# Spring Boot application Hello

## 1.1修改pom.xml

|  |
| --- |
| 1.添加父节点依赖  <!--  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的配置，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>1.5.3.RELEASE</version>  </parent> |
| 1. 设置jdk的版本，在properties   <java.version>1.8</java.version> |
| 1. spring-boot-starter-web依赖的添加，这里面包含了MVC、aop   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <!-- 不添加版本是因为parent中已经有了 -->  </dependency> |

注：如果项目中有错误，我们Update Project来处理，勾选上Force Update of Snapshots/Releases

## 1.2创建Spring boot 的启动类

|  |
| --- |
| /\*\*  \* SpringBoot应用的加载类  \* 使用@SpringBootApplication来指定这是一个Spring boot的程序  \*  \*/  @SpringBootApplication  **public** **class** App  {  **public** **static** **void** main( String[] args )  {  //在main方法进行启动我们的应用程序  SpringApplication.*run*(App.**class**, args);  }  } |

## 1.3Controller的创建

|  |
| --- |
| /\*\*  \*  \* 在这里使用@RestController等价于@Controller+@ResponseBody  \*/  @RestController  **public** **class** HelloController {    @RequestMapping("/hello")  **public** String hello(){  **return** "hello Spring boot!";  }  } |

# 2. Spring Boot使用FastJson

## 2.1方式一

1. 添加依赖

|  |
| --- |
| <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.32</version>  </dependency> |

2. 启动类继承extends WebMvcConfigurerAdapter

3. 覆盖方法configureMessageConverters

|  |
| --- |
| @Override  **public** **void** configureMessageConverters(List<HttpMessageConverter<?>> converters) {  **super**.configureMessageConverters(converters);  //需要先定义一个convert转换消息对象  FastJsonHttpMessageConverter fastConverter = **new** FastJsonHttpMessageConverter();  //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***);  //在convert中添加配置信息  fastConverter.setFastJsonConfig(fastJsonConfig);  //将convert添加到converters当中  converters.add(fastConverter);  } |

为什么要使用fastJson，可以使用fastJson的好处

@JSONField(format="yyyy-MM-dd HH:mm") 格式化

@JSONField(serialize=false) 不序列化

## 2.2方式二

创建一个HttpMessageConverters的实例对象

|  |
| --- |
| @Bean  **public** HttpMessageConverters fastJsonHttpMessageConverters(){  // 1、需要先定义一个 convert 转换消息的对象;  FastJsonHttpMessageConverter converter = **new** FastJsonHttpMessageConverter();    FastJsonConfig fastJsonConfig = **new** FastJsonConfig();  fastJsonConfig.setSerializerFeatures(SerializerFeature.***PrettyFormat***);    converter.setFastJsonConfig(fastJsonConfig);    **return** **new** HttpMessageConverters(converter);  } |

注：中文乱码的处理方式

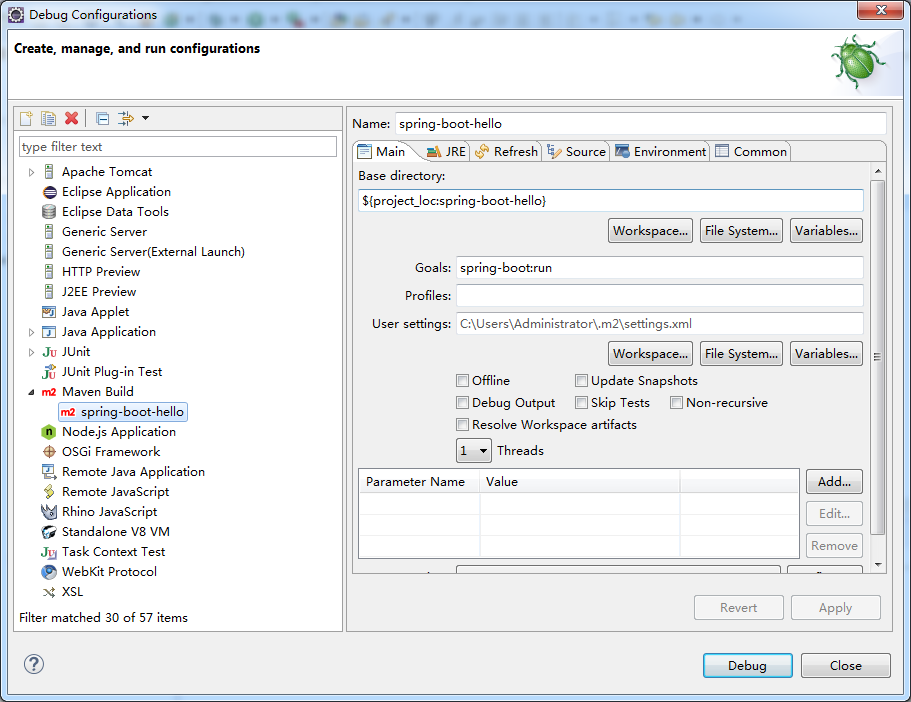
|  |
| --- |
| //附加：处理中文乱码（后期添加）  List<MediaType> fastMedisTypes = **new** ArrayList<MediaType>();  fastMedisTypes.add(MediaType.***APPLICATION\_JSON\_UTF8***);  converter.setSupportedMediaTypes(fastMedisTypes); |

