# Spring Boot之Hello World

## 1.1spring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。

## 1.2如何建立一个hello的项目

开发环境JDK 1.8（必须是1.8以上）

开发工具(intellij idea)

项目管理工具（ Maven ）

### 1.2.1 配置pom.xml

GroupId ：com.yg

ArtifactId : 项目名称

<groupId>com.yg</groupId>

<artifactId>spring\_boot\_hello</artifactId>

<version>1.0-SNAPSHOT</version>

<!-- parent的 配置信息-->

<parent>

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

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

<version>1.5.8.RELEASE</version>

<relativePath/>

</parent>

<!--项目版本新-->

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<!--依赖的包-->

<dependencies>

<dependency>

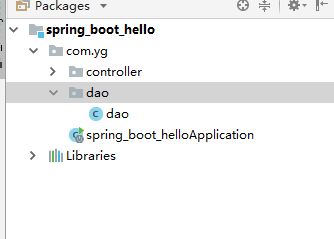
<groupId>org.springframework.boot</groupId>

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

</dependency>

</dependencies>

### 1.2.2项目结构



### 1.2.3controller

//在这里我们使用RestController （等待于 @Controller 和 @RequestBody）

@RestController

public class HelloController1 {

@RequestMapping("/hello")

public String hello(){

return "hello1";

}

}

### 1.2.4入口类

1. 建立一个入口类 项目名+Application

位置并且在所有包之外

1. test url 地址 ：<http://localhost:8080/hello>

@SpringBootApplication

public class spring\_boot\_helloApplication {

public static void main(String args[]){

SpringApplication.run(spring\_boot\_helloApplication.class,args);

// SpringApplication springApplication = new SpringApplication(spring\_boot\_helloApplication.class);

// springApplication.run(args);

}

}

# 2.spring boot返回json数据

步骤

1. 编写实体类user

2. 编写getDemo()方法

3. 测试

public class User {

private int id;

private String name ;

….省略get和set方法

}

@RequestMapping("user")

public User getUser(){

User user = new User();

user.setId(1);

user.setName("于光");

return user;

}

测试 ：<http://localhost:8080/user>

注意：这里面并没有使用json，但是却返回json 因为默认的有jackson

# 3.使用fastjson（两种方法）

（1）启动类继承extends WebMvcConfigurerAdapter

（2）覆盖方法configureMessageConverters

## 3.1加入依赖

*<!--添加fastjson的依赖包-->*<**dependency**>  
 <**groupId**>com.alibaba</**groupId**>  
 <**artifactId**>fastjson</**artifactId**>  
 <**version**>1.2.15</**version**>  
</**dependency**>

## 3.2第一种方法

添加到启动类里面的

public class spring\_boot\_helloApplication 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;  
 \* 2.添加fastJson 的配置信息，比如：是否要格式化返回的json数据;  
 \* 3.在convert中添加配置信息.  
 \* 4.将convert添加到converters当中.  
 \*/  
 //1.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  
 FastJsonConfig fastJsonConfig = new FastJsonConfig();  
 fastJsonConfig.setSerializerFeatures(SerializerFeature.PrettyFormat);  
 //设置编码格式  
 fastJsonConfig.setCharset(Charset.forName("utf-8"));  
 //3.convert 添加配置信息  
 fastConverter.setFastJsonConfig(fastJsonConfig);  
 //4.将定义的convert 添加到 converters 就是这个方法里面的参数  
 System.out.println("使用了 自定义的json解析器");  
 converters.add(fastConverter);  
 }

注意：user里面的这里

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

private Date createTime; //创建时间

## 3.3第二种方法、

//通过注解的方式 其实道理是一样的

//converters 就是包含了很多转换器的东西，，所有自定义的都要添加进来

@Bean  
public HttpMessageConverters fastJsonHttpMessageConverters() {  
  
 FastJsonHttpMessageConverter fastConverter = new FastJsonHttpMessageConverter();  
  
