# 处理模型数据

Spring MVC 提供了以下几种途径输出模型数据：

ModelAndView，ModelMap，Map 及 Model: 把模型数据放入请求域。

@SessionAttributes: 将模型中的某个属性暂存到HttpSession 中，以便多个请求之间可以共享这个属性

@ModelAttribute: 方法入参标注该注解后, 入参的对象就会放到数据模型中

# 共享数据

ModelAndView（可返回页面）和Model/ ModelMap/ Map

## （数据）Model/ ModelMap/ Map

### 概述

其实3者基本没区别

Model：使用Model来向页面传递数据，Model是一个接口，在参数里直接声明model即可。

ModelMap：ModelMap是Model接口的实现类，也可以通过ModelMap向页面传递数据；使用Model和ModelMap的效果一样，如果直接使用Model，springmvc会实例化ModelMap。

Map：

### model

代码实现：

/\*\*

\* 根据id查询商品,使用Model

\*

\* **@param** request

\* **@param** model

\* **@return**

\*/

@RequestMapping("/itemEdit")

**public** String queryItemById(HttpServletRequest request, Model model) {

// 从request中获取请求参数

String strId = request.getParameter("id");

Integer id = Integer.*valueOf*(strId);

// 根据id查询商品数据

Item item = **this**.itemService.queryItemById(id);

// 把结果传递给页面

// ModelAndView modelAndView = new ModelAndView();

// 把商品数据放在模型中

// modelAndView.addObject("item", item);

// 设置逻辑视图

// modelAndView.setViewName("itemEdit");

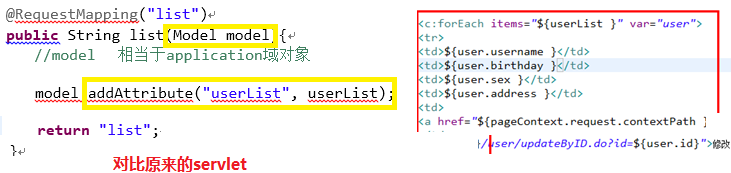
// 把商品数据放在模型中

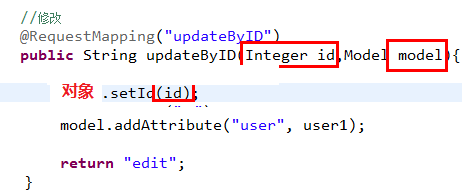
model.addAttribute("item", item);

**return** "itemEdit";

}

Spring mvc使用model对象，model对象相当于application，然后简而言之和用原来el获取没区别





### ModelMap

### map

## （数据视图）ModelAndView

  控制器处理方法的返回值如果为ModelAndView, 则其既包含视图信息，也包含模型数据信息。SpringMVC 会把 ModelAndView 的 model 中数据放入到 request 域对象中.

ModelAndView mv = newModelAndView(viewName);

    添加模型数据 :

MoelAndView addObject(String attributeName, Object attributeValue)

ModelAndView addAllObject(Map<String,?>  modelMap)

    设置视图 :

void setView(View view)

void setViewName(String viewName)

/\*\*

\* 目标方法的返回值可以是 ModelAndView 类型。

\* 其中可以包含视图和模型信息

\*SpringMVC 会把 ModelAndView 的 model 中数据放入到 request 域对象中.

\* @return

\*/

@RequestMapping("/testModelAndView")

public ModelAndView testModelAndView(){

String viewName = SUCCESS;

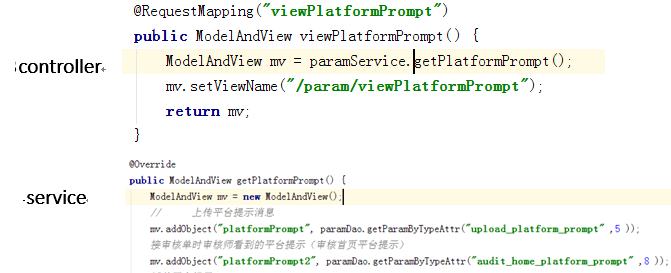
ModelAndView modelAndView = new ModelAndView(viewName);

//添加模型数据到 ModelAndView 中.

modelAndView.addObject("time", new Date());

return modelAndView;

