SpringBoot教程

一、使用idel直接创建SpringBoot

> 搭建网站的授权管理框架
>
> https://start.aliyun.com国内代理

二、SpringBoot入门使用

2.1创建maven项目,在父模块中导入依赖

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.5.RELEASE</version>
    </parent>
    <dependencyManagement>
        <dependencies>
            <!--springboot版本管理-->
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>2.0.5.RELEASE</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>
```

2.2创建普通子模块,在子模块中导入依赖

```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
</dependencies>
```

2.3编码测试

> 新建一个Controller类

```
    //@RestController(类上)=@Controller(类上)+@RequestMapper(方法上)
    @RestController
public class HelloController {
    @RequestMapping("/hello")
    public String hello(){
    return "hello";
    }
}
```

新建启动类(App – Main方法)

```
@SpringBootApplication
public class SpringbootApp {
public static void main(String[] args) {
SpringApplication.run(SpringbootApp.class);
}
}
```

三、SpringBoot的Web

> 静态资源放在resource是下的 public resource static的包下
> templates包下放静态页面

❗注意

> 在templates下的页面只能通过controller来跳转

3.1首页跳转

```
@RestController
@RequestMapping("test")
public class HelloController {
    @RequestMapping("/hello")
    public String hello(){
        return "hello_springBoot";
```

```
    }
}
```

四、SpringBoot的启动原理

4.1 自动配置：pom.xml

> springboot的核心依赖在父工程中
> 在配置文件中使用，不需要指定版本

五、Thymeleaf的语法

> 在页面中插入数据使用 org.springframework.ui里面的Modle接口

5.1Thymeleaf 模板引擎支持多种表达式：

- 变量表达式：${...}
- 选择变量表达式：*{...}
- 链接表达式：@{...}
- 国际化表达式：#{...}
- 片段引用表达式：~{...}

5.2 变量表达式

> 使用 ${} 包裹的表达式被称为变量表达式，该表达式具有以下功能：
> 获取对象的属性和方法
> 使用内置的基本对象
> 使用内置的工具对象

```
public class indexController {
    @RequestMapping("/")
    public String IndexPage(Model model){
        //将
        model.addAttribute("msg","<h1>hehe</h1>");
        model.addAttribute("users", Arrays.asList("zhangsan","lisi","kuangsheng"));
        return "index";
    }
}
```

5.3 关闭模板引擎的缓存

```
## Thymeleaf的提示语
<html lang="en" xmlns:th="http://www.thymeleaf.org">
```

```
## 取出数据
<div th:text="${msg}"></div>     取出文本

<div th:utext="${msg}"></div>  取出文本并使用html

## 遍历
<div th:each="user:${users}" th:text="${user}"></div>
```

```
## 遍历对象
<div th:object="${user1}">
    <p th:text="*{name}"></p>
    <p th:text="*{id}"></p>
</div>
```

5.4 Thymeleaf的语法总结

寻找项目下的根目录的资源

@{/}

属性	描述	示例
th:id	替换 HTML 的 id 属性	<input id="html-id" th:id="thymeleaf-id" />
th:text	文本替换，转义特殊字符	<h1 th:text="hello,bianchengbang" >hello</h1>
th:utext	文本替换，不转义特殊字符	<div th:utext="' 欢迎来到编程帮！ '" >欢迎你

th:object	在父标签选择对象，子标签使用 *{...} 选择表达式选取值。 没有选择对象，那子标签使用选择表达式和 ${...} 变量表达式是一样的效果。 同时即使选择了对象，子标签仍然可以使用变量表达式。	<div th:object="${session.user}" > firstname
th:value	替换 value 属性	<input th:value = "${user.name}" />
th:with	局部变量赋值运算	<div th:with="isEvens = ${prodStat.count}%2 == 0" th:text="${isEvens}">
th:style	设置样式	编程帮 www.biancheng.net
th:onclick	点击事件	<td th:onclick = "'getInfo()'"></td>
th:each	遍历，支持 Iterable、Map、数组等。	<table> <tr th:each="m:${session.map}"> <td th:text="${m.getKey()}"> </td> <td th:text="${m.getValue()}"> </td> </tr></table>
th:if	根据条件判断是否需要展示此标签	<a th:if ="${userId == collect.userId}">
th:unless	和 th:if 判断相反，满足条件时不显示	
th:switch	与 Java 的 switch case语句类似 通常与 th:case 配合使用，根据不同的条件展示不同的内容	编程帮 www.biancheng.net
th:fragment	模板布局，类似 JSP 的 tag，用来定义一段被引用或包含的模板片段	<footer th:fragment="footer">插入的内容</footer>
th:insert	布局标签； 将使用 th:fragment 属性指定的模板片段（包含标签）插入到当前标签中。	
th:replace	布局标签；	<div

	使用 th:fragment 属性指定的模板片段（包含标签）替换当前整个标签。	th:replace="commons/bar::footer">
th:selected	select 选择框选中	<select> <option>---</option> <option th:selected="${name=='a'}">编程帮 </option> <option th:selected="${name=='b'}">www.biancheng.net</option></select>
th:src	替换 HTML 中的 src 属性	
th:inline	内联属性； 该属性有 text、none、javascript 三种取值， 在 <script> 标签中使用时，js 代码中可以获取到后台传递页面的对象。	<script type="text/javascript" th:inline="javascript"> var name = /*[[${name}]]*/ 'bianchengbang'; alert(name)</script>
th:action	替换表单提交地址	<form th:action="@{/user/login}" th:method="post"></form>

六、MVC配置原理

6.1 拓展MVC

> 拓展springMVC
> 如果你想保持MVC的配置并拓展相关功能
> @EnableWebMvc 全面接管SpringMVC ---不要加此注解
> ViewResolver --SpringBoot自动装配的 如果我们需要接管视图解析器需要我们重写

6.2 添加一个拦截器