 FastJsonConfig fastJsonConfig = new FastJsonConfig();  
 fastJsonConfig.setSerializerFeatures(SerializerFeature.PrettyFormat);  
 fastJsonConfig.setCharset(Charset.forName("gbk"));  
  
 fastConverter.setFastJsonConfig(fastJsonConfig);  
  
 HttpMessageConverter<?> converter = fastConverter;  
 //在converters 里面加入converter  
 return new HttpMessageConverters(converter);  
}

注意一点;

@JSONField(serialize = true) //是否序列化 就是是否显示 默认是显示的

private String remark;// 备注

## 总结：

我们要使用第三方的json解析框架的话：

1、我们需要在pom.xml中引入相应的依赖；

2、需要在App.java中继承WebMvcConfigurerAdapter重写方法：configureMessageConverters 添加我们自己定义的json解析框架;

2.1 @Bean注入第三方的json解析框架：

过程都是四个：

1. 需要先定义一个 convert 转换消息的对象;

2.添加fastJson 的配置信息，比如：是否要格式化返回的json数据;

3.在convert中添加配置信息.

4.将convert添加到converters当中.

# 4．热部署 springloader（不怎么有用）

问题的提出：

在编写代码的时候，你会发现我们只是简单把打印信息改变了，就需要重新部署，如果是这样的编码方式，那么我们估计一天下来就真的是打几个Hello World就下班了。那么如何解决热部署的问题呢？那就是springloaded

# 5．springboot + devtools（热部署）

## 5.1devtools介绍

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。

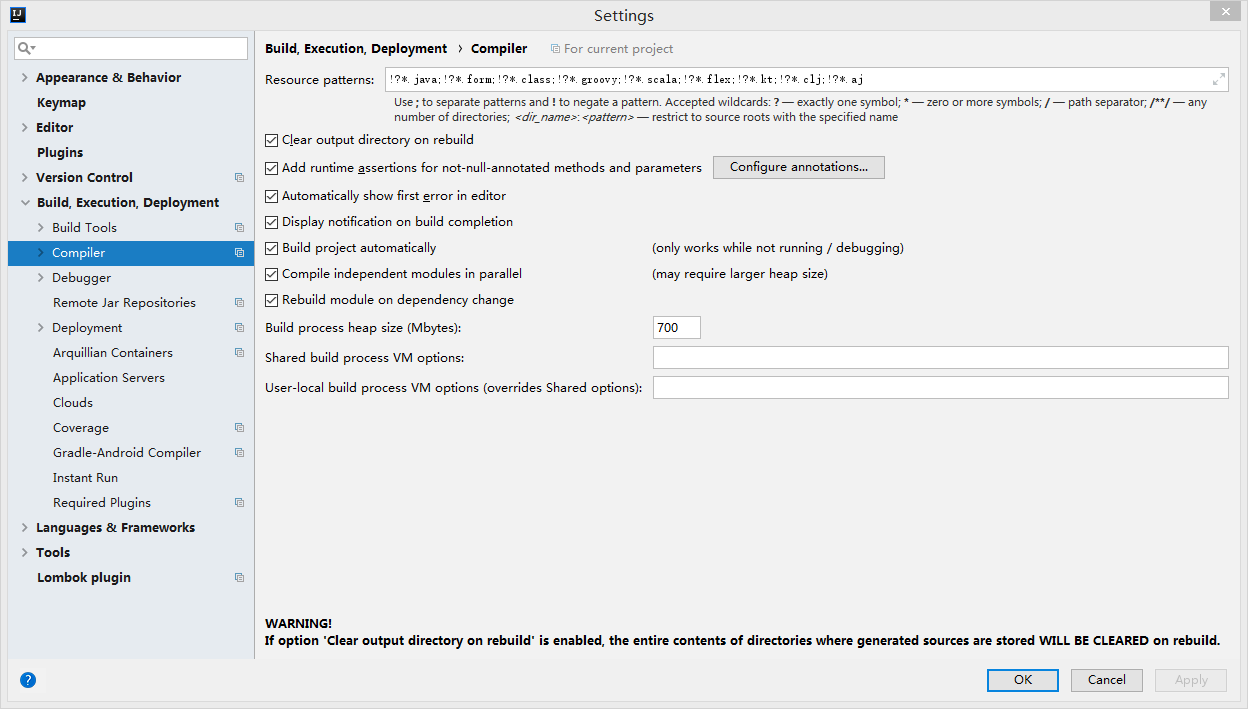
其深层原理是使用了两个ClassLoader，一个Classloader加载那些不会改变的类（第三方Jar包），另一个ClassLoader加载会更改的类，称为 restart ClassLoader