}



## 自定义vo

BaseResult：

pageInfo：

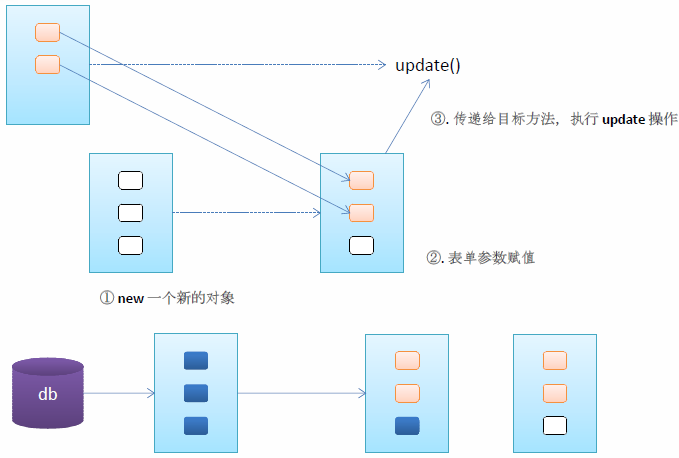
## 说白了，根据需求来

不一定要返回特殊vo，只是一种约束习惯

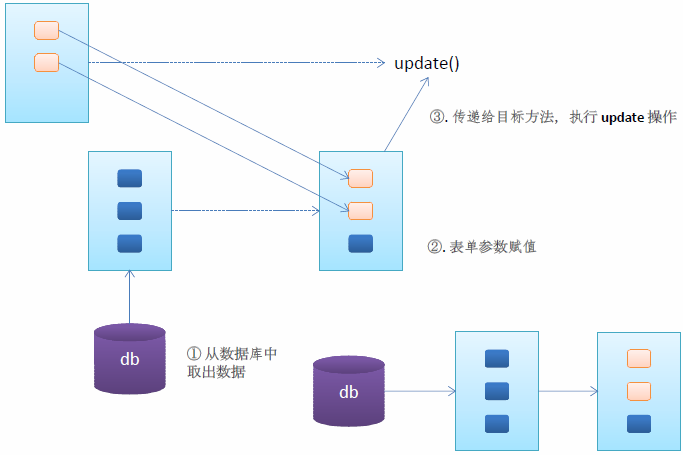
# 所有请求域@ModelAttribute

使用@ModelAttribute将公用的取数据的方法返回值传到页面，不用在每一个controller方法通过Model将数据传到页面。

传统会更新不必要的字段或更新后的字段为空



例如，更新操作，有一个字段不能被修改类似，过程此图：



## 运行流程

在方法定义上使用 @ModelAttribute 注解：Spring MVC在调用目标处理方法前，会先逐个调用在方法级上标注了@ModelAttribute 的方法。

在方法的入参前使用 @ModelAttribute 注解：可以从隐含对象中获取隐含的模型数据中获取对象，再将请求参数绑定到对象中，再传入入参；将方法入参对象添加到模型中

/\*\*

\* 运行流程:

\* 1. 执行 **@ModelAttribute** 注解修饰的方法: 从数据库中取出对象, 把对象放入到了 Map 中. 键为: user

\* 2. SpringMVC 从 Map 中取出 User 对象, 并把表单的请求参数赋给该 User 对象的对应属性.

\* 3. SpringMVC 把上述对象传入目标方法的参数.

\* 注意: 在 @ModelAttribute 修饰的方法中, 放入到 Map 时的键需要和目标方法入参类型的第一个字母小写的字符串一致!

\*\*/

@RequestMapping("/testModelAttribute")

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

System.*out*.println("修改: " + user);

**return** *SUCCESS*;

}

@ModelAttribute

**public** **void** getUser(@RequestParam(value="id",required=**false**) Integer id,

Map<String, Object> map){

System.*out*.println("modelAttribute method");

**if**(id != **null**){

//模拟从数据库中获取对象

User user = **new** User(1, "Tom", "123456", "tom@qq.com", 12);

System.*out*.println("从数据库中获取一个对象: " + user);

map.put("user", user);

}

}

<!--

模拟修改操作

1. 原始数据为: 1, Tom, 123456,tom@qq.com,12

2. 密码不能被修改.