```
1 //拓展springMVC
2 //如果你想保持MVC的配置并拓展相关功能
3 @Configuration
```

```java
public class MyMvcConfig implements WebMvcConfigurer {
    /**
     * 添加拦截器
     * @param registry
     */
    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        //super.addInterceptors(registry);
        //静态资源;   *.css , *.js
        //SpringBoot已经做好了静态资源映射
        registry.addInterceptor(new LoginHandlerInterceptor()).addPathPatterns("/**")
            .excludePathPatterns("/index.html","/","/user/login","/asserts/**",
            "/**/asserts/**/*.js","/asserts/img/**","/asserts/js/**","/webjars/**");
    }
}

```

6.3 添加自定义的视图解析器

```java
/**
 * 添加一个自己的视图解析器
 * @return
 */
@Bean
public ViewResolver myViewResolver(){
    return new MyViewResolver();
}
//定义自己视图解析器的功能
public  static class MyViewResolver implements ViewResolver{
    //写自己视图器的功能
    @Override
    public View resolveViewName(String s, Locale locale) throws Exception {
        return null;
    }
}
```

6.4 添加自定义的格式化

> 默认的格式化 dd/MM/yyyy

> 在yaml中配置 spring.mvc.date-format = ...

6.5 实现视图跳转

> 还是在springMVC的拓展中重写addViewController方法，定义自己的视图跳转

```
@Configuration
public class MyMvcConfig implements WebMvcConfigurer {
    /**
     * 添加首页控制
     * @param registry
     */
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
          registry.addViewController("/zou").setViewName("login");
    }
}
```

七、页面的国际化

7.1国际化的基本配置

> 确保项目的编码是UTF8
>
> 取出国际化的方法是
>
> #{...}

7.2 在resources文件夹中建立i18n包（国际化包）

```
login.btn=登录
login.password=密码
login.remember=记住我
login.tip=请登录
login.username=用户名
```

7.3 配置文件存放的位置在yaml中写入

```
## 配置文件的位置
```

```
messages:
  basename: i18n.login
```

7.4 在页面中修改国际化的代码

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://wwww.thymeleaf.org">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
      <meta name="description" content="">
      <meta name="author" content="">
      <title>Signin Template for Bootstrap</title>
      <!-- Bootstrap core CSS -->
      <link th:href="@{/css/bootstrap.min.css}" rel="stylesheet">
      <!-- Custom styles for this template -->
      <link th:href="@{/css/signin.css}" rel="stylesheet">
   </head>
   <body class="text-center">
      <form class="form-signin" action="dashboard.html">
         <img class="mb-4" th:src="@{/img/bootstrap-solid.svg}" alt="" width="72" height="72">
         <h1 class="h3 mb-3 font-weight-normal" th:text="#{login.tip}">Please sign in</h1>
         <label class="sr-only" th:text="#{login.username}">Username</label>
         <input type="text" class="form-control" th:placeholder="#{login.username}" required="" autofocus="">
         <label class="sr-only"  th:text="#{login.password}">Password</label>
         <input type="password" class="form-control" th:placeholder="#{login.password}" required="">
         <div class="checkbox mb-3">
            <label>
          <input type="checkbox" value="remember-me"  th:text="#{login.remember}">
        </label>
         </div>
         <button class="btn btn-lg btn-primary btn-block" type="submit"  th:text="#{login.btn}">Sign in</button>
         <p class="mt-5 mb-3 text-muted">© 2022-2023</p>
         <a class="btn btn-sm" th:href="@{/index.html(l='zh_CN')}">中文</a>
```

```
30          <a class="btn btn-sm" th:href="@{/index.html(l='en_US')}">English</a>
31       </form>
32    </body>
33 </html>
```