,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）。

## 5.2使用方法

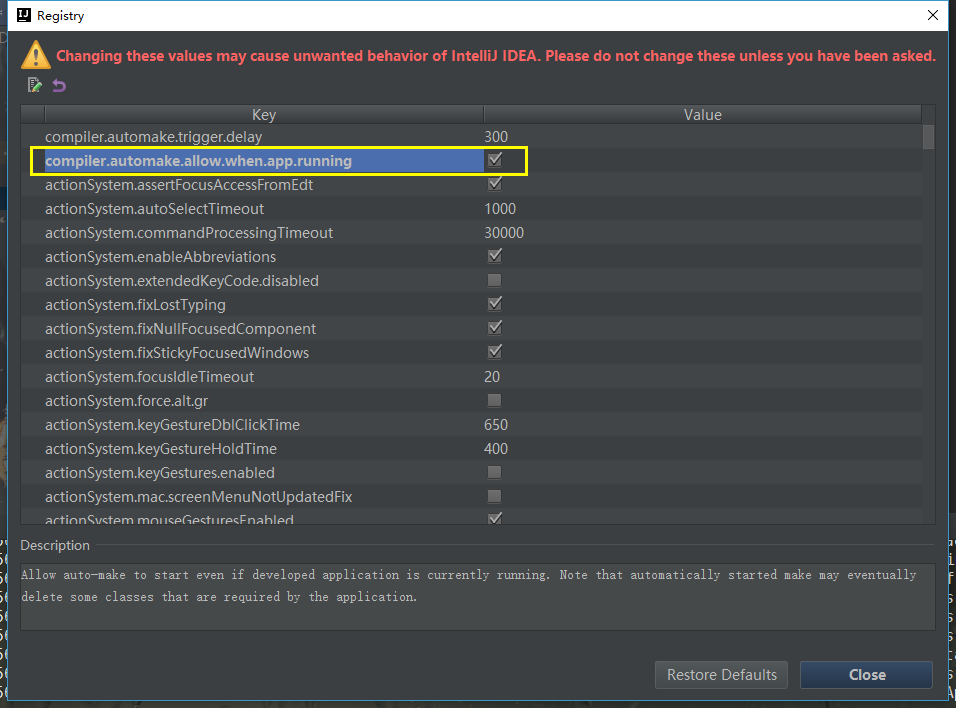
### 使用之前解决不能自动编译的问题

第一步：file –> setting 搜索compiler



第二步：Shift+Ctrl+Alt+/，选择Registry

最后重启项目就可以了



### 5.2.1添加依赖包

<dependency>

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

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

<optional>true</optional>

<scope>true</scope>

</dependency>

<**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**>

### 5.2.2添加spring-boot-maven-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**>

### 5.2.3注意

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)。

如何测试：

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）

eclipse的使用分析



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；

# 6. JPA/Hibernate/Spring Data概念

## 6.1基本概念

### 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。百度百科JPA

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

### 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。

### 什么是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？（crud）

我们先看一个描述：

crud ：create Retrieve update delete

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。

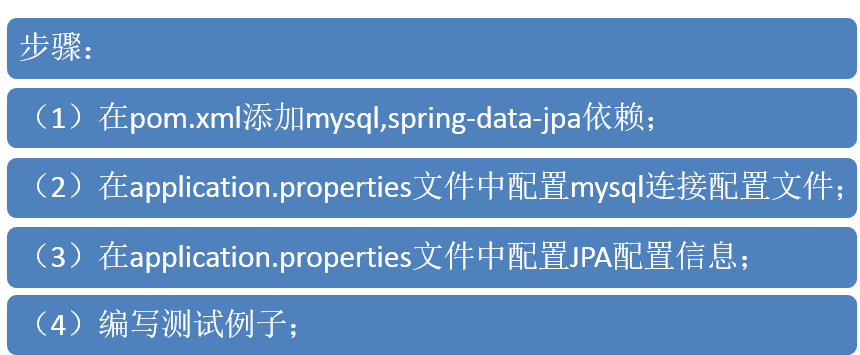
### jpa（规范）和hibernate的关系？

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。

# 7. Spring Boot JPA-Hibernate 操作

## 7.1步骤



## 7.2配置过程

### 添加依赖pom.xml

<!-- mysql的依赖-->

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

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

</dependency>

<!--jps的依赖-->

<dependency>

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

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

</dependency>

### application.properties配置

#端口

#server.port=8090

#数据库连接池

########################################################

###datasource mysql的配置

########################################################

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

spring.datasource.username = root

spring.datasource.password = 111111

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

########################################################

### Java Persistence Api jpa的配置

########################################################

# 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

## 7.3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;

### 1、创建实体类

**package** com.yg.demo.bean;  
  
  
**import** javax.persistence.Entity;  
**import** javax.persistence.GeneratedValue;  
**import** javax.persistence.GenerationType;  
**import** javax.persistence.Id;  
  
@Entity  