# 3.Spring boot热部署

## 3.1方式一

在pom中添加如些配置，设置spring-boot:run

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin </artifactId>  <dependencies>  <!--springloaded hot deploy -->  <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>  </plugins>  </build> |

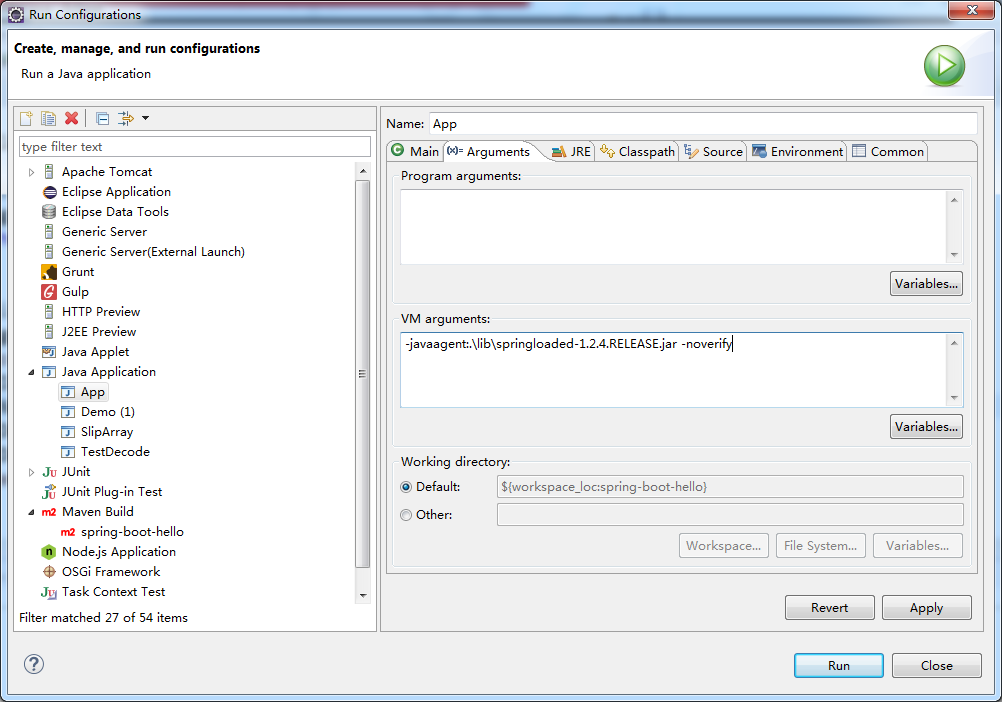


注：这种方式不好的地方就是项目关闭进程还在运行，会出现地址被占用

## 3.2方式二

1. 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：

-javaagent:.\lib\springloaded-1.2.4.RELEASE.jar -noverify



## 3.3方式三

上面的两种都实现了热部署，但是我们在添加新接口的依然不能热部署，所以我们使用Spring-boot-devtools

1.添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <optional>true</optional>  <scope>true</scope>  </dependency> |

2. 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，devtools不会起作用，即应用不会restart -->  <fork>true</fork>  </configuration>  </plugin>  </plugins>  </build> |

注: 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)

* 是否加入plugin以及属性<fork>true</fork>
* 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）。
* 如果设置SpringApplication.setRegisterShut时候关闭了自动编译的功能downHook(false)，则自动重启将不起作用。