7.4 写一个国际化解析器

> 继承LocaleResolver类 并重写方法

```
1  public class MyLocaleResolver implements LocaleResolver {
2
3      @Override
4      public Locale resolveLocale(HttpServletRequest request) {
5
6          String language = request.getParameter("l");
7          //获取默认的Locale
8          //如果获取的为空
9          Locale locale = Locale.getDefault();
10         if(!StringUtils.isEmpty(language)){
11             String[] split = language.split("_");
12             System.out.println(split);
13             locale = new Locale(split[0],split[1]);
14         }
15         return locale;
16     }
17        @Override
18     public void setLocale(HttpServletRequest request, HttpServletResponse response,
   Locale locale) {
19
20     }
21 }
22
```

7.5 在MyMvcConfig中加入国际化组件，并使其交给Spring管理

```
1  /**
2   * 将自己的国际化添加到自己的组件中
3   * @return MyLocaleResolver()
```

```
 */
@Bean
public LocaleResolver localeResolver(){
    return new MyLocaleResolver();
}

```

7.6 国际化中出现404（记错）

> 当页面跳转时 inde.htmlx?l=ch_ZN ---> index.html找不到该页面，所以在myconfig.java中配置改页面跳转，因为默认找到的资源在static下，若不配置会出现404错误。

```
@Override
public void addViewControllers(ViewControllerRegistry registry) {
    registry.addViewController("/").setViewName("index");
    registry.addViewController("/index.html").setViewName("index");
}
```

八、用户登录

8.1用户提交请求（表单）

```
<form th:action="@{/user/login}"></form>
```

> 基本的登录代码

```
@RequestMapping("/user/login")
public String login(
        @RequestParam("username") String username,
        @RequestParam("password") String password,
        Model model
){
    System.out.println("进入用户登录");
    if(!StringUtils.isEmpty(username) && "123456".equals(password)){
        return "dashboard";
    }else {
        //告诉用户登录失败
        //用户名或者密码错误
        model.addAttribute("msg","用户名或者密码错误");
```

```
        return "index";
    }
}
```

8.2 页面提示是否展示

> 如果存在则展示，不存在则不展示

```
<h6 style="color: red" th:text="${msg}" th:if="${not #strings.isEmpty(msg)}"></h6>
```

8.3 重定向跳转页面

```
@RequestMapping("/user/login")
public String login(
        @RequestParam("username") String username,
        @RequestParam("password") String password,
        Model model,
        HttpSession session
){
    System.out.println("进入用户登录");
    if(!StringUtils.isEmpty(username) && "123456".equals(password)){
        session.setAttribute("LoginUser",username);
        return "redirect:/main.html";
    }else {
        //告诉用户登录失败
        //用户名或者密码错误
        model.addAttribute("msg","用户名或者密码错误");
        return "redirect:/index.html";
    }
}

```

8.4 登录成功添加session与cookie

> // HttpSession session 是在向浏览器中加入session的方法
> 如上所示

九、拦截器的编写

9.1 LoginHanderInterceptor

- 在编写拦截器时，根据前者的session或cookie判断是否登录过
- 在config中配置loginhandler拦截器继承HandlerInterceptor
- 再将拦截器加入MyMvcConfig里面，实现拦截器
- 代码如下

```
/**
 * 登录拦截器
 */
public class LoginHandlerInterceptor implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        //1.获取浏览器中的session
        Object loginUser = request.getSession().getAttribute("loginUser");
        //2.判断session是否存在
        if(loginUser == null){
            request.setAttribute("msg","没有权限请登录");
            request.getRequestDispatcher("/index.html").forward(request,response);
            return false;
        }else{
               return true;
        }

    }
}
```

```
/**
 * 添加拦截器
 * @param registry
 */
@Override
public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new LoginHandlerInterceptor())
    .addPathPatterns("/**")
    .excludePathPatterns();
}

