代码案例：<https://gitee.com/zx19890628/spring-boot-example/tree/master/lab_004_web_freemarker>

# 介绍

FreeMarker是一款模板引擎： 即一种基于模板和要改变的数据，并用来生成输出文本（HTML网页、电子邮件、配置文件、源代码等）的通用工具。

它不是面向最终用户的，而是一个Java类库，是一款程序员可以嵌入他们所开发产品的组件。

FreeMarker可以用来创建模板，生成我们需要的任意文本文件，也可以用来代替jsp，用来输出页面(不产生新的文件，以流的形式输出到浏览器中)。

## 什么是FreeMarker

FreeMarker不是一个Web应用框架，而适合作为Web应用框架一个组件。

FreeMarker与容器无关，因为它并不知道HTTP或Servlet；FreeMarker同样可以应用于非Web应用程序环境。

FreeMarker更适合作为Model2框架（如Struts）的视图组件，你也可以在模板中使用JSP标记库。

FreeMarker是免费的。而且你还可以通过Eclipse的插件来编辑FreeMarker，经过验证，FreeMarker 最好的 Eclipse 编辑插件是 JBoss Tools。

## 优缺点

在java领域，表现层技术主要有三种：jsp、freemarker、velocity。

### Jsp

是大家最熟悉的技术

优点：

1、功能强大，可以写java代码

2、支持jsp标签（jsp tag）

3、支持表达式语言（el）

4、官方标准，用户群广，丰富的第三方jsp标签库

5、性能良好。jsp编译成class文件执行，有很好的性能表现

缺点：

jsp没有明显缺点，非要挑点骨头那就是，由于可以编写java代码，如使用不当容易破坏mvc结构。

### Velocity

是较早出现的用于代替jsp的模板语言

优点：

1、不能编写java代码，可以实现严格的mvc分离

2、性能良好，据说比jsp性能还要好些

3、使用表达式语言，据说jsp的表达式语言就是学velocity的

缺点：

1、不是官方标准

2、用户群体和第三方标签库没有jsp多。

3、对jsp标签支持不够好

### freemarker

优点：

1、不能编写java代码，可以实现严格的mvc分离

2、性能非常不错

3、对jsp标签支持良好

4、内置大量常用功能，使用非常方便

5、宏定义（类似jsp标签）非常方便

6、使用表达式语言

缺点：

1、不是官方标准

2、用户群体和第三方标签库没有jsp多

### 选择freemarker的原因

1、性能。velocity应该是最好的，其次是jsp，普通的页面freemarker性能最差（虽然只是几毫秒到十几毫秒的差距）。但是在复杂页面上（包含大量判断、日期金额格式化）的页面上，freemarker的性能比使用tag和el的jsp好。

2、宏定义比jsp tag方便

3、内置大量常用功能。比如html过滤，日期金额格式化等等，使用非常方便

4、支持jsp标签

5、可以实现严格的mvc分离

# 简单应用

## 使用maven，导入即可使用

<dependency>

<groupId>freemarker</groupId>

<artifactId>freemarker</artifactId>

<version>2.3.9</version>

</dependency>

## 创建一个实体类

public class Person {

private String id;

private String name;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

## 创建模板

在D盘，创建文件夹ftl，里面创建一个freemarker.html文件，内容如下

${person.id}<br />

${person.name}

## 创建启动类

启动程序即可在ftl页面创建一个hello.html，使用浏览器打开，显示的是111 哈哈哈。

public static void main(String[] args) throws Exception {

//第一步：实例化Freemarker的配置类

Configuration conf = new Configuration();

//第二步：给配置类设置路径

String dir = "D:\\ftl\\";

conf.setDirectoryForTemplateLoading(new File(dir));

Template template = conf.getTemplate("freemarker.html");

//第三步：处理模板及数据之间将数据与模板合成一个HTML

Writer out = new FileWriter(new File(dir+"hello.html"));

Map<String,Object> root = new HashMap<String,Object>();

