**《Sprig Boot 2.x》**

**SpringBoot入门：框架介绍与HelloWorld**

## 课程主要内容

* MVC架构思想
* 使用STS构建SpringBoot项目
* 使用SpringBoot构建Mvc web项目
* MVCWeb项目中的注入
* 热部署

# 介绍

SpringBoot主要解决的是在微服务的架构下简化配置（有快速配置）、前后端分离、快速开发

优点：

* 提供了快速启动入门
* 开箱即用、提供默认配置
* 内嵌容器化web项目
* 没有冗余代码生成和xml配置要求

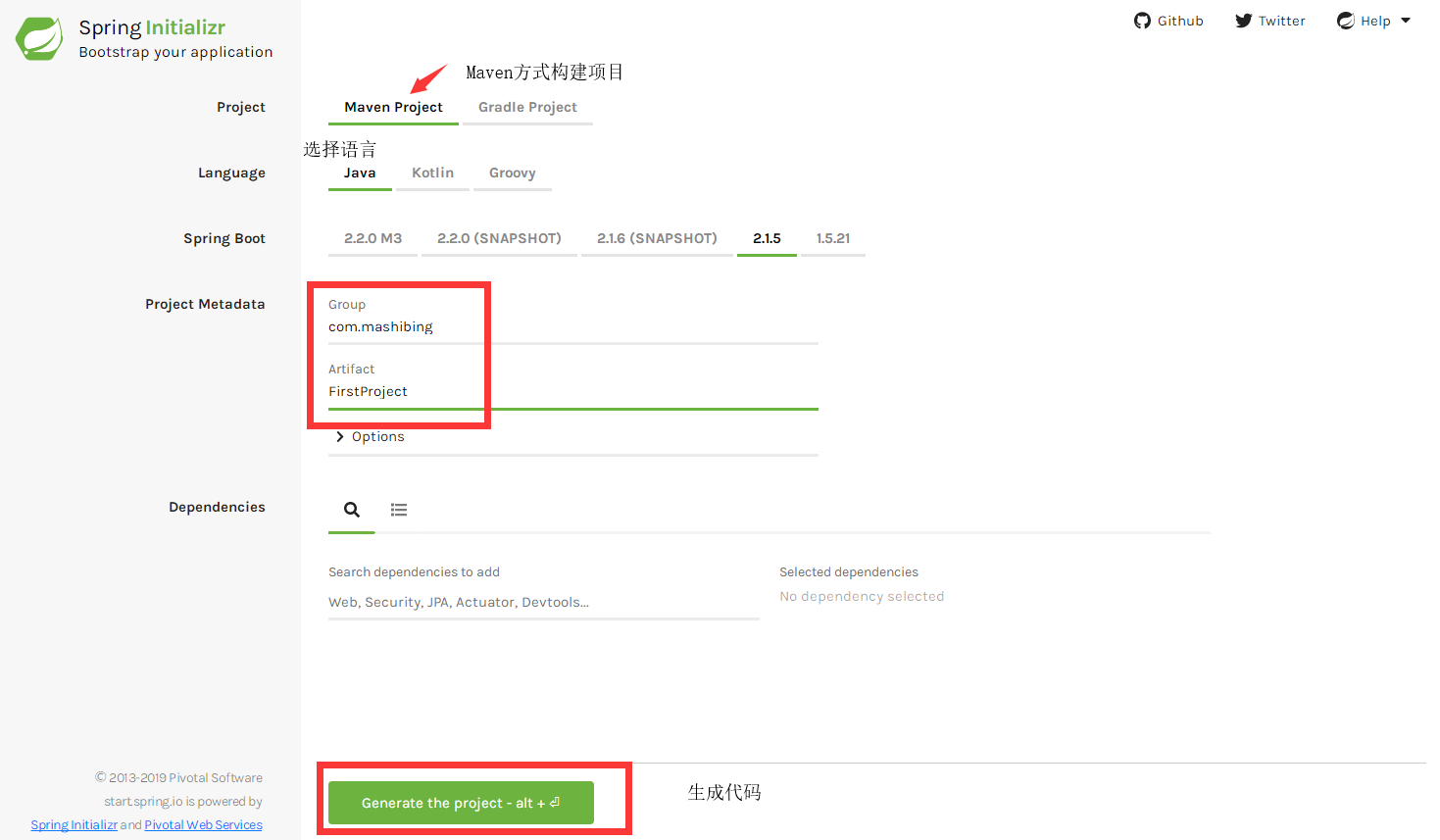
## 2.运行Demo

### 创建项目

创建SpringBoot项目的几种方式：

* 官网的Initializr
* 使用Eclipse、STS、Idea等IDE创建Maven项目并引入依赖
* 使用STS插件的Spring Initializr创建项目

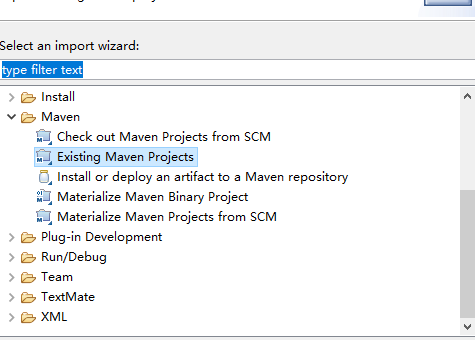
访问<http://start.spring.io/> 进入Spring项目Initializr