```

十、用户的CRUD

10.1 展示员工列表

> 前端代码

```
<table class="table table-striped table-sm">
    <thead>
        <tr>
            <th>id</th>
            <th>lastName</th>
            <th>email</th>
            <th>gender</th>
            <th>department</th>
            <th>birth</th>
        </tr>
    </thead>
   <tbody>
    <tr th:each="emp:${emps}">
        <td th:text="${emp.getId()}"></td>
        <td th:text="${emp.getLastName()}"></td>
        <td th:text="${emp.getEmail()}"></td>
        <td th:text="${emp.getGender()==0?'女':'男'}"></td>
        <td th:text="${emp.getDepartment().getDepartmentName()}"></td>
        <td th:text="${#dates.format(emp.getBirth(),'yyyy-MM-dd HH:mm:ss')}"></td>
        <td >
            <button class="btn-primary">添加</button>
            <button class="btn-danger">删除</button>
        </td>
    </tr>
</tbody>
</table>
```

> 后端代码

```
@Controller
public class EmployController {
    @Autowired
    private EmployeeDao employeeDao;
    @RequestMapping("/emps")
```

```
    public String list(Model model){
        Collection<Employee> employees = employeeDao.getAll();
        model.addAttribute("emps",employees);
        return "emp/list";
    }
}
```

10.2 thmeleaf的片段切片

> 在被插入的片段中使用

```
th:fragment="插入片段的名字"
<div th:fragment="sidebar"></div>
```

> 在被插入的地方引用

```
<div th:insert="~{插入片段的页面::插入片段的名字}"></div>
<div th:insert="~{dashborad::sidebar}"></div>
```

10.3 点击高亮

> 解决方法传递参数

```
<div th:insert="~{common/common::sidebar(active='main.html')}"></div>
<a th:class="${active =='main.html' ? 'nav-link active' : 'nav-link'}"
th:href="@{/index.html}">
```

10.4 添加员工

- 按钮提交
- 页面跳转
- 添加成功
- 先查出部门的数据

```
/**
 * 跳转到添加页面
 * @return
 */
@GetMapping("/emp")
public String toAddPage(Model model){
    //1.先查询部门的所以信息
    Collection<Department> departments = departmentDao.getDepartments();
```

```
    //2.将数据加入Model
    model.addAttribute("departments",departments);
    return "emp/add";
}
```

在前端中的代码为

```
<div class="form-group">
    <label>department</label>
    <!--提交的是部门的id-->
    <select class="form-control" name="department.id">
        <option th:each="dept:${departments}" th:text="${dept.getDepartmentName()}" th:value="${dept.id}">1</option>
    </select>
</div>
```

```
/**
 * 添加用户
 * @param employee
 * @return
 */
@PostMapping("/emp")
public String AddEmp(Employee employee){

    Collection<Department> departments = departmentDao.getDepartments();
    System.out.println(employee);
    employeeDao.save(employee);
    //重定向到页面
    return "redirect:/emps";
}
```

10.5 修改用户

- 跳转页面、
- 查询出用户信息
- 最后进行修改

在前端发送请求时发送请求的id

```
<a class="btn btn-sm btn-primary" th:href="@{/emp/}+${emp.id}">编辑</a>
```

> 进行用户跳转

```
/**
 * 修改用户页面跳转
 *
 * @param
 * @return model
 *  * @PathVariable 从请求路径中获取参数 与
 *  * @RequestParam(value="param1", required=true) String param1,
 *  * @RequestParam(value="param2", required=false) String param2
 */
@GetMapping("/emp/{id}")
public String UpdatePage(@PathVariable("id") Integer id,
                         Model model) {
    // 查询该用户的信息
    Employee employee = employeeDao.get(id);
    model.addAttribute("emp",employee);
    // 查询所有公司的id与名字
    Collection<Department> departments = departmentDao.getDepartments();
    model.addAttribute("departments",departments);
    //重定向到页面
    return "emp/update";
}
```

> 修改put请求在前端

```
<!--发送put请求修改员工数据-->
<!--
1、SpringMVC中配置HiddenHttpMethodFilter;（SpringBoot自动配置好的）
2、页面创建一个post表单
3、创建一个input项，name="_method";值就是我们指定的请求方式
-->
<input type="hidden" name="_method" value="put" th:if="${emp!=null}"/>
```

10.6 用户删除

- 在前端封装Delete请求
- 在后端写出对请求的处理代码

> attr() 方法设置或返回被选元素的属性和值。

```
<button class="btn-danger deleteBtn"  th:attr="del_uri=@{/emp/}+${emp.id}">删除</button>
<form id="deleteEmpForm" method="post">
    <input type="hidden" name="_method" value="delete">
</form>
<script>
    $(".deleteBtn").click(function(){
        //删除当前员工的
        $("#deleteEmpForm").attr("action",$(this).attr("del_uri")).submit();
        return false;
    });
</script>
```

```
/**
 * 删除员工：需要提交id
 * @param id
 * @return
 */
@DeleteMapping("/emp/{id}")
public String deleteEmployee(@PathVariable("id") Integer id){
    employeeDao.delete(id);
    return "redirect:/emps";
}
```