**public class** Cat {  
  
 *//设置主键 同时设置自动增长* @Id @GeneratedValue(strategy = GenerationType.***AUTO***)  
 **private int id**; *//主键* **private** String **catName**; *//生成字段 cat\_name* **private int catAge**; *////生成字段 cat\_name* **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;  
 }  
}

### 2、CatRepository

**package** com.yg.demo.repository;  
  
**import** com.yg.demo.bean.Cat;  
**import** org.springframework.data.repository.CrudRepository;  
  
**public interface** CatRepository **extends** CrudRepository<Cat,Integer> {  
 //Cat 和主键的类型  
}

### 3、service

**package** com.yg.demo.service;  
  
**import** com.yg.demo.bean.Cat;  
**import** com.yg.demo.repository.CatRepository;  
**import** org.springframework.stereotype.Service;  
  
**import** javax.annotation.Resource;  
**import** javax.transaction.Transactional;  
**import** java.util.Iterator;  
  
@Service  
**public class** CatService {  
  
 @Resource  
 **public** 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* @Transactional  
 **public void** save(Cat cat){  
 **catRepository**.save(cat);  
 }  
  
 @Transactional  
 **public void** delete(Cat cat){  
 **catRepository**.delete(cat.getId());  
 }  
  
 *//获取所有* **public** Iterable<Cat> getAll(){  
 **return catRepository**.findAll();  
 }  
  
  
}

### 4、controller

package com.yg.demo.controller;

import com.yg.demo.bean.Cat;

import com.yg.demo.service.CatService;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import javax.annotation.Resource;

@RestController

@RequestMapping("/cat")

public class CatController {

@Resource

private CatService catService;

@RequestMapping("/save")

public String save(){

Cat cat = new Cat();

cat.setCatAge(1);

cat.setCatName("名字");

catService.save(cat);

return "save ok";

}

@RequestMapping("/delete")

public String delete(){

Cat cat = new Cat();

cat.setId(1);

catService.delete(cat);

return "delete ok";

}

@RequestMapping("/getAll")

public Iterable<Cat> getAll(){

return catService.getAll();

}

}

### 5、测试及问题

<http://localhost:8080/cat/getAll>

乱码问题

只读查询不需要添加事务 即 @Transactional

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；

# 8.节 Spring Boot 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（乱码问题）

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

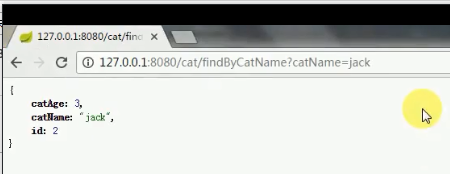
--Page<T> findAll(Pageable pageable); //分页查询（含排序功能）