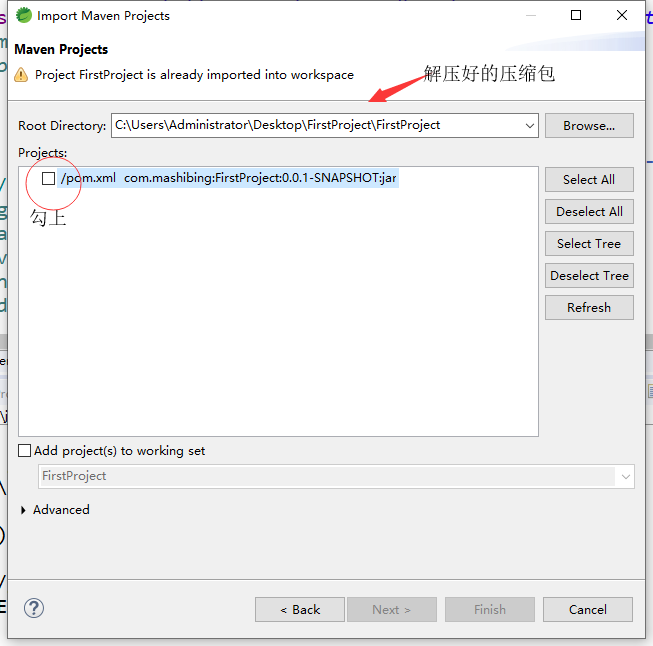
生成下载demo.zip

### 导入项目

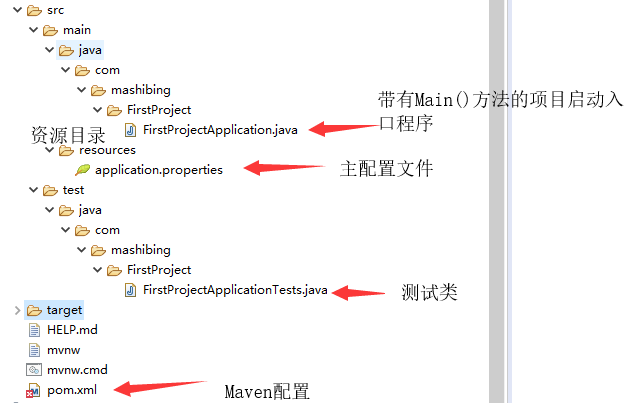
#### 1.Import一个Maven项目



#### 2.选择要导入的文件



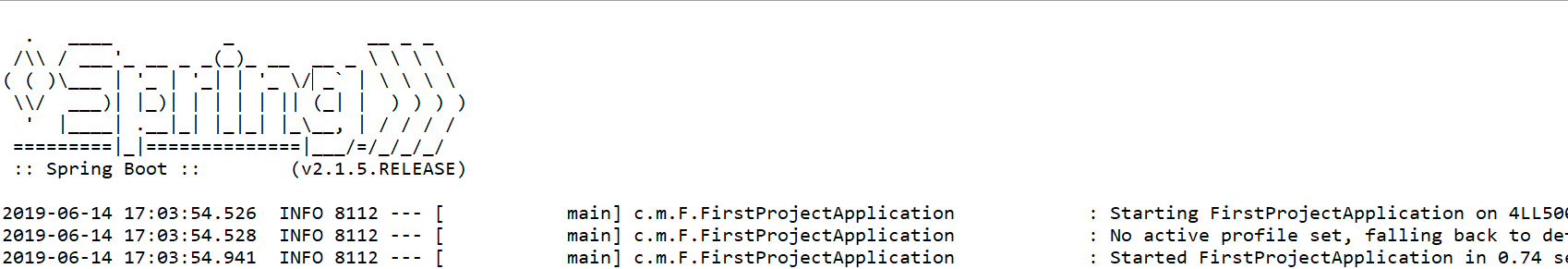
#### 3.项目骨架



### 启动项目

* 直接run启动程序里的Main（）方法
* 安装过STS插件或使用STS可以在项目上右键RunAS->Spring Boot APP

运行成功提示信息：



如果运行报错，请参照常见问题。

### 热部署

#### devtools 热部署

pom.xml文件中添加依赖：

<dependency>

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

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

<optional>true</optional>

<scope>true</scope>

</dependency>

在application.yml中配置一下devtools：

spring:

devtools:

restart:

enabled: true #设置开启热部署

additional-paths: src/main/java #重启目录

exclude: WEB-INF/\*\*

freemarker:

cache: false #页面不加载缓存，修改即时生效

#### 2、jrebel

#### jrebel是一款JVM插件，它使得Java代码修改后不用重启系统，立即生效

## 个性化

### 修改启动banner

在resources目录下新建banner.txt

<http://www.network-science.de/ascii/> 英文

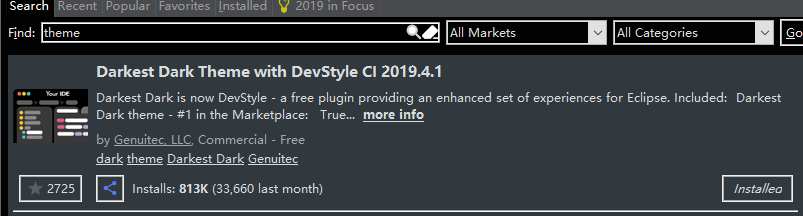
<https://www.degraeve.com/img2txt.php> 图片