Person p = new Person();

p.setId("111");

p.setName("哈哈哈");

root.put("person", p);

//第四步：输出html

template.process(root, out);

out.flush();

out.close();

}

Ok，简单的使用到此结束。

# 基本语法

## 数据类型

和java不同，FreeMarker不需要定义变量的类型，直接赋值即可。

字符串： value = "xxxx" 。如果有特殊字符 string = r"xxxx" 。单引号和双引号是一样的。

数值：value = 1.2。数值可以直接等于，但是不能用科学计数法。

布尔值：true or false。

List集合：list = [1,2,3] ; list=[1..100] 表示 1 到 100 的集合，反之亦然。

Map集合：map = {"key" : "value" , "key2" : "value2"}，key 必须是字符串哦！

实体类：和EL表达式差不多，直接点出来。

## 字符串操作

字符串连接：可以直接嵌套${"hello , ${name}"} ； 也可以用加号${"hello , " + name}

字符串截取：string[index]。index 可以是一个值，也可以是形如 0..2 表示下标从0开始，到下标为2结束。一共是三个数。

## 比较运算符

== （等于），!= （不等于），gt（大于），gte（大于或者等于），lt（小于），lte（小于或者等于）。不建议用 >，< 可能会报错！

一般和 if 配合使用

## 内建函数

FreeMarker 提供了一些内建函数来转换输出，其结构：变量?内建函数，这样就可以通过内建函数来转换输出变量。

html： 对字符串进行HTML编码；

cap\_first： 使字符串第一个字母大写；

lower\_case： 将字符串转成小写；

upper\_case： 将字符串转成大写；

size： 获得集合中元素的个数；

int： 取得数字的整数部分。

## 变量空判断

! 　　指定缺失变量的默认值；一般配置变量输出使用

?? 　判断变量是否存在。一般配合if使用 <#if value??></#if>

## 宏指令

可以理解为java的封装方法，供其他地方使用。宏指令也称为自定义指令，macro指令

语法很简单：<#macro val > 声明macro </#macro>; 使用macro <@val />

## 命名空间

可以理解为java的import语句，为避免变量重复。一个重要的规则就是:路径不应该包含大写字母，使用下划线\_分隔词语，myName --> my\_name

语法很简单：<#import "xxx.ftl" as val>

其他没有说明的语法是因为和java一样，没什么特别之处。所以没有列出来。

# 宏指令（标签用法）

## Assign

1、<#assign name1="北京" name2="上海" name3="广东">

调用：${name1}、${name2}、${name3}

2、<#assign name1> 消息模拟.. </#assign>

调用：${name1}

3、<#assign name5 in my命名空间> 消息模拟.. </#assign>

调用：<#import "InstructionReference.ftl" as my命名空间>

${my.name5}

## 遍历集合

<!-- 遍历list -->

<#list persons as person> ${person} </#list>

<!--获取当前索引 -->

<#list persons as p> ${p\_index} </#list>

<!-- 调用map -->

${mx.alingluo}/${mx.lling}

<!-- 遍历map -->

<#list mx?keys as k> ${mx[k]} </#list>

## 格式化时间

如果cur\_time 是 java.util.Date 类型的对象。

1:${cur\_time?date} 输出yyyy-MM-dd

2:${cur\_time?datetime} 输出yyyy-MM-dd HH:mm:ss

3:${cur\_time?time} 输出 HH:mm:ss

## attempt, recover

<#attempt>

=============>输出：${my.name5}

<#recover>

错误recover block

</#attempt>

attempt block：任意内容的模板块。这是会被执行的， 但是如果期间发生了错误，那么这块内容的输出将会回滚，之后 recover block 就会被执行。

recover block: 任意内容的模板块。 这个仅在 attempt block 执行期间发生错误时被执行。你可以在这里打印错误信息或其他操。

## compress

当你使用了对空白、换行的格式(比如HTML或XML) 时压缩指令对于移除多余的空白

<#assign x = " moo \n\n ">

<#compress>

1 2 3 4 5

${x}

test only

I said, test only

</#compress>

## flush

<#Flush>

说明：当 FreeMarker 生成输出时，它通常不会立即发送到最终接收端

(比如web浏览器或最终的文件)，而是会将内容累积在缓冲区，

发送一个大块的内容。 缓冲区的精确规则不是由 FreeMarker 决定的

，而是由嵌入的软件决定的。 将缓冲区中累积的内容发送出去称为冲洗。

## function, return

<#function name param1 param2 ... paramN>

...