# Spring-data-jpa

## 4.1使用步骤

1.pom中添加对应的依赖

|  |
| --- |
| <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency> |

2.在应用中添加application.properties配置文件设置datasource

|  |
| --- |
| ########################################################  ###datasource  ########################################################  spring.datasource.url = jdbc:mysql://localhost:3307/spring\_data  spring.datasource.username = root  spring.datasource.password = 963852  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.在配置文件中添加jpa的配置信息

|  |
| --- |
| ########################################################  ### 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

1.创建实体类

|  |
| --- |
| **package** cn.yanliang.jpademo.domain;  **import** javax.persistence.Entity;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.GenerationType;  **import** javax.persistence.Id;  /\*\*  \* 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** Integer id;  **private** String catName;  **private** Integer catAge;    **public** Integer getId() {  **return** id;  }  **public** **void** setId(Integer id) {  **this**.id = id;  }  **public** String getCatName() {  **return** catName;  }  **public** **void** setCatName(String catName) {  **this**.catName = catName;  }  **public** Integer getCatAge() {  **return** catAge;  }  **public** **void** setCatAge(Integer catAge) {  **this**.catAge = catAge;  }  } |

2.编写repository

|  |
| --- |
| 创建repository的一个接口并集成于CrudRepository  **package** cn.yanliang.jpademo.repository;  **import** org.springframework.data.repository.CrudRepository;  **import** cn.yanliang.jpademo.domain.Cat;  **public** **interface** CatRepository **extends** CrudRepository<Cat, Integer>{  } |

3.编写service

|  |
| --- |
| **package** cn.yanliang.jpademo.service;  **import** java.util.List;  **import** javax.annotation.Resource;  **import** javax.transaction.Transactional;  **import** org.springframework.stereotype.Service;  **import** cn.yanliang.jpademo.domain.Cat;  **import** cn.yanliang.jpademo.repository.CatRepository;  @Service  **public** **class** CatService {  @Resource  **private** CatRepository catRepository;    @Transactional  **public** **void** save(Cat cat){  catRepository.save(cat);  }    @Transactional  **public** **void** delete(Integer id){  catRepository.delete(id);  }    **public** Iterable<Cat> getAll(){  **return** catRepository.findAll();  }  } |

4.编写controller

|  |
| --- |
| **package** cn.yanliang.jpademo.controller;  **import** javax.annotation.Resource;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** cn.yanliang.jpademo.domain.Cat;  **import** cn.yanliang.jpademo.service.CatService;  @RestController  @RequestMapping("/cat")  **public** **class** CatController {  @Resource  **private** CatService catService;    @RequestMapping("/save")  **public** String save(){  Cat cat = **new** Cat();  cat.setCatName("cat1");  cat.setCatAge(1);  catService.save(cat);  **return** "save success!";  }    @RequestMapping("/delete")  **public** String delete(){  catService.delete(1);  **return** "delete success!";  }    @RequestMapping("/findAll")  **public** Iterable<Cat> findAll(){  **return** catService.getAll();  }  } |

# 5.Spring boot JdbcTemplate的使用

## 5.1pom中添加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> |

## 5.2实现Dao类，使用@Repository申明，并引入JdbcTemplate

|  |
| --- |
| **package** cn.yanliang.jpademo.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** cn.yanliang.jpademo.domain.Cat;  @Repository  **public** **class** CatDao {  @Resource  **private** JdbcTemplate jdbcTemplate;    **public** Cat selectByCatName(String catName){  /\*  \* 1.先定义sql语句  \* 2.常见RowMapper对象  \* 3.通过JdbcTemplate来查询  \*/  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;  }  } |

# 6.全局异常的捕获

|  |
| --- |
| **package** cn.yanliang.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;  /\*\*  \* 全局异常的处理  \* 1.创建一个GlobaDefaultException的类  \* 2.在class上添加@ControllerAdvice注解  \* 3.在类中添添加一个方法在方法上添加@ExceptionHandler(Exception.class)  \* 4.如果返回的是View -- 方法的返回值是ModelAndView;  \* 5.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** Administrator  \*  \*/  @ControllerAdvice  **public** **class** GlobaDefaultException {    @ExceptionHandler(Exception.**class**)  @ResponseBody  **public** String defaultExceptionHandler(HttpServletRequest request, Exception e){  **return** e.getMessage().toString();  }  } |

