金现代
JINXIANDAI

专题三

Spring Boot应用

——集成beetl、MyBatisPlus

目录

什么是模板引擎

- 它就是把后台的动态的数据渲染到页面上的技术
  - JSP就是一种模板引擎，在JSP中我们使用EL表达式获取后台的值，使用JSTL标签进行逻辑判断或者循环操作，将业务数据合理的显示在页面中
- 其他常见模板引擎
  - freemarker、beetl、thymeleaf

beetl简介

- 它是一款非常优秀的Java模板引擎
- 性能非常好：速度是jsp的两倍，是freemarker的6倍
- 非常容易学：语法类似于JavaScript
- 支持自定义个性化设置

基础配置项

- beetl有很多默认的配置项，当你觉得不满意时，可以自定义覆盖默认配置

```
DELIMITER_PLACEHOLDER_START=${
DELIMITER_PLACEHOLDER_END=}
DELIMITER_STATEMENT_START=<%
DELIMITER_STATEMENT_END=%>
```

- 默认占位符：${}
  - 可以把占位符理解为jsp中的el表达式,用于显示数据，语法也完全相同
- 默认界定符：<% %>
  - 把beetl的逻辑代码与html标签进行分隔,在界定符中可以进行if判断或者for循环，类似jsp中隔离HTML代码和JAVA代码的界定符

SpringBoot集成beetl

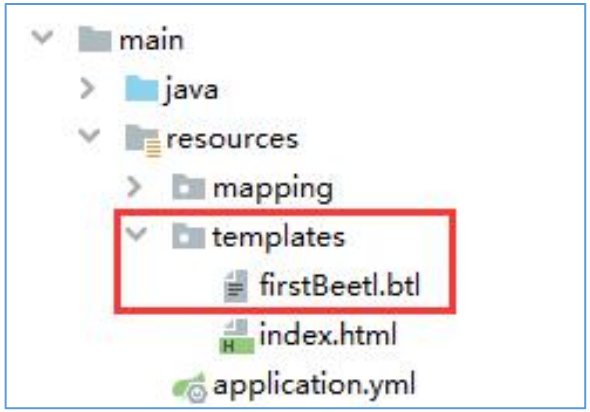
01_添加依赖

- 在pom文件中添加依赖

```
<!--集成beetl需要的依赖-->
<dependency>
        <groupId>com.ibeetl</groupId>
        <artifactId>beetl-framework-starter</artifactId>
        <version>1.1.77.RELEASE</version>
</dependency>
 <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.4</version>
</dependency>
```

02_创建模板文件

- 在模板默认位置templates下创建btl文件
  - 模板文件其实就是前台页面，替代原先的jsp文件，beetl默认模板文件就是以btl为后缀



03_编辑模板文件

- 我们在模板文件中编写静态的HTML代码即可
- 但是如果要渲染后台返回的数据时，需要使用占位符${}

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<div>
    <h1>欢迎来到beetl世界</h1>
    <h1>${msg}</h1>
</div>
</body>
</html>
```

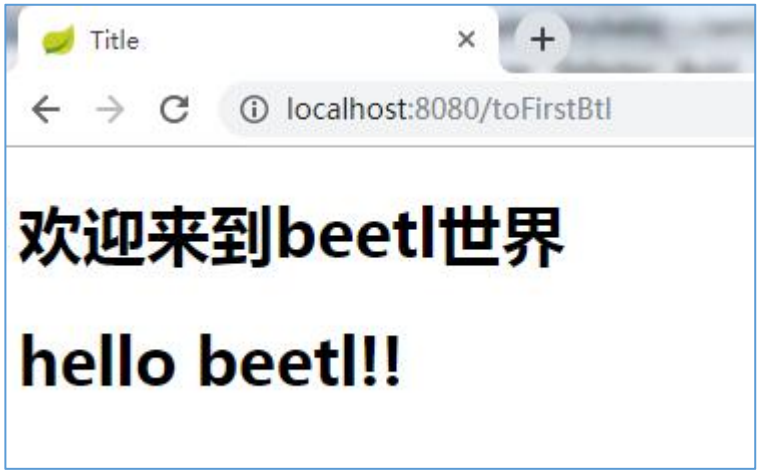
04_后台请求

- 在后台直接返回前台页面