### Eclipse的皮肤

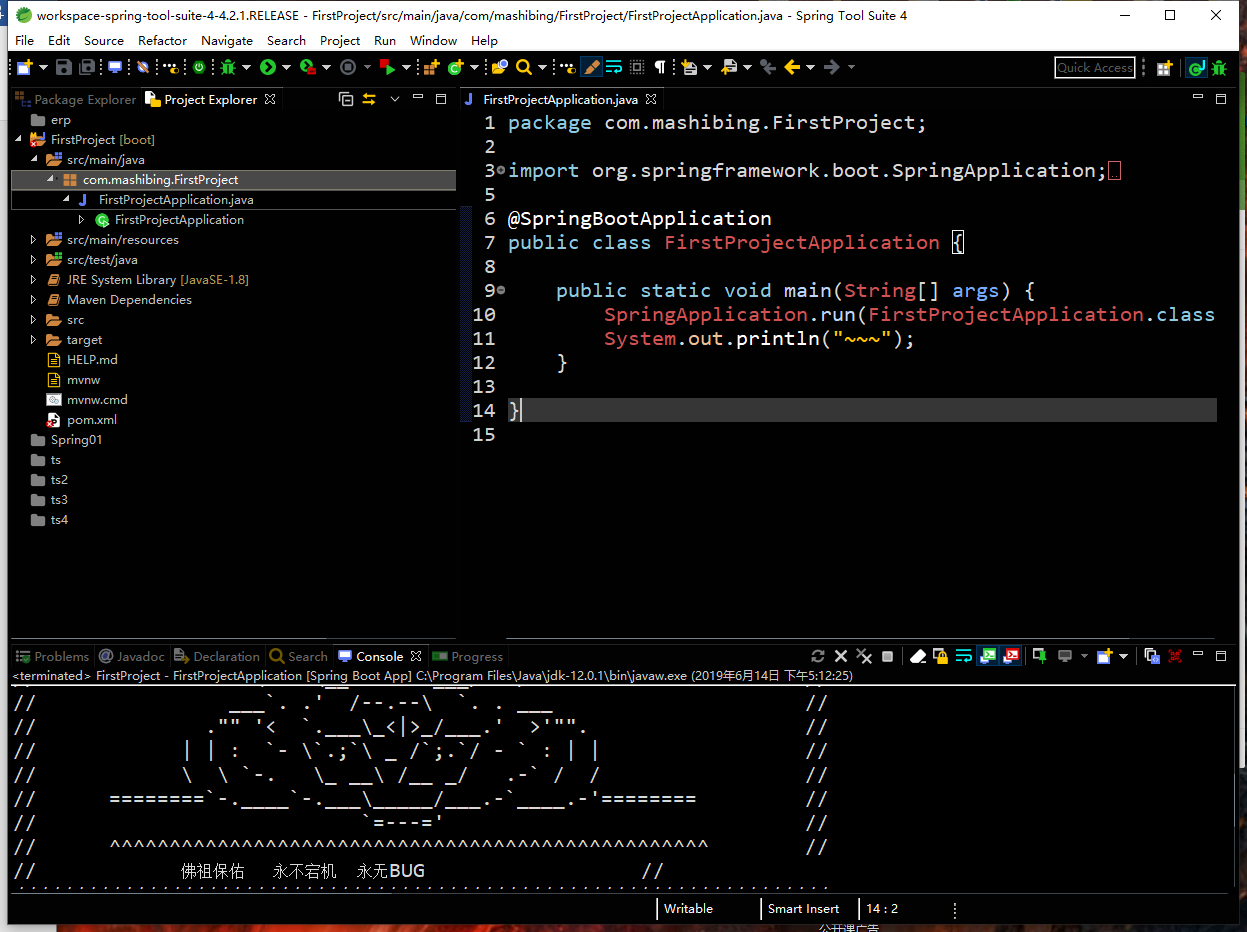
菜单栏中

Help -> EclipseMarketplace

搜索Theme



傻瓜式安装这个，安装完成会提示重启，跟随指引选择喜欢的风格。



## 简单使用

### application.properties

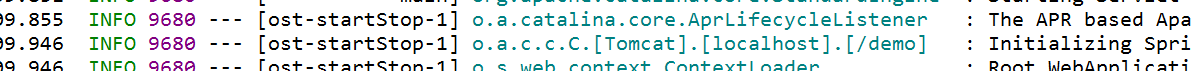
把所有的配置全放在这个文件里，方便统一管理，maven也可以做到

### 修改tomcat端口

server.port=90

### 修改项目路径

server.servlet.context-path=/demo



### 多个入口main方法，打包之后找不到入库类

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<mainClass>com.mashibing.MyApp</mainClass>

</configuration>

</plugin>

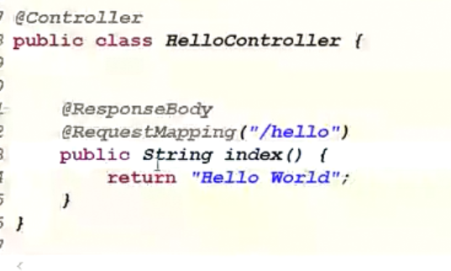
</plugins>

</build>

### HelloWorld

#### RestController

RestController = @Controller+@ResponseBody



一个效果

@RestController

**public** **class** MyAppController {

@RequestMapping("/")

**public** Map<String, String> index() {

Map<String, String> map = **new** HashMap<>();

map.put("aaa", "bbb");

map.put("aaa", "bbb");

map.put("aaa", "bbb");

map.put("aaa", "bbb");

**return** map;

}

#### 使用thymeleaf模板引擎

##### Pom.xml引用

<dependency>

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

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

</dependency>

##### Controller代码

@Controller

**public** **class** IndexController {

@RequestMapping("/")

**public** String index(ModelMap map) {

// 加入一个属性，用来在模板中读取

map.addAttribute("msg", "nihao~");

return模板文件的名称，对应src/main/resources/templates/index.html

**return** "index";

}

##### 模板文件代码

<h1 th:text="${msg}">hi!</h1>

### 稍微复杂的restful api应用

#### UserRestfulController

@RequestMapping("/")

@RestController

**public** **class** UserRestfulController {

**static** Map<Long, User> *users* = Collections.*synchronizedMap*(**new** HashMap<Long,User>());

@RequestMapping(value="/User",method=RequestMethod.***GET***)