10.7 404

> 在resources包的template包下建一个error包

- 加入404.html
- 加入500.html

十一、连接数据库

11.1 连接jdbc

- 添加jdbc的xml的maven配置
- 配置连接信息

```
spring:
  datasource:
    username: root
```

```
    password: root
    url: jdbc:mysql://localhost:3306/ssm?useUnicode=true&characterEncoding=utf-8
    driver-class-name: com.mysql.cj.jdbc.Driver

```

```

@Controller
public class JdbcController {
    @Autowired
    JdbcTemplate jdbcTemplate;

    @GetMapping("/userList")
    @ResponseBody
    public List<Map<String, Object>> userList() {
        String sql = "select * from user";
        List<Map<String, Object>> maps = jdbcTemplate.queryForList(sql);
        return maps;
    }

    @GetMapping("/userAdd")
    @ResponseBody
    public String addUserList() {
        String sql = "insert into user (id,username,birthday,sex,address) values (?,?,?,?,?)";
        String name = "test";
        List<String> addresses = new ArrayList<>();
        addresses.add("重庆");
        addresses.add("上海");
        addresses.add("天津");
        addresses.add("北京");
        addresses.add("深圳");
        addresses.add("广州");
        addresses.add("武汉");
        addresses.add("大连");
        addresses.add("长沙");
        for (int i = 0; i < 20; i++) {

            jdbcTemplate.update(sql, 123, name+4, new Date(), "女", addresses.get(i%9));
```

```
        }
        return "add_OK";
    }
}

```

11.2 Druid数据库监控

```
datasource:
  username: root
  password: root
  url: jdbc:mysql://localhost:3306/ssm?useUnicode=true&characterEncoding=utf-8
  driver-class-name: com.mysql.cj.jdbc.Driver
  ## 注入druid数据
  type: com.alibaba.druid.pool.DruidDataSource

  ## druid的相关配置
  initialSize: 5
  minIdle: 5
  maxActive: 20

```

监控后台

```
@Configuration
public class DruidConfig{

   @ConfigurationProperties(prefix="spring.datasource")
    @Bean
    public DataSource druid(){
        return new DruidDataSource();
    }

   //配置Druid监控
   @Bean
   public ServletRegistrationBean statViewServlet(){
      ServletRegistrationBean bean=new ServletRegistrationBean(new
StatViewServlet(),"/druid/*");
```

```
14
15        Map<String,String> initParams=new HashMap<>();
16        initParams.put("loginUsername","admin");
17        initParams.put("loginPassword","admin");
18        initParams.put("allow","0.0.0.0");//默认允许所有访问
19
20        bean.setInitParameters(initParams);
21        return bean;
22    }
23
24 }
```

11.3 整合Mybtis

11.3.1mybatis的配置文件

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
3          "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
4  <mapper namespace="com.test.springtest.User">
5      <select id="GetUserByID" parameterType="int"
   resultType="com.test.springtest.dao.MUser">
6          select * from `student` where id = #{id}
7      </select>
8
9      <insert
10             id="saveUser" parameterType="com.test.springtest.User"
11             useGeneratedKeys="true">
12         insert into student(NAME,AGE) values (#{name},#{age})
13     </insert>
14 </mapper>
```

11.3.2 在yaml中mybatis的配置

```
1 ## mybatis的配置
2 mybatis:
3   type-aliases-package: springboot_test_learn.demo.pojo  # 别名的包，实体类所在的包
4   mapper-locations: classpath:mapper/*.xml               # 指明xml所在的位置
```

11.3.3 xml例子

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="springboot_test_learn.demo.mapper.UserMapper">
    <select id="queryUserList" resultType="User">
        select  * from  user
    </select>

    <select id="queryUserById" resultType="User" parameterType="int">
        select * from  user where  id = #{id}
    </select>

    <insert id="addUser" parameterType="User">
        insert into user (id,name,birthday,sex,address) values (#{id},#{name},#{sex},#{birthday},#{address})

    </insert>

    <update id="updateUser"  parameterType="User">
        update user set name=#{name},birthday=#{birthday},address=#{address},birthday=#{birthday} where id = #{id}
    </update>

    <delete id="deleteUser" parameterType="int">
        delete from user where id = #{id}
    </delete>
</mapper>
```

十一、SpringSecurity01