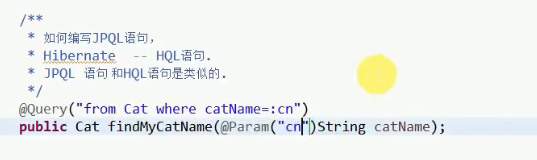
## 其他接口

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

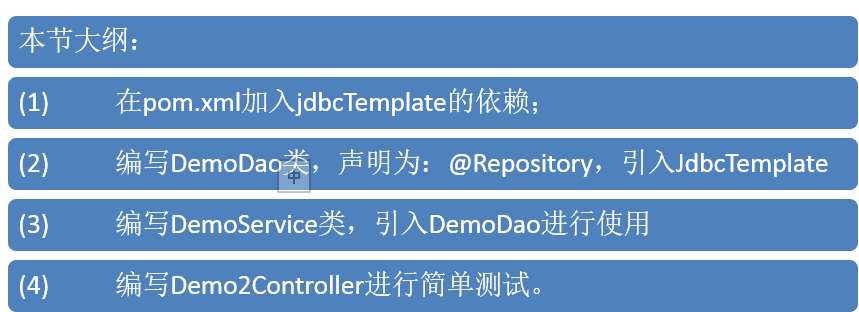
JpaSpecificationExecutor：不属于Repository体系，实现一组 JPA Criteria 查询相关的方法，封装 JPA Criteria 查询条件。通常使用匿名内部类的方式来创建该接口的对象。

自定义 Repository：可以自己定义一个MyRepository接口。





# 9.Spring Boot JdbcTemplate



<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>

## 建立CatDao

@Repository

public class CatDao {

@Resource

private JdbcTemplate jdbcTemplate;

/\*\*

\* 1.sql语句

\* 2.RowMapper

\* @param catName

\* @return

\*/

public Cat selectByCageName(String catName){

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;

}

}

## service里面加入

@Resource  
**private** CatDao **catDao**;

@Transactional  
**public** Cat selectByCageName(String catName){  
  
 **return catDao**.selectByCageName(catName);  
  
}

## controller里面加入

@RequestMapping(**"/selectByCageName"**)  
**public** Cat selectByCageName(String catName){  
 **return catService**.selectByCageName(catName);  
}

## 测试

<http://localhost:8080/cat/selectByCageName?catName=name>

**public** Cat selectByCageName(String catName,**int** cat\_age){  
 String sql = **"select** *\** **from cat where cat\_name=? and cat\_age=?"**;  
 RowMapper<Cat> rowMapper = **new** BeanPropertyRowMapper<>(Cat.**class**);  
 Cat cat = **jdbcTemplate**.queryForObject(sql, **new** Object[]{catName,cat\_age},rowMapper);  
 **return** cat;  
}

<http://localhost:8080/cat/selectByCageName?catName=name&catAge=1>

# 10.全局异常捕捉

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) {

}

\* 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.

/\*\*

\* 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.

\*/

@ControllerAdvice

public class GlobalDefaultExceptionHandler {

@ResponseBody

@ExceptionHandler

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);

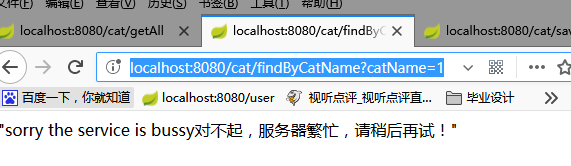
return "sorry the service is bussy对不起，服务器繁忙，请稍后再试！";

}

}

## 测试：

<http://localhost:8080/cat/findByCatName?catName=1>



# 11.Spring Boot之Hello World访问404

----------代码Hello World 之 404 ------

1、确认访问地址是否正确：

（1）确认端口号，默认是8080，这个可以在启动的控制台进行查看；

（2）确认访问的URI地址是否填写正确，这个在启动的控制台查看是否被映射了。

2、确定注解是否正确，使用@RestController而不是@Controller,

另外@RestController等价于@Controller和@ResponseBody;