**public** List<User> getUserList(){

ArrayList<User> list = **new** ArrayList<>(*users*.values());

**return** list;

}

@RequestMapping(value="User",method=RequestMethod.***POST***)

**public** String addUser(@ModelAttribute User user) {

*users*.put(user.getId(), user);

**return** "addUser Success";

}

}

#### User

**public** **class** User {

**private** Long id;

**private** String loginName;

**private** String password;

**private** String nickName;

### 注入Service

#### UserRestfulController

@Autowired

**private** UserService userSrv;

@RequestMapping(value="/User",method=RequestMethod.***GET***)

**public** List<User> getUserList(){

**return** userSrv.getUserList();

}

@RequestMapping(value="User",method=RequestMethod.***POST***)

**public** String addUser(@ModelAttribute User user) {

String msg = userSrv.addUser(user);

**return** msg;

}

#### UserService

@Service

**public** **class** UserService {

**static** Map<Long, User> *users* = Collections.*synchronizedMap*(**new** HashMap<Long,User>());

**public** List<User> getUserList() {

ArrayList<User> list = **new** ArrayList<>(*users*.values());

**return** list;

}

**public** String addUser(User user) {

*users*.put(user.getId(), user);

**return** "addUser Success";

}

}

### 前端模板显示

<h1>User list</h1>

<table>

<tr>

<th>NAME</th>

<th>loginName</th>

<th>nickName</th>

</tr>

<tr th:each=*"user : ${list}"*>

<td th:text=*"${user.id}"*>id</td>

<td th:text=*"${user.loginName}"*>loginName</td>

<td th:text=*"${user.nickName}"*>nickName</td>

</tr>

</table>

<p>

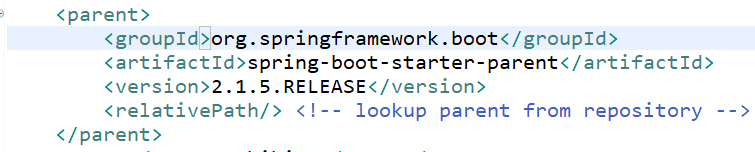
<a href=*"../home.html"* th:href=*"@{/}"*>Return to home</a>

</p>

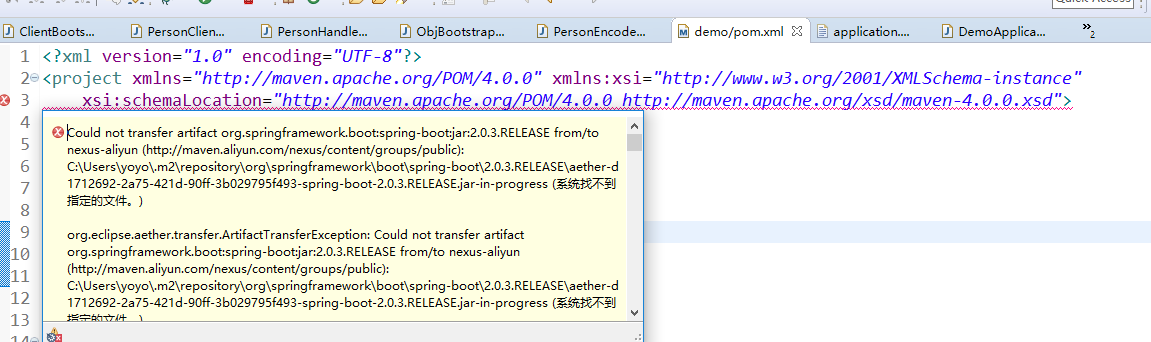
## 常见问题

### Pom.xml

Springboot项目必须要继承的parnet

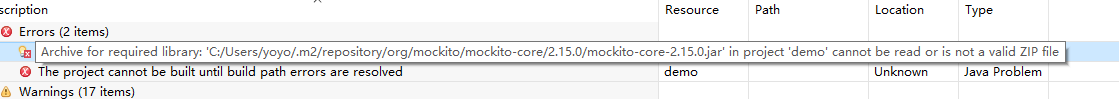


### Pom.xml第一行报错



在cmd中进入本地仓库执行: for /r %i in (\*.lastUpdated) do del %i ，然后执行Maven->“Update Project …”

