  2、需要在pom.xml文件添加相应的依赖，  比如：mysql驱动；  PageHelper分页插件，需要版本号；  3、编写启动类，大部分和之前的代码是一样的，需要的注意的是：  需要添加一个注解@MapperScan --指定MyBatis持久类的位置；  4、编写一个测试的实体类Demo;  5、编写一个DemoMapper,使用@Select和@Save进行数据库操作；  使用@Options配置返回的主键信息；  6、编写DemoService;  7、编写DemoController;  8、添加分页配置信息，需要添加一个MyBatisConfiguration;  --------------------------  操作步骤  （1）新建maven project;  （2）在pom.xml文件中引入相关依赖；  （3）创建启动类App.java  （4）在application.properties添加配置文件；  （5）编写Demo测试类;  （6）编写DemoMapper；  （7）编写DemoService  （8）编写DemoController;  （9）加入PageHelper  （10）获取自增长ID;  1.在pom.xml文件中引入相关依赖  （1）基本依赖，jdk版本号； （2）mysql驱动，mybatis依赖包，mysql分页PageHelper:  <!-- mysql 数据库驱动. --> <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId> </dependency>  <!--  spring-boot mybatis依赖：   请不要使用1.0.0版本，因为还不支持拦截器插件，  1.1.1 是博主写帖子时候的版本，大家使用最新版本即可  --> <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.1.1</version> </dependency>  <!--  MyBatis提供了拦截器接口，我们可以实现自己的拦截器，  将其作为一个plugin装入到SqlSessionFactory中。  Github上有位开发者写了一个分页插件，我觉得使用起来还可以，挺方便的。  Github项目地址：  https://github.com/pagehelper/Mybatis-PageHelper  --> <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>4.1.0</version> </dependency>  -------------------------------  2.创建启动类App.java  @SpringBootApplication @MapperScan("com.kfit.\*.mapper") public class App {  public static void main(String[] args) {  SpringApplication.run(App.class, args);  } } //这里和以往不一样的地方就是MapperScan的注解，这个是会扫描该包下的接口  -------------------------------  3.在application.properties添加配置文件  ######################################################## ###datasource ######################################################## spring.datasource.url = jdbc:mysql://localhost:3306/test spring.datasource.username = root spring.datasource.password = root spring.datasource.driverClassName = com.mysql.jdbc.Driver spring.datasource.max-active=20 spring.datasource.max-idle=8 spring.datasource.min-idle=8 spring.datasource.initial-size=10   1. 编写Demo测试类   public class Demo {  private long id;  private String name;  //省略getter and setter…. }   1. 编写DemoMapper   public interface DemoMappper {   @Select("select \*from Demo where name = #{name}")  public List<Demo> likeName(String name);   @Select("select \*from Demo where id = #{id}")  public Demo getById(long id);   @Select("select name from Demo where id = #{id}")  public String getNameById(long id); }  6.编写DemoService  @Service public class DemoService {  @Autowired  private DemoMappper demoMappper;   public List<Demo> likeName(String name){  return demoMappper.likeName(name);  } }  7.编写DemoController  @RestController  public class DemoController {  @Autowired  private DemoService demoService;   @RequestMapping("/likeName")  public List<Demo> likeName(String name){  return demoService.likeName(name);  }   }  //运行访问：http://127.0.0.1:8080/likeName?name=张三 就可以看到返回的数据了  8.加入PageHelper  @Configuration public class MyBatisConfiguration {   @Bean  public PageHelper pageHelper() {  System.***out***.println("MyBatisConfiguration.pageHelper()");  PageHelper pageHelper = new PageHelper();  Properties p = new Properties();  p.setProperty("offsetAsPageNum", "true");  p.setProperty("rowBoundsWithCount", "true");  p.setProperty("reasonable", "true");  pageHelper.setProperties(p);  return pageHelper;  } }  @RequestMapping("/likeName") public List<Demo> likeName(String name){  PageHelper.startPage(1,1);  return demoService.likeName(name); }  9.获取自增长ID  @Insert("insert into Demo(name,password) values(#{name},#{password})") public long save(Demo name);  @Options(useGeneratedKeys = true, keyProperty = "id", keyColumn = "id") |