```
//后台代码
@Controller
public class DemoController {
    @RequestMapping("/toFirstBtl")
    public String toFirst(Model m){
        m.addAttribute("msg","hello beetl!!");
        return "firstBeetl.btl";
    }
}
```

05_运行结果

- 启动主程序，访问http://localhost:8080/toFirstBtl



修改beetl默认配置

- 通常情况下，我们都是将页面文件放在WEB-INF下保护起来，templates这个位置就不是很合理了，所以我们需要修改beetl的模板位置，将其放在WEB-INF下。
- beetl默认的文件后缀btl也不是我们经常使用的方式，可以将其改为我们更加熟悉的html后缀。
- 另外如果你觉得<%%> 这样的界定符用起来麻烦的话，也可以修改为其他更加简单方便的符号。

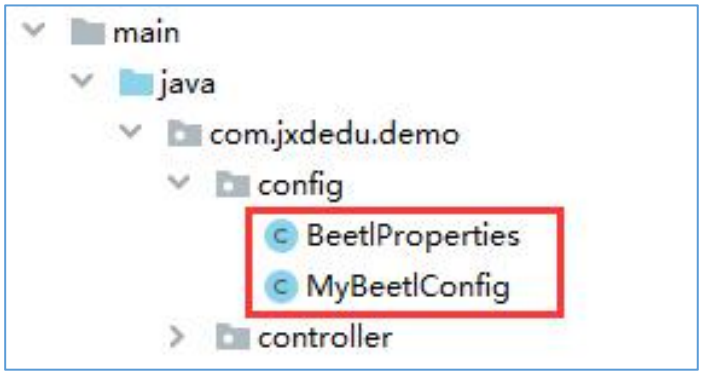
01_修改beetl配置

- 依赖添加完成之后，在application.yml中配置自定义的beetl模板文件路径以及界定符

```
spring:
   mvc:
      static-path-pattern: /static/**      # 静态资源路径
      view:
         # 以前缀方式配置自定义页面文件路径，也就是模板文件路径
         prefix: /WEB-INF/view  //该文件夹需要自定义
beetl:
 ####设置模板后缀为html,这样在后台我们直接return "*.html"就可以了
 #suffix: html    //如果我们又设置了属性类以及配置类等，可以省略改配置
 delimiter-statement-start: \@   # 开始界定符标签(yaml不允许@开头)
 delimiter-statement-end: null   #结束界定符 ：回车换行结束
```

02_读取自定义配置

- 创建beetl的属性类和配置类来读取自定义的配置项，实际做的工作是把beetl的模板加载进来，同时配置视图解析器，用于渲染前台页面。



03_后台请求

- 在后台直接返回html页面即可

```
@Controller
public class DemoController {
    @Autowired
    private  EmplMapper emplMapper;

   @RequestMapping("/getAll")
    public String welcome(Model m){
        m.addAttribute("emps",emplMapper.getAll());
        return "welcome.html";
    }
}
```

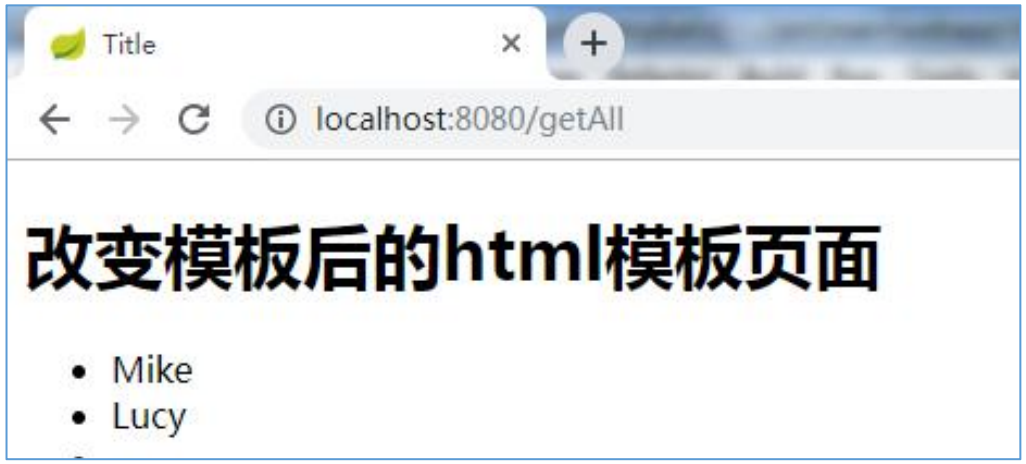
04_前台文件

- 在WEB-INF/view文件夹下创建welcome.html，遍历显示list数据