<#return returnValue>

...

</#function>

说明：

name：方法变量的名称(不是表达式)

param1, param2 等： 局部变量的名称， 存储参数的值(不是表达式)，在 = 号后面和默认值 (是表达式)是可选的。

paramN，最后一个参数， 可以可选的包含一个尾部省略(...)， 这就意味着宏接受可变的参数数量。局部变量 paramN 将是额外参数的序列。

returnValue： 计算方法调用值的表达式。

return 指令可以在 <#function ...> 和 </#function> 之间被用在任意位置和任意次数。

没有默认值的参数必须在有默认值参数 (paramName=defaultValue) 之前

示例1：

<#function avg x y>

<#return (x + y) \* 6>

</#function>

${avg (10, 20)}

示例2：

<#function avg nums...>

<#local sum = 0>

<#list nums as num>

<#local sum = sum + num>

</#list>

<#if nums? size = 5>

<#return sum / nums? size>

</#if>

</#function>

${avg (10, 20, 30, 40, 40)}//参数可任意个

## global全局变量

1、<#global name1="北京2">

调用：${name1}

2、<#global name1="北京" name2="上海" name3="广东">

调用：${name1}、${name2}、${name3}

3、<#global name4>消息模拟. </#global>

调用：${name4}

说明：一个相同名称的变量存在的话， 那么会隐藏由 global 指令创建的变量。

## if, else, elseif

示例1：只有 if 没有 elseif 和 else

<#if x == 1>

x is 1

</#if>

示例2：只有 if 没有 elseif 但是有 else

<#if x == 1>

x is 1

<#else>

x is not 1

</#if>

示例3：有 if 和两个 elseif 但是没有 else

<#if x == 1>

x is 1

<#elseif x == 2>

x is 2

<#elseif x == 3>

x is 3

</#if>

示例4：有 if 和三个 elseif 还有 else

<#if x == 1>

x is 1

<#elseif x == 2>

x is 2

<#elseif x == 3>

x is 3

<#elseif x == 4>

x is 4

<#else>

x is not 1 nor 2 nor 3 nor 4

</#if>

示例5：嵌套 if 指令

<#if x == 1>

x is 1

<#if y == 1>

and y is 1 too

<#else>

but y is not

</#if>

<#else>

x is not 1

<#if y < 0>

and y is less than 0

</#if>

</#if>

## import

<#import path as hash>

说明：使用参考assign，创建一个命名空间引入然后被调用。

path模板的路径。 这是一个算作是字符串的表达式。(换句话说，它不是一个固定的字符串， 它可以是这样的一些东西，比如，profile.baseDir + "/menu.ftl"。)

hash访问命名空间的哈希表变量不带引号的名字。不是表达式。 (要引入动态创建的名字)

示例：<#import "/libs/mylib.ftl" as my>

<@my.copyright date="1999-2002"/>

## include

<#include path> 或 <#include path options>

path包含文件的路径；一个算作是字符串的表达式。(用其他话说， 它不用是一个固定的字符串，它也可以是像"/menu.ftl"这样的东西。)

options一个或多个这样的选项： encoding=encoding, parse=parse

parse： 算作是布尔值的表达式(为了向下兼容，也接受一部分字符串值)

encoding： 算作是字符串的表达式

ignore\_missing: 算作是布尔值的表达式

parse：如果它为 true， 那么被包含的文件将会当作FTL来解析，否则整个文件将被视为简单文本 (也就是说不会在其中查找 FreeMarker 的结构)。如果你忽略了这个选项， 那么它默认是 true。

encoding：被包含文件从包含它的文件继承的编码方式 (实际就是字符集)，除非你用这个选项来指定编码方式。 合法的名字有：ISO-8859-2，UTF-8，Shift\_JIS，Big5，EUC-KR，GB2312。

ignore\_missing：当为 true，模板引用为空时压制错误，而 <#include ...> 不会输出任何东西。当为 false 时，如果模板不存在， 那么模板处理就会发生错误并停止。如果忽略这个选项，那么它的默认值是 false。