# 7.server的配置

## 1.设置server的端口号

server.port = 8081

## 2.设置server的context-path

server.context-path = /springboot

## 3.其它的server

|  |
| --- |
| #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 |

# 8.Spring boot使用thymeleaf

## 8.1添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-thymeleaf</artifactId>  </dependency> |

## 8.2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 |

## 8.3编写html模板

|  |
| --- |
| 编写模板文件src/main/resouces/templates/ hello1.html:  <!DOCTYPE html>  <html>  <head>  <meta charset=*"UTF-8"* />  <title>thymeleaf hello</title>  </head>  <body>  <h1>hello Spring boot</h1>  <h1>this is first thymeleaf </h1>  欢迎你，<span th:text=*"${name}"*></span>  </body>  </html> |

## 8.4编写controller

|  |
| --- |
| package cn.yanliang.thymeleaf;  import java.util.Map;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  @Controller  @RequestMapping("/templates")  public class TemplateController {    @RequestMapping("/hello")  public String hello(Map<String, String> map){  map.put("name", "test");  return "hello1";//页面是hello1.html  }  } |

# 9.Spring boot 使用Freemarker

## 9.1添加依赖

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-freemarker</artifactId>  </dependency> |

## 9.2关闭缓存

|  |
| --- |
| ########################################################  ###FREEMARKER (FreeMarkerAutoConfiguration)  ########################################################  spring.freemarker.allow-request-override=false  spring.freemarker.cache=false  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 |

## 9.3编写模板文件.ftl -- helloFtl.ftl

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8" />  <title>freemarker hello</title>  </head>  <body>  hello freemarker!<br/>  欢迎你，<span>${name}</span>  </body>  </html> |

## 9.4编写controller

|  |
| --- |
| @RequestMapping("/helloFtl")  **public** String helloFtl(Map<String, String> map){  map.put("name", "test");  **return** "helloFtl";// helloFtl是文件的名字，对应的模板就是helloFtl.ftl  } |

# 10.Spring boot 使用jsp

## 10.1创建一个创建Maven web project

## 10.2添加依赖

|  |
| --- |
| 添加父节点依赖  <!--  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的配置，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>1.5.3.RELEASE</version>  </parent> |
| 设置jdk的版本，在properties  <java.version>1.8</java.version> |
| spring-boot-starter-web依赖的添加  <!-- 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>  <!-- 不添加版本是因为parent中已经有了 -->  </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> |

## 10.3创建application.properties

|  |
| --- |
| 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 |

## 10.4编写jsp文件

|  |
| --- |
| <%@ 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>  this is jsp.<br/>  欢迎你，${name}  </body>  </html> |

## 10.5编写controller

|  |
| --- |
| **import** java.util.Map;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  @Controller  **public** **class** HelloJsp {  @RequestMapping("/hellojsp")  **public** String hello(Map<String, String> map){  map.put("name", "test");  **return** "hello";  }  } |

# 11.Spring boot继承myBatis

## 11.1添加依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.3.RELEASE</version>  </parent> |
| <java.version>1.8</java.version> |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <!-- 不添加版本是因为parent中已经有了 -->  </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，  -->  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.0</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>5.0.1</version>  </dependency> |

## 11.2创建App.java

|  |
| --- |
| **package** cn.yanliang.spring\_boot\_mybatis;  **import** org.mybatis.spring.annotation.MapperScan;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  /\*\*  \* **@MapperScan** 这个是会扫描该包下的接口  \*/  @SpringBootApplication  @MapperScan("cn.yanliang.spring\_boot\_mybatis.\*")  **class** App {  **public** **static** **void** main( String[] args ){  SpringApplication.*run*(App.**class**, args);  }  } |

## 11.3添加application.properties

|  |
| --- |
| ########################################################  ###datasource  ########################################################  spring.datasource.url = jdbc:mysql://localhost:3307/spring\_data  spring.datasource.username = root  spring.datasource.password = 963852  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 |

## 11.4编写实体类

|  |
| --- |
| **package** cn.yanliang.spring\_boot\_mybatis.entity;  **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;  }  } |

## 11.5编写DemoMapper

|  |
| --- |
| **package** cn.yanliang.spring\_boot\_mybatis.mapper;  **import** java.util.List;  **import** org.apache.ibatis.annotations.Select;  **import** org.springframework.stereotype.Repository;  **import** cn.yanliang.spring\_boot\_mybatis.entity.Demo;  @Repository  **public** **interface** DemoMapper {  @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);  } |