3、确定包的路径是否正确

我们需要知道的是：Spring Boot默认情况下可以扫描到的是

@SpringBootApplication所在的类的同包或者子包下的类。

4、 确定类引入的包的路径是否正确

@RestController：import org.springframework.web.bind.annotation.RestController

@RequestMapping("/helo33")：import org.springframework.web.bind.annotation.RequestMapping

# 12.配置server信息

server.port=8081

server.context-path=/springboot

地址: <http://localhost:8080/springboot/hello>

访问地址就是<http://ip:port/spring-boot>

#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

# 13. spring boot使用thymeleaf



注意：

作用就是省了头和尾

spring.thymeleaf.prefix=classpath:/templates/

spring.thymeleaf.suffix=.html

## 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:

***路径必须是resource/templates/***

***不然访问不到***

<!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";

}

}

# 14.Spring Boot 使用freemarker



*#spring.freemarker.prefix=  
#spring.freemarker.request-context-attribute=  
#spring.freemarker.settings.\*=  
#spring.freemarker.suffix=.ftl  
#spring.freemarker.template-loader-path=classpath:/templates/*

## 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";

}

# 总结13和14的一起总结：

------在spring boot中如何使用模板引擎的步骤：

1、在pom.xml文件中添加相应的依赖包，thymeleaf和freemarker；

2、在application.properties文件中添加配置信息，开发过程建议关闭缓存；

3、编写模板文件，thymeleaf的默认的后缀是.html,freemarker的默认后缀是.ftl;

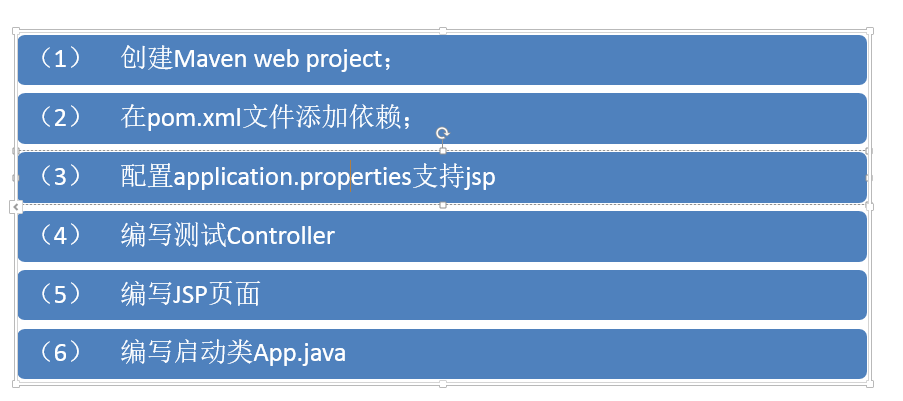
4、编写访问模板文件的controller建立请求映射地址；

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

同时 我们知道了：thymeleaf和freemarker可以并存使用。

# 15.添加jsp支持

步骤：



创建工程不同：

没成功

## 1 . pom.xml的需要添加的

<parent>

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

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

<version>1.5.8.RELEASE</version>

<relativePath/>

</parent>

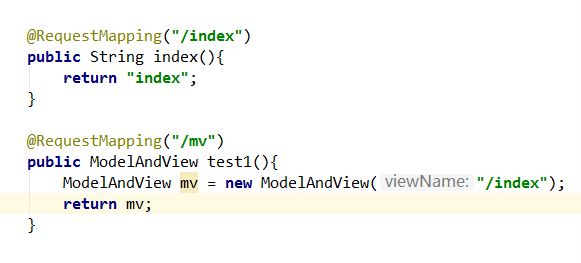
*<!-- 主要的依赖 mvc依赖 包含tomcat-->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>jstl</**artifactId**>  
 <**version**>1.2</**version**>  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>org.apache.tomcat.embed</**groupId**>  
 <**artifactId**>tomcat-embed-jasper</**artifactId**>