说明：使用它在你的模板中插入另外一个 FreeMarker 模板文件 (由 path 参数指定)。 被包含模板的输出格式是在 include 标签出现的位置插入的。 被包含的文件和包含它的模板共享变量，就像是被复制粘贴进去的一样。

示例：

<#include "/common/copyright.ftl">

<#include "/common/navbar.html" parse=false encoding="Shift\_JIS">

## local

<#local name=value>

或

<#local name1=value1 name2=value2 ... nameN=valueN>

或

<#local name>

capture this

</#local>

说明：name在root中局部对象的名称。它不是一个表达式。但它可以被写作是字符串形式， 如果变量名包含保留字符，这是很有用的，比如 <#local "foo-bar" = 1>。 请注意，这个字符串没有扩展插值(如"${foo}")。

=：赋值操作符，也可以简写的赋值操作符之一 (++，+= 等...)，和 assign 指令 相似。value 存储的值，是表达式。

示例：它和 assign 指令 类似，但是它创建或替换局部变量。 这仅仅在宏和方法的内部定义才会有作用。

<#function Str1 nums>

<#local sum = ".......">

<#return sum + nums>

</#function>

调用：${Str1("测试")} 输出：.......测试

## macro, nested, return

<#macro name param1 param2 ... paramN>

...

<#nested loopvar1, loopvar2, ..., loopvarN>

...

<#return>

...

</#macro>

说明：

• name： 宏变量的名称，它不是表达式。和 顶层变量 的语法相同，比如 myMacro 或 my\-macro。 然而，它可以被写成字符串的形式，如果宏名称中包含保留字符时，这是很有用的，

　　　　比如 <#macro "foo~bar">...。 注意这个字符串没有扩展插值(如"${foo}")。

• param1， param2，等...： 局部变量 的名称，存储参数的值 (不是表达式)，在 = 号后面和默认值(是表达式)是可选的。 默认值也可以是另外一个参数，

　　　　比如 <#macro section title label=title>。参数名称和 顶层变量 的语法相同，所以有相同的特性和限制。

• paramN， 最后一个参数，可能会有三个点(...)， 这就意味着宏接受可变数量的参数，不匹配其它参数的参数可以作为最后一个参数 (也被称作笼统参数)。

　　　　当宏被命名参数调用， paramN 将会是包含宏的所有未声明的键/值对的哈希表。当宏被位置参数调用，

paramN将是额外参数的序列。 (在宏内部，要查找参数，可以使用 myCatchAllParam?is\_sequence。)

• loopvar1， loopvar2等...： 可选的，循环变量 的值， 是 nested 指令想为嵌套内容创建的。这些都是表达式。

return 和 nested 指令是可选的，而且可以在 <#macro…> 和 </#macro> 之间被用在任意位置和任意次数

示例1：标准写法

<#macro test1>

Test text

</#macro>

调用：<@mtest1/>

示例2：带参数的宏(参数可多个)

<#macro test2 data>

Test text：${data}

</#macro>

调用：<@test2 data="Hello"/>

示例3：带参数、默认参数(可多个)