## 11.6编写service

|  |
| --- |
| **package** cn.yanliang.spring\_boot\_mybatis.service;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** cn.yanliang.spring\_boot\_mybatis.entity.Demo;  **import** cn.yanliang.spring\_boot\_mybatis.mapper.DemoMapper;  @Service  **public** **class** DemoService {    @Autowired  **private** DemoMapper demoMapper;    **public** List<Demo> likeName(String name){  **return** demoMapper.likeName(name);  }  } |

## 11.7编写Controller

|  |
| --- |
| **package** cn.yanliang.spring\_boot\_mybatis.controller;  **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** cn.yanliang.spring\_boot\_mybatis.entity.Demo;  **import** cn.yanliang.spring\_boot\_mybatis.service.DemoService;  @RestController  **public** **class** DemoController {    @Autowired  **public** DemoService demoService;    @RequestMapping("/likeName")  **public** List<Demo> likeName(String name){  **return** demoService.likeName(name);  }  } |

## 11.8加入PageHelper实现分页

|  |
| --- |
| **添加分页配置**  **package** cn.yanliang.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;  }  } |

|  |
| --- |
| 添加分页拦截 -- 在controller上  PageHelper.startPage(1,1); |

## 11.9获取自增长的Id

|  |
| --- |
| @Insert("insert into demo(name) value (#{name})")  @Options(useGeneratedKeys = **true**, keyProperty = "id", keyColumn = "id")  **public** Long save(Demo demo); |

# Spring Boot 集成Druid

## 添加依赖

|  |
| --- |
| <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.0.18</version>  </dependency> |

## 添加配置项

|  |
| --- |
| spring:  application:  name: microservice-druid  datasource:  type: com.alibaba.druid.pool.DruidDataSource  driver-class-name: com.mysql.jdbc.Driver  url: jdbc:mysql://localhost:3307/test?characterEncoding=utf8  username: root  password: 963852  # 初始化大小，最小，最大  initialSize: 5  minIdle: 5  maxActive: 20  # 配置获取连接等待超时的时间  maxWait: 60000  # 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒  timeBetweenEvictionRunsMillis: 60000  # 配置一个连接在池中最小生存的时间，单位是毫秒  minEvictableIdleTimeMillis: 300000  validationQuery: SELECT 1 FROM DUAL  testWhileIdle: **true**  testOnBorrow: **false**  testOnReturn: **false**  # 打开PSCache，并且指定每个连接上PSCache的大小  poolPreparedStatements: **true**  maxPoolPreparedStatementPerConnectionSize: 20  # 配置监控统计拦截的filters，去掉后监控界面sql无法统计，'wall'用于防火墙  filters: stat,wall,log4j  # 通过connectProperties属性来打开mergeSql功能；慢SQL记录  connectionProperties: druid.stat.mergeSql=true;druid.stat.slowSqlMillis=5000  # 合并多个DruidDataSource的监控数据  #useGlobalDataSourceStat=true |

## 添加监控类

|  |
| --- |
| **package** cn.yanliang.config;  **import** org.springframework.boot.web.servlet.FilterRegistrationBean;  **import** org.springframework.boot.web.servlet.ServletRegistrationBean;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** com.alibaba.druid.support.http.StatViewServlet;  **import** com.alibaba.druid.support.http.WebStatFilter;  @Configuration  **public** **class** DruidConfiguration {  @Bean  **public** ServletRegistrationBean statViewServle(){  ServletRegistrationBean servletRegistrationBean = **new** ServletRegistrationBean(**new** StatViewServlet(), "/druid/\*");  // IP白名单  servletRegistrationBean.addInitParameter("allow","127.0.0.1");  // IP黑名单 (共同存在时，deny优先于allow)  servletRegistrationBean.addInitParameter("deny","192.168.1.100");  // 控制台管理用户  // servletRegistrationBean.addInitParameter("loginUsername","druid");  // servletRegistrationBean.addInitParameter("loginPassword","963852");  // 是否能够重置数据  servletRegistrationBean.addInitParameter("resetEnable","false");  **return** servletRegistrationBean;  }    @Bean  **public** FilterRegistrationBean statFilter(){  FilterRegistrationBean filterRegistrationBean = **new** FilterRegistrationBean(**new** WebStatFilter());  // 添加过滤规则  filterRegistrationBean.addUrlPatterns("/\*");  // 忽略过滤的格式  filterRegistrationBean.addInitParameter("exclusions","\*.js,\*.gif,\*.jpg,\*.png,\*.css,\*.ico,/druid/\*");  **return** filterRegistrationBean;  }  } |