--------------

pom.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>com.kfit</groupId>  <artifactId>spring-boot-mybatis</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>  <name>spring-boot-mybatis</name>  <url>http://maven.apache.org</url>  <!-- spring boot parent节点，引入这个之后，在下面和spring boot相关的就不需要引入版本了; -->  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.4.1.RELEASE</version>  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <!-- 指定一下jdk的版本 ，这里我们使用jdk 1.8 ,默认是1.6 -->  <java.version>1.8</java.version>  </properties>  <dependencies>    <!-- web支持: 1、web mvc; 2、restful; 3、jackjson支持; 4、aop ........ -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>    <!-- mysql 数据库驱动. -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>    <!--  spring-boot mybatis依赖：    请不要使用1.0.0版本，因为还不支持拦截器插件，  1.1.1 是博主写帖子时候的版本，大家使用最新版本即可  -->  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.1.1</version>  </dependency>      <!--  MyBatis提供了拦截器接口，我们可以实现自己的拦截器，  将其作为一个plugin装入到SqlSessionFactory中。  Github上有位开发者写了一个分页插件，我觉得使用起来还可以，挺方便的。  Github项目地址： https://github.com/pagehelper/Mybatis-PageHelper  -->  <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>4.1.0</version>  </dependency>  </dependencies>  </project> |

application.properties

|  |
| --- |
| ########################################################  ###datasource -- mysql\u7684\u6570\u636e\u5e93\u914d\u7f6e.  ########################################################  spring.datasource.url = jdbc:mysql://localhost:3306/test  spring.datasource.username = root  spring.datasource.password = root  spring.datasource.driverClassName = com.mysql.jdbc.Driver  spring.datasource.max-active=20  spring.datasource.max-idle=8  spring.datasource.min-idle=8  spring.datasource.initial-size=10 |

config

|  |
| --- |
| package com.kfit.spring\_boot\_mybatis.config;  import java.util.Properties;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import com.github.pagehelper.PageHelper;  @Configuration  public class MyBatisConfiguration {    @Bean  public PageHelper pageHelper() {  System.out.println("MyBatisConfiguration.pageHelper()");  PageHelper pageHelper = new PageHelper();  Properties p = new Properties();  p.setProperty("offsetAsPageNum", "true");  p.setProperty("rowBoundsWithCount", "true");  p.setProperty("reasonable", "true");  pageHelper.setProperties(p);  return pageHelper;  }  } |

App.java

|  |
| --- |
| package com.kfit.spring\_boot\_mybatis;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  /\*\*  \* Spring Boot启动类.  \* @author Angel -- 守护天使  \* @version v.0.1  \* @date 2017年1月1日  \*/  @SpringBootApplication  @MapperScan("com.kfit.\*")//扫描：该包下相应的class,主要是MyBatis的持久化类.  public class App {  public static void main(String[] args) {  SpringApplication.run(App.class, args);  }  } |

Demo.java

|  |
| --- |
| package com.kfit.spring\_boot\_mybatis;  public class Demo {  private long id;  private String name;  public long getId() {  return id;  }  public void setId(long id) {  this.id = id;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  } |

DemoController.java

|  |
| --- |
| package com.kfit.spring\_boot\_mybatis;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  import com.github.pagehelper.PageHelper;  @RestController  public class DemoController {    @Autowired  private DemoService demoService;    @RequestMapping("/likeName")  public List<Demo> likeName(String name){  /\*  \* 第一个参数：第几页;  \* 第二个参数：每页获取的条数.  \*/  PageHelper.startPage(1, 2);  return demoService.likeName(name);  }    @RequestMapping("/save")  public Demo save(){  Demo demo = new Demo();  demo.setName("张三");  demoService.save(demo);  return demo;  }  } |

DemoMappper.java

|  |
| --- |
| package com.kfit.spring\_boot\_mybatis;  import java.util.List;  import org.apache.ibatis.annotations.Insert;  import org.apache.ibatis.annotations.Options;  import org.apache.ibatis.annotations.Select;  public interface DemoMappper {    //#{name}:参数占位符  @Select("select \*from Demo where name=#{name}")  public List<Demo> likeName(String name);      @Select("select \*from Demo where id = #{id}")  public Demo getById(long id);    @Select("select name from Demo where id = #{id}")  public String getNameById(long id);    /\*\*  \* 保存数据.  \*/  @Insert("insert into Demo(name) values(#{name})")  @Options(useGeneratedKeys=true,keyProperty="id",keyColumn="id")  public void save(Demo demo);    } |

DemoService.java

|  |
| --- |
| package com.kfit.spring\_boot\_mybatis;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;  @Service  public class DemoService {  @Autowired  private DemoMappper demoMappper;    public List<Demo> likeName(String name){  return demoMappper.likeName(name);  }    @Transactional//添加事务.  public void save(Demo demo){  demoMappper.save(demo);  }  } |