<#macro test3 data1 data2="2" data3="3">

===>${data1}

===>${data2}

===>${data3}

</#macro>

调用：<@test3 data1="Hello" data2="Hello"/> <@test3 data1="Hello" />

示例4：集合迭代元素的宏

<#macro list title items>

<p>${title? cap\_first}: </p>

<ul>

<#list items as x>

<li>${x? cap\_first} </li>

</#list>

</ul>

</#macro>

调用：<@list items=["mouse", "elephant", "python"] title="标题"/>

Nested

Nested

说明：ested 指令执行自定义指令开始和结束标签中间的模板片段。 嵌套的片段可以包含模板中任意合法的内容

示例1：标准简单

<#macro do\_twice>

1. <#nested>

2. <#nested>

</#macro>

调用：<@do\_twice>something</@do\_twice>

输出：

示例2：嵌套内容创建循环变量

<#macro do\_thrice>

<#nested 1>

<#nested 2>

<#nested 3>

</#macro>

调用：<@do\_thrice; x>

${x} Anything.

</@do\_thrice>

return

说明：可以在任意位置留下一个宏或函数定义。

<#macro test d>

This test

<#return>

Will not be printed${d}

</#macro>

调用：<@test d="===>"/>

## noparse(不解析ftl)

<#noparse>

...

</#noparse>

说明：不会在这个指令体中间寻找FTL标签， 插值和其他特殊的字符序列，除了noparse的结束标记，避开对noparse内部ftl的解析。

示例1：

<#noparse>

<#list animals as animal>

<tr><td>${animal.name} <td>${animal. price} Euros

</#list>

</#noparse>

## stop

<#stop>

或

<#stop reason>

说明：reason停止对模版解析，并且抛出一个异常，终止原因的信息化消息。是表达式，被算做是字符串，中止模板处理，给出(可选的)错误消息。 不要在普通情况下对结束模板处理使用！ FreeMarker 模板的调用者会将它视为失败的模板呈现。

## switch, case, default, break

<#switch "2">

<#case "1">

===>；1

<#break>

<#case "2">

===>； 2

<#break>

<#case "3">

===>； 3

<#break>

</#switch>

说明：value， refValue1表达式将会计算成相同类型的标量，break 和 default 是可选的。

## t, lt, rt

<#t>

<#lt>

<#rt>

说明：指示FreeMarker去忽略标记中行的特定的空白的移除发生在解析阶段

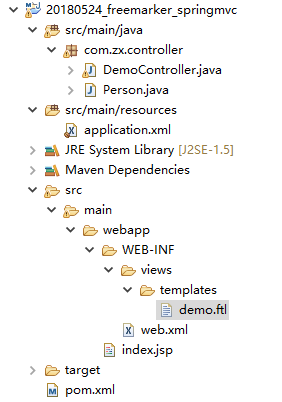
t (整体削减)：忽略本行中首和尾的所有空白。

lt (左侧削减)：忽略本行中首部所有的空白。

rt (右侧削减)：忽略本行中尾部所有的空白。

# 整合spring mvc

使用maven项目构建项目。项目案例是20180524\_freemarker\_springmvc。

项目结构如图。

## 导入依赖

如上，2.1 并且加入spring mvc相关的所有包

## 配置web.xml

就是配置spring 和 spring mvc

<!-- Spring配置文件 -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:application.xml</param-value>

</context-param>

<!-- Spring监听器 -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- 添加对springmvc的支持 -->

<servlet>

<servlet-name>springMVC</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:application.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springMVC</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

## 配置application.xml

配置freemarker作为视图层，并配置前后缀、文档编码

<context:annotation-config /><!-- 启动注解 -->

<mvc:annotation-driven /><!-- 启动mvc注解 -->

<context:component-scan base-package="com.zx"/><!-- 扫描包 -->

<bean id="freemarkerConfig" class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer">

<property name="templateLoaderPath" value="/WEB-INF/views/templates"/>

</bean>

<bean id="viewResolver" class="org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver">

<property name="prefix" value=""/>

<property name="suffix" value=".ftl"/>

<property name="contentType" value="text/html; charset=UTF-8"/>

</bean>

## 创建一个controller

用于返回数据到freemarker模板

@Controller

public class DemoController {

@RequestMapping("/get")

public String get(Model model) {

Person p1 = new Person("1", "张三");