*<!--<scope>provided</scope>-->*  
</**dependency**>  
  
<**dependency**>  
 <**groupId**>javax.servlet</**groupId**>  
 <**artifactId**>javax.servlet-api</**artifactId**>  
 <**version**>3.1.0</**version**>  
 <**scope**>provided</**scope**>  
</**dependency**>

## 2 . controller

E:\ProgramFile\Tencent\QQFile\1414081877\Image\C2C\A40~`IIJ0LYC878PCT@EN`W.png

@Controller  
**public class** HelloController {  
  
  
 @RequestMapping(**"/index"**)  
 **public** String index(){  
 **return "index"**;  
 }  
  
 @RequestMapping(**"/mv"**)  
 **public** ModelAndView test1(){  
 ModelAndView mv = **new** ModelAndView(**"/index"**);  
 **return** mv;  
 }



## 注意：问题

1. 新建项目的问题 ，要重新配置devtools
2. 配置maven路径
3. provide的作用是什么

<http://localhost:8080/test>

<http://localhost:8080/index>

<http://localhost:8080/mv>

E:\ProgramFile\Tencent\QQFile\1414081877\Image\C2C\A40~`IIJ0LYC878PCT@EN`W.png

# 16.集成mybatis

（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.集成mybatis（每一个新建项目都要重新配置maven）

### 1 .新建maven project

### 2 . 配置maven路径

### 3 。配置启动类

@SpringBootApplication  
@MapperScan(**"com.yg.\*"**) //扫描的路径  
**public class** spring\_boot\_mybatisApplication {  
 **public static void** main(String args[]){  
 SpringApplication.*run*(spring\_boot\_mybatisApplication.**class**,args);  
 }  
}

### 4 。创建mapper 注意是 interface

@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);

//@Options(useGeneratedKeys = true, keyProperty = "id", keyColumn = "id")

@Insert("insert into Demo(name) values(#{name})")

@Options(useGeneratedKeys = true,keyProperty = "id",keyColumn = "id")

public void save(Demo demo);

}

### 5 。创建service

@Service  
**public class** DemoService {  
  
 @Autowired  
 **private** DemoMapper **demoMapper**;  
  
 **public** List<Demo> likeName(String name){  
 **return demoMapper**.likeName(name);  
 }  
  
 *// 只有查询不需要事务* @Transactional  
 **public void** save(Demo demo){  
 **demoMapper**.save(demo);  
 }  
  
}

### 6 。创建controller

@RestController  
**public class** Controller {  
  
 @Autowired  
 **private** DemoService **demoService**;  
  
 @RequestMapping(**"/likeName"**)  
 **public** List<Demo> likeName(String name){  
 PageHelper.*startPage*(1,2);  
 **return demoService**.likeName(name);  
 }  
  
 @RequestMapping(**"/save"**)  
 **public** Demo save(){  
 Demo demo = **new** Demo();  
 demo.setName(**"于光"**);  
 **demoService**.save(demo);  
 **return** demo;  
 }  
  
7 . application.properties 配置数据库连接

*########################################################  
###datasource  
########################################################***spring.datasource.url** = **jdbc:mysql://localhost:3306/test  
spring.datasource.username** = **root  
spring.datasource.password** = **111111  
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**

}

### demo类

**private long id**;  
**private** String **name**;

### pom.xml 数据

<parent>

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

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

<version>1.5.8.RELEASE</version>

<relativePath/>

</parent>

<!--项目版本新-->

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<!--依赖的包-->

<dependencies>

<dependency>

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

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

</dependency>

<!-- mysql 的-->

<dependency>

<groupId>mysql</groupId>

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

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

<version>1.1.1</version>

</dependency>

<dependency>

<groupId>com.github.pagehelper</groupId>

<artifactId>pagehelper</artifactId>

<version>4.1.0</version>

</dependency>

<dependency>

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

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

<optional>true</optional>

<scope>true</scope>

</dependency>

</dependencies>

<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>

## 2.使用pagehelper

@RequestMapping(**"/likeName"**)  
**public** List<Demo> likeName(String name){  
  ***PageHelper.startPage(1,2);***  
 **return demoService**.likeName(name);  
}

## 3获取自增id

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