缺少或包错误



删掉 重新update

### 找不到主类



所有错误都解决后

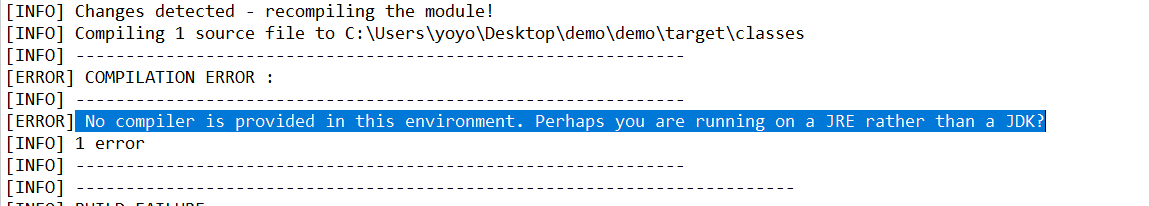
Jar方式运行 首先得有这个jar包

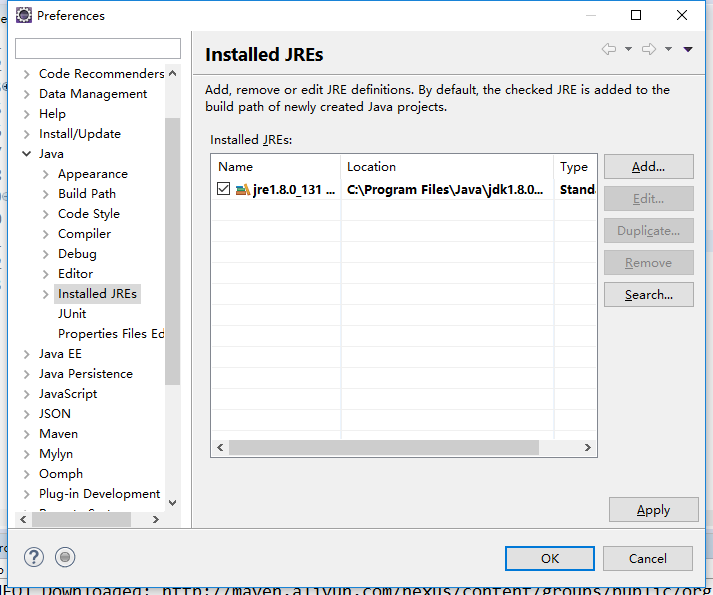
先clean package 生成jar文件，然后再run main方法

## 课程主要内容

* MVC架构思想
* 使用STS构建SpringBoot项目
* 使用SpringBoot构建Mvc web项目
* MVCWeb项目中的注入
* 热部署

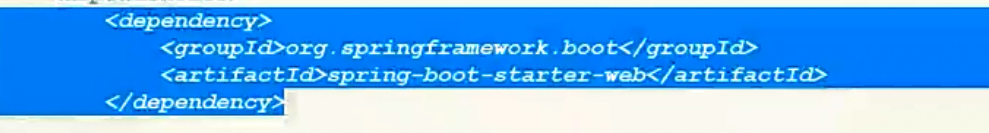
### 找不到jdk





把jre的路径换成jdk的

### 启动后自动停止



# 静态资源访问

src/main/resources/static

src/main/webapp

# 视图层整合

## Jsp

### Pom

<!-- jstl -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<!-- jasper -->

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

### Properties

spring.mvc.view.prefix=/WEB-INF/jsp/

spring.mvc.view.suffix=.jsp

### jsp

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<!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>

<table border="1" align="center" width="50%">

<tr>

<th>ID</th>

<th>Name</th>

<th>Age</th>

</tr>

<c:forEach items="${list }" var="user">

<tr>

<td>${user.userid }</td>

<td>${user.username }</td>

<td>${user.userage }</td>

</tr>

</c:forEach>

</table>

</body>

</html>

### Controller

@Controller

public class UserController {

/\*

\* 处理请求，产生数据

\*/

@RequestMapping("/showUser")

public String showUser(Model model){

List<Users> list = new ArrayList<>();

list.add(new Users(1,"张三",20));

list.add(new Users(2,"李四",22));

list.add(new Users(3,"王五",24));

//需要一个Model对象

model.addAttribute("list", list);

//跳转视图

return "userList";

}

# 整合DAO层

## Spring-data-jpa

## Entity

<h1>User list</h1>

<table>

<tr>

<th>NAME</th>

<th>loginName</th>

<th>nickName</th>

</tr>

<tr th:each=*"city : ${list}"*>

<td th:text=*"${city.id}"*>id</td>

<td th:text=*"${city.name}"*>loginName</td>

</tr>

</table>

<p>

<a href=*"../home.html"* th:href=*"@{/}"*>Return to home</a>

</p>

## Repository

**public** **interface** CityRepository **extends** JpaRepository<City, Integer> {

}

## Properties

##端口号

server.port=8888

##数据库配置

##数据库地址

spring.datasource.url=jdbc:mysql://localhost:3306/erp?characterEncoding=utf8&useSSL=false&serverTimezone=UTC

##数据库用户名

spring.datasource.username=root

##数据库密码

spring.datasource.password=840416

##数据库驱动

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

##validate 加载hibernate时，验证创建数据库表结构

##create 每次加载hibernate，重新创建数据库表结构，这就是导致数据库表数据丢失的原因。

##create-drop 加载hibernate时创建，退出是删除表结构

##update 加载hibernate自动更新数据库结构

##validate 启动时验证表的结构，不会创建表

##none 启动时不做任何操作

#spring.jpa.hibernate.ddl-auto=validate

##控制台打印sql

spring.jpa.show-sql=true

spring.devtools.restart.enabled: true

spring.thymeleaf.cache=false

spring.thymeleaf.encoding=UTF-8

spring.http.encoding.force=true

spring.http.encoding.charset=UTF-8

spring.http.encoding.enabled=true

server.tomcat.uri-encoding=UTF-8

spring.mvc.date-format=yyyy-MM-dd

## pom

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

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

</dependency>

## Html

<h1>User list</h1>

<table>

<tr>

<th>NAME</th>

<th>loginName</th>

<th>nickName</th>

</tr>

<tr th:each=*"city : ${list}"*>

<td th:text=*"${city.id}"*>id</td>

<td th:text=*"${city.name}"*>loginName</td>

</tr>

</table>

<p>

<a href=*"../home.html"* th:href=*"@{/}"*>Return to home</a>

</p>

# SpringBoot整合

**《Sprig Boot 2.x》**

**SpringBoot入门：框架介绍与HelloWorld**

## 课程主要内容

* SpringDataJpa进阶使用
* SpringDataJpa自定义查询
* 整合Servlet、Filter、Listener
* 文件上传
* Thymeleaf常用标签

## 整合Servlet

### 注解方式

#### 启动类上添加注解

@SpringBootApplication

@ServletComponentScan

**public** **class** Springboot011Application {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(Springboot011Application.**class**, args);

}

}

#### Servlet类

@WebServlet(name = "myServlet",urlPatterns = "/srv",loadOnStartup = 1)

**public** **class** MyServlet **extends** HttpServlet {

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

System.***out***.println("111");