```
<body>
<div>
    <h1>改变模板后的html模板页面</h1>
    <ul>
        @for(emp in emps){
        <li>${emp.ename}</li>
        @}
    </ul>
</div>
</body>
```

05_运行结果

- 启动主程序，访问http://localhost:8080/getAll



MyBatisPlus简介

- MybatisPlus（以下简称MP）是MyBatis的增强版，对MyBatis没有改变，只是扩展，相当于MyBatis的一个插件
- MP封装了基本的增删改查操作，我们可以直接调用，无需再单独实现
- MP提供了强大的代码生成器， 可以快速生成Entity、Mapper、MapperXML、Service、Controller 等模块的代码，极大提高了开发效率
- MP封装了条件生成器，使用非常灵活

集成MyBatisPlus

01_添加依赖

- 在pom文件中添加依赖

```
<!--      mybatis的依赖模块-->
      ……
<!--      mysql数据库驱动-->
      ……
<!--      mybatis-plus的依赖-->
      <dependency>
         <groupId>com.baomidou</groupId>
         <artifactId>mybatis-plus-boot-starter</artifactId>
         <version>3.1.0</version>
      </dependency>
```

02_配置数据源

- 在application.yml中配置数据源

```
spring:
  datasource:
    url: jdbc:mysql://127.0.0.1:3306/mysql
    username: root
    password: root
    driver-class-name: com.mysql.jdbc.Driver
```

03_创建实体类

- 对应数据表创建实体类，注意，实体类名和属性名称要和数据库中的表名以及字段名一一对应

```
public class Emp {
    private Integer empno;
    private String ename;
    private String job;
    private Double sal;
    //get和set方法
   ……
}
```

03_创建实体类

- 也可以通过注解的方式将类名和表名、属性名和字段名对应起来
- 注解的value属性值即对应数据库的信息

```
@TableName(value="emp")  //表名注解
public class Emp {
   @TableId(value="empno") //主键注解
    private Integer empno;
   @TableField(value="ename") //普通字段注解
    private String ename;
    ……
}
```

04_创建Dao接口

- dao接口只需要继承BaseMapper<实体类类型>，无需再定义方法
- 穿透BaseMapper接口可以看到该接口中已经声明了基本的增删改查方法

```
public interface EmpMapper extends BaseMapper<Emp> {


}
```

```
public interface BaseMapper<T> {
    int insert(T entity);
    int deleteById(Serializable id);
    int deleteByMap(@Param("cm") Map<String, Object> columnMap);
    int delete(@Param("ew") Wrapper<T> wrapper);
     ……
}
```

查看发现

05_创建Controller

- 创建controller,调用dao层方法，此处省略了业务逻辑层

```
@Controller
public class PlusControlelr {
    @Autowired
    private EmpMapper empMapper;
    @GetMapping("/")  //需要通过get方式访问
    @ResponseBody
    public List<Emp> getAll(){
        return empMapper.selectList(null);//参数为查询条件，传递null代表查询全部数据
    }
}
```

06_主程序配置

- 添加MapperScan注解，扫描dao层接口

```
@SpringBootApplication
@MapperScan("com.jxdedu.dao")
public class PlusApplication {
    public static void main(String[] args) {
        SpringApplication.run(PlusApplication.class);
    }
}
```

07_运行结果

- 启动主程序，访问http://localhost:8080