Person p2 = new Person("2", "李四");

Person p3 = new Person("3", "王五");

List<Person> restList = Arrays.asList(p1, p2, p3);

Map<String, Person> restMap = new HashMap<String, Person>();

restMap.put("1", p1);

restMap.put("2", p2);

restMap.put("3", p3);

model.addAttribute("restList", restList);

model.addAttribute("restMap", restMap);

model.addAttribute("cur\_time", new Date());

return "demo";

}

}

## 创建模板

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>freemarker demo</title>

</head>

<body>

<h1>遍历集合 </h1>

<table>

<#list restList as person >

<tr>

<td>${person.id}</td>

<td>${person.name}</td>

</tr>

</#list>

</table>

<h1>遍历map</h1>

<table>

<#list restMap?keys as key >

<tr>

<td>${restMap[key].id}</td>

<td>${restMap[key].name}</td>

</tr>

</#list>

</table>

<h1>格式化时间</h1>

1:${cur\_time?date} <br/>

2:${cur\_time?datetime} <br/>

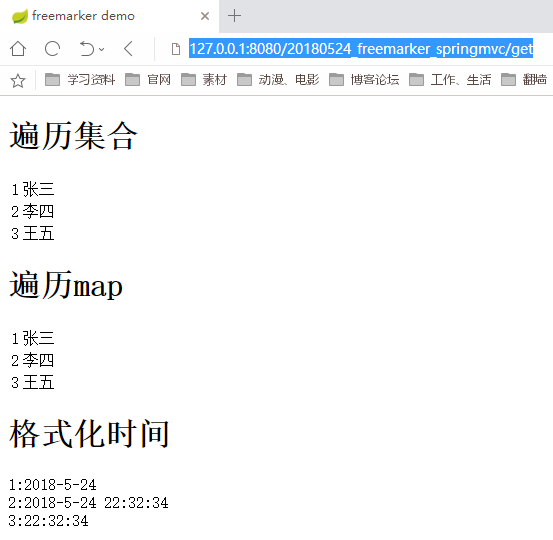
3:${cur\_time?time} <br/>

</body>

</html>

## 启动项目

部署项目后启动，访问: http://127.0.0.1:8080/20180524\_freemarker\_springmvc/get。页面如下。



# 整合spring boot

实际中，spring boot 原始支持freemarker，简单配置后即可使用

## 添加依赖

<dependency>

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

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

</dependency>

## 配置application.perproties

以下是freemarker全部的配置。以下为全部的默认配置。详细内容在配置类FreeMarkerProperties中。

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/

## 创建controller

@Controller

public class DemoController {

@RequestMapping("sysUser")

public String user(Model m) {

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

SysUser u1 = new SysUser(0001, "hello1", "11111111111111111");

SysUser u2 = new SysUser(0002, "hello2", "22222222222222222");

SysUser u3 = new SysUser(0003, "hello3", "33333333333333333");

list.add(u1);

list.add(u2);

list.add(u3);

m.addAttribute("userList", list);

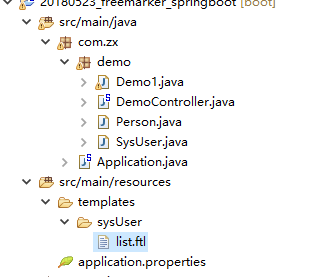
m.addAttribute("sysUser", "SysUser");

return "sysUser/list";

}

}

## 创建模板



<!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>

<meta content="text/html;charset=utf-8"></meta>

<title>Hello World!</title>

</head>

<body>

<div class="container">

<table class="table">

<caption>${sysUser}</caption>

<thead>

<tr>

<th>First Name</th>

<th>Last Name</th>

<th>User Name</th>

</tr>

</thead>

<tbody>

<#list userList as user>

<tr>

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

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

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

</tr>

</#list>

</tbody>

</table>

</div>

</body>

</html>

## 启动项目

访问URL： http://127.0.0.1:8080/sysUser/ 出现以下页面