**super**.doGet(req, resp);

}

}

### 编码方式

#### 启动类中添加

@Bean

**public** ServletRegistrationBean<MyServlet2> getServletRegistrationBean(){

ServletRegistrationBean<MyServlet2> bean = **new** ServletRegistrationBean<>(**new** MyServlet2(), "/s2");

bean.setLoadOnStartup(1);

**return** bean;

}

#### Servlet

无需注解

## 整合Filter

**implements** Filter

过滤器

同Servlet

## Listener

监听器

MyListener **implements** ServletContextListener

同上

# 静态资源访问

src/main/resources/static

src/main/webapp

# SpringBoot文件上传

## 指定大小

spring.http.multipart.maxFileSize=200MB

spring.http.multipart.maxRequestSize=200MB

spring.servlet.multipart.max-request-size = 200MB

spring.servlet.multipart.max-file-size = 200MB

## 表单

<form action="fileUploadController" method="post" enctype="multipart/form-data">

上传文件：<input type="file" name="filename"/><br/>

<input type="submit"/>

</form>

## Controller

@RequestMapping("/fileUploadController")

**public** String fileUpload(MultipartFile filename) **throws** Exception{

System.***out***.println(filename.getOriginalFilename());

filename.transferTo(**new** File("e:/"+filename.getOriginalFilename()));

**return** "ok";

}

# 表单数据接收

@GetMapping(value = "/hello/{id}")

public String hello(@PathVariable("id") Integer id){

return "ID:" + id;

}

@PostMapping(value = "/user-2")

public User saveUser2(@RequestBody User user) {

return user;

}

@PostMapping(value = "/post")

public String post(@RequestParam(name = "name") String name,

@RequestParam(name = "age") Integer age) {

String content = String.format("name = %s,age = %d", name, age);

return content;

}

# SpringData Jpa进阶使用

## 显示SQL

spring.jpa.show-sql=true

## 自定义查询

### 自定义接口

* And --- 等价于 SQL 中的 and 关键字，比如 findByUsernameAndPassword(String user, Striang pwd)；
* Or --- 等价于 SQL 中的 or 关键字，比如 findByUsernameOrAddress(String user, String addr)；
* Between --- 等价于 SQL 中的 between 关键字，比如 findBySalaryBetween(int max, int min)；
* LessThan --- 等价于 SQL 中的 "<"，比如 findBySalaryLessThan(int max)；
* GreaterThan --- 等价于 SQL 中的">"，比如 findBySalaryGreaterThan(int min)；
* IsNull --- 等价于 SQL 中的 "is null"，比如 findByUsernameIsNull()；
* IsNotNull --- 等价于 SQL 中的 "is not null"，比如 findByUsernameIsNotNull()；
* NotNull --- 与 IsNotNull 等价；
* Like --- 等价于 SQL 中的 "like"，比如 findByUsernameLike(String user)；
* NotLike --- 等价于 SQL 中的 "not like"，比如 findByUsernameNotLike(String user)；
* OrderBy --- 等价于 SQL 中的 "order by"，比如 findByUsernameOrderBySalaryAsc(String user)；
* Not --- 等价于 SQL 中的 "！ ="，比如 findByUsernameNot(String user)；
* In --- 等价于 SQL 中的 "in"，比如 findByUsernameIn(Collection<String> userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；
* NotIn --- 等价于 SQL 中的 "not in"，比如 findByUsernameNotIn(Collection<String> userList) ，方法的参数可以是 Collection 类型，也可以是数组或者不定长参数；

### 自定义SQL@Query

#### 占位符

public interface UserDao extends Repository<AccountInfo, Long> {

@Query("select a from AccountInfo a where a.accountId = ?1")

public AccountInfo findByAccountId(Long accountId);

   @Query("select a from AccountInfo a where a.balance > ?1")

public Page<AccountInfo> findByBalanceGreaterThan(

Integer balance,Pageable pageable);

}

#### 参数名

public interface UserDao extends Repository<AccountInfo, Long> {

public AccountInfo save(AccountInfo accountInfo);

@Query("from AccountInfo a where a.accountId = :id")

public AccountInfo findByAccountId(@Param("id")Long accountId);

  @Query("from AccountInfo a where a.balance > :balance")

  public Page<AccountInfo> findByBalanceGreaterThan(

@Param("balance")Integer balance,Pageable pageable);

}

#### 更新

@Modifying

@Query("update AccountInfo a set a.salary = ?1 where a.salary < ?2")

public int increaseSalary(int after, int before);

### 直接使用Native SQL

public interface UserRepository extends JpaRepository<User, Long> {

@Query(value = "SELECT \* FROM USERS WHERE EMAIL\_ADDRESS = ?1", nativeQuery = true)

User findByEmailAddress(String emailAddress);

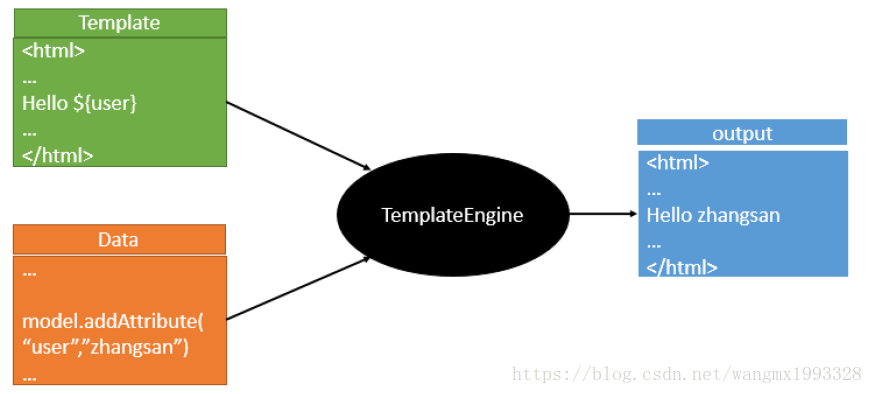
}

# Thymeleaf

## 编辑插件

安装地址：

<html xmlns:th=*"http://www.thymeleaf.org"*>



## URL地址

在应用上下文内

1）@{userList} 相对当前路径结果为：http://localhost/thymeleaf/user/userList

2）@{./userList} 相对当前路径结果为：http://localhost/thymeleaf/user/userList

3）@{../tiger/home} 相对当前路径结果为：http://localhost/thymeleaf/tiger/home

4）@{/tiger/home} 相对应用根目录结果为：http://localhost/thymeleaf/tiger/home

5）@{https://www.baidu.com/} 绝对路径结果为：https://www.baidu.com

6）<link type="text/css" rel="stylesheet" th:href="@{/css/home.css}">，@ 以 "/" 开头相对应用根目录，否则是相对当

## Href

<body>

<a th:href="@{userList(id=9527)}">1、@{userList(id=9527)}</a>

<a th:href="@{userList(id=9527,name=华安)}">2、@{userList(id=9527,name=华安)}</a>

<a th:href="@{userList(id=9527,name=${userName})}">3、@{userList(id=9527,name=${userName})}</a>

</body>

## 文本

<!--空格属于特殊字符，必须使用单引号包含整个字符串-->

<p class=*"css1 css2"* th:class=*"'css1 css2'"*>样式</p>

<!--下面如果没用单引号 th:text="Big China"，则页面直接报错-->

<p th:text=*"'Big China'"*>中国</p>

<!--后台使用：model.addAttribute("info", "Love you 中国"); 传值有空格也是没有问题的-->

<p th:text=*"${userName}"*>userName</p>

<!--后台传值字符串有空格是可以的，可以使用 + 进行字符串连接-->

<p th:text=*"'small smile'+',very good.' + ${userName}"*>浅浅的微笑</p>

## 数字计算

<p th:text=*"80"*>8</p>

<!--计算结果为 16 在进行替换-->

<p th:text=*"8+8"*>8 + 8</p>

<!--前面 8+8 计算结果为 16，然后字符串拼接上 Love，后面的 9+9也会被当做字符串拼接-->

<p th:text=*"8+8+' Love '+9+9"*>8 + 8+' Love '+9+9</p>

<!--前面 8+8 计算结果为 16，后面的 9+9因为有括号,所以也会计算结果，最后拼接 Love 字符串-->

<p th:text=*"8+8+' Love '+(9+9)"*>8 + 8+' Love '+(9+9)</p>

<!--后台传了一个：model.addAttribute("age", 35);取得结果后在进行计算-->

<p th:text=*"100-${age}"*></p>

## Boolean判断

<p th:text=*"true"*>布尔</p>

<!--true、false 是布尔值，and 是布尔运行符，and(与)，or(或)，not(非)、!(非)-->

<p th:text=*"true and false"*>true and true</p>

<!--后台使用 model.addAttribute("isMarry", true); 传了值-->

<!--th:if 表达式为 true，则显示标签内容，否则不显示-->

<p th:if=*"${isMarry}"*>已结婚</p>

<!--后台传值：model.addAttribute("age", 35);-->

<!--比较运算符：&gt;，&lt;，&gt; =，&lt;=（gt，lt，ge，le）-->

<p th:if=*"${age}&gt;18"*>已成年</p>

<p th:if=*"${age}&lt;18"*>未成年</p>

## 运算

<p th:text=*"15 \* 4"*>值为 60 </p>

<p th:text=*"15 \* 4-100/10"*>值为 50 </p>

<p th:text=*"100 % 8"*>值为 4</p>

## 比较

<p th:if=*"5>3"*>5 大于 3</p>

<p th:if=*"5 &gt;4"*>5 大于 4</p>

<p th:if=*"10>=8 and 7 !=8"*>10大于等于8，且 7 不等于 8 </p>

<p th:if=*"!${isMarry}"*>!false</p>

<p th:if=*"not(${isMarry})"*>not(false)</p>

## 三元运算符

<p th:text=*"7&gt;5?'7大':'5大'"*>三元运算符</p>

<!--后台控制器输出了：model.addAttribute("age", 35);-->

<!--因为 ${xx}取值时，如果值为null，则默认整个标签不再显示-->

<p th:text=*"${age}!=null?${age}:'age等于 null'"*></p>

<!--这里使用嵌套判断，嵌套的部分要使用括号-->

<p th:text=*"${age}!=null?(${age}>=18?'成年':'未成年'):'age等于 null'"*></p>

<!--变量 age2 后台并没有输出，所以 age2 不存在，此时 age2 ==null-->

<p th:text=*"${age2}!=null?${age2}:'age2等于 null'"*></p>

<!--后台输出了：model.addAttribute("isMarry", true);-->

<!--A>B?X:Y，这里的 Y 部分是省略的，此时如果 A>B 返回 false，则整个三元运算结果为 null-->

<p th:class=*"${isMarry}?'css2':'css3'"*>已婚</p>

## TH标签

### th:utext转义

map .addAttribute("china", "<b>Chian</b>,USA,UK");

<p th:text=*"${china}"*>默认转义</p>

<!--th:utext 不会结果进行转义-->

<p th:utext=*"${china}"*>不会转义</p>

### th:attr 设置属性

HTML5 所有的属性，都可以使用 th:\* 的形式进行设置值

<a href="http://baidu.com" th:attr="title='百度'">百度</a>

<!--设置 title、href 多个属性-->

<a href="" th:attr="title='前往百度',href='http://baidu.com'">前往百度</a>

<!--设置 href 属性-->

<a href="userList.html" th:attr="href=@{/user/userHome}">用户首页</a>

<!--设置 id 属性，data-schoolName 属性 Html 本身是没有的，但允许用户自定义 -->

<a href="#" th:attr="id='9527',data-target='user'">归海一刀</a>

<p th:abc=*"9527"*>th:abc="9527"</p>

<!--输出：<p abc123="华安">th:abc123="华安"</p>-->

<p th:abc123=*"华安"*>th:abc123="华安"</p>

### Checked selected

<input type=*"checkbox"* name=*"option1"* checked/><span>是否已婚1？</span>

<input type=*"checkbox"* name=*"option2"* checked=*"checked"*/><span>是否已婚2？</span>

---<br>

<!--后台控制器传递了一个：model.addAttribute("isMarry", true);-->

<!--option3、option4 会选中；option5 不会选中-->

<input type=*"checkbox"* name=*"option3"* th:checked=*"${isMarry}"*/><span>是否已婚3？</span>

<input type=*"radio"* name=*"option4"* th:checked=*"${isMarry}"*/><span>是否本科？</span>

<input type=*"radio"* name=*"option5"* th:checked=*"!${isMarry}"*/><span>是否应届生？</span>

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

<select>

<option>a</option>

<option th:selected=*"${isMarry}"*>已婚</option>

<option th:selected=*"${!isMarry}"*>未婚</option>

</select>

<input type=*"text"* th:autofocus=*"false"*>

<input type=*"text"* th:autofocus=*"true"*>

<input type=*"text"* th:autofocus=*"false"*>

## 日期格式化

    <span th:text=*"${#dates.format(date, 'yyyy-MM-dd HH:mm')}"*></span>

## 循环

JSTL 有一个 <c:foreach>，同理 Thymeleaf 也有一个 th:each。作用都是一样的，都是用于遍历数组、List、Set、Map 等数据。

### 在Select上循环

<option th:each="city : ${list}" th:text="${city.name}" th:selected="${cityName} eq ${city.name}">横岗</option>

### 状态变量loopStatus

如果不指定 为变量+Stat

* index: 当前迭代对象的index（从0开始计算）
* count: 当前迭代对象的index(从1开始计算)
* size: 被迭代对象的大小 current:当前迭代变量
* even/odd: 布尔值，当前循环是否是偶数/奇数（从0开始计算）
* first: 布尔值，当前循环是否是第一个
* last: 布尔值，当前循环是否是最后一个

<tr th:each="city,status : ${list}" th:style="${status.odd}?'background-color:#c2c2c2'">

<!-- EL JSTL-->

<td th:text = "${status.count}"></td>

<td th:text = "${city.id}"></td>

<td th:text = "${city.name}"></td>

</tr>

## If/else

<p th:if=*"${isMarry}"*>已婚1</p>

<p th:unless=*"${isMarry}"*>未婚</p>

## Switch/case 多条件判断

<div th:switch=*"1"*>

<p th:case=*"0"*>管理员</p>

<p th:case=*"1"*>操作员</p>

<p th:case=*"\*"*>未知用户</p>

</div>

<!--数字类型：当没有 case 匹配时，取默认值，当有多个匹配，只取第一个-->

<div th:switch=*"-1"*>

<p th:case=*"0"*>管理员</p>

<p th:case=*"\*"*>操作员</p>

<p th:case=*"\*"*>未知用户</p>

</div>

<!--布尔类型，多个case满足时，只取第一个-->

<div th:switch=*"${isMarry}"*>

<p th:case=*"true"*>已婚</p>

<p th:case=*"true"*>已成年</p>

<p th:case=*"false"*>未婚</p>

</div>

<!--字符串类型-->

<div th:switch=*"'For China'"*>

<p th:case=*"'For USA'"*>美国</p>

<p th:case=*"'For UK'"*>英国</p>

<p th:case=*"'For China'"*>中国</p>

<p th:case=*"\*"*>未知国籍</p>

</div>

## 内联表达式

[[...]] 等价于 th:text（结果将被 HTML 转义），[(...)] 等价于 th:utext（结果不会执⾏HTML转义）

<p>[[${china}]]</p>

<p>[(${china})]</p>

<p>[[Lo1ve]]</p>

<p>[['I Love You Baby']]</p>

<p>[(9527)]</p>

th:inline =“none” 来禁⽤内联。

### 内联 JavaScript

<script type="text/javascript" th:inline="javascript">

**var** info = [[${info}]];

**var** age = [[${age}]];

**var** id = [[${id}]];

**var** name = [[${name}]];

console.log(id, name, age, info);

</script>

### 前后端分离

<script type="text/javascript" th:inline="javascript">

/\*\*

\* Thymeleaf 将自动忽略掉注释之后 和 分号之前的所有内容,如下为 "前端测试"

\*/

**var** info = /\*[[${info}]]\*/ "前端测试";

console.log(info);

</script>

## 对象属性

### URL

<p>${param.size()}=[[${param.size()}]]</p>

<!--/\*判断请求参数是否为空\*/-->

<p>${param.isEmpty()}=[[${param.isEmpty()}]]</p>

<!--获取某个参数值，不存在时为null-->

<p>${param.u\_id}=[[${param.u\_id}]]</p>

### Session

<p>${session.size()}=[[${session.size()}]]</p>

<!--/\*判断请求参数是否为空\*/-->

<p>${session.isEmpty()}=[[${session.isEmpty()}]]</p>

<!--获取某个参数值，不存在时为null-->

<p>${session.user.id}=[[${session.user.id}]]</p>

## 完整文档

https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#base-objects