3. 表单回显, 模拟操作直接在表单填写对应的属性值

-->

<form action=*"springmvc/testModelAttribute"* method=*"Post"*>

<input type=*"hidden"* name=*"id"* value=*"1"*/>

username: <input type=*"text"* name=*"username"* value=*"Tom"*/>

<br>

email: <input type=*"text"* name=*"email"* value=*"tom@qq.com"*/>

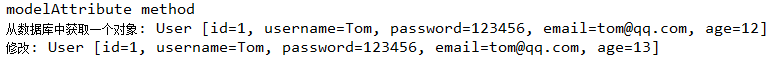
<br>

age: <input type=*"text"* name=*"age"* value=*"12"*/>

<br>

<input type=*"submit"* value=*"Submit"*/>

</form>



注意：

map.put("abc", user);

(@ModelAttribute("abc") User user)

## 源代码分析的流程

1. 调用 @ModelAttribute 注解修饰的方法. 实际上把 @ModelAttribute 方法中 Map 中的数据放在了 implicitModel 中.

2. 解析请求处理器的目标参数, 实际上该目标参数来自于 WebDataBinder 对象的 target 属性

1). 创建 WebDataBinder 对象:

①. 确定 objectName 属性: 若传入的 attrName 属性值为 "", 则 objectName 为类名第一个字母小写.

注意: attrName. 若目标方法的 POJO 属性使用了 @ModelAttribute 来修饰, 则 attrName 值即为 @ModelAttribute 的 value 属性值

②. 确定 target 属性:

在 implicitModel 中查找 attrName 对应的属性值. 若存在, ok

若不存在: 则验证当前 Handler 是否使用了 @SessionAttributes 进行修饰, 若使用了, 则尝试从 Session 中

获取 attrName 所对应的属性值. 若 session 中没有对应的属性值, 则抛出了异常.

若 Handler 没有使用 @SessionAttributes 进行修饰, 或 @SessionAttributes 中没有使用 value 值指定的 key

和 attrName 相匹配, 则通过反射创建了 POJO 对象

2). SpringMVC 把表单的请求参数赋给了 WebDataBinder 的 target 对应的属性.

3). SpringMVC 会把 WebDataBinder 的 attrName 和 target 给到 implicitModel. 近而传到 request 域对象中.

4). 把 WebDataBinder 的 target 作为参数传递给目标方法的入参.

## SpringMVC 确定目标方法 POJO 类型入参的过程

1. 确定一个 key:

若目标方法的 POJO 类型的参数是否使用 @ModelAttribute 作为修饰,：

没有，则 key 为 POJO 类名第一个字母的小写；

有, 则为 @ModelAttribute 注解的 value 属性值.

对象

2. 在 implicitModel 中查找 key 对应的对象,

若存在, 则作为入参传入；

若在 @ModelAttribute 标记的方法中在 Map 中保存过, 且 key 和 1 确定的 key 一致, 则会获取到。

3. 若 implicitModel 中不存在, 则检查当前的 Handler

是否使用 @SessionAttributes 注解修饰,

若使用了该注解, 且 @SessionAttributes 注解的 value 属性值中包含了 key, 则会从 HttpSession 中来获取 key 所对应的 value 值, 若存在则直接传入到目标方法的入参中. 若不存在则将抛出异常.

4. 若 Handler 没有，则会通过反射

来创建 POJO 类型的参数, 传入为目标方法的参数

5. SpringMVC 会把 key 和 POJO 类型的对象保存到 implicitModel 中, 进而会保存到 request 中.

结论：

1. 有 @ModelAttribute 标记的方法, 会在每个目标方法执行之前被 SpringMVC 调用!

2. @ModelAttribute 注解也可以来修饰目标方法 POJO 类型的入参, 其 value 属性值有如下的作用:

SpringMVC 会使用 value 属性值在 implicitModel 中查找对应的对象, 若存在则会直接传入到目标方法的入参中.

SpringMVC 会一 value 为 key, POJO 类型的对象为 value, 存入到 request 中.

# Session域@SessionAttributes

## 运行流程

若希望在多个请求之间共用某个模型属性数据，则可以在控制器类上标注一个 @SessionAttributes, Spring MVC 将在模型中对应的属性暂存到 HttpSession 中。

@SessionAttributes 除了可以通过属性名指定需要放到会话中的属性外，还可以通过模型属性的对象类型指定哪些模型属性需要放到会话中

类前：

@SessionAttributes(value={"user"}, types={String.**class**})

/\*\*

\* **@SessionAttributes** 除了可以通过属性名指定需要放到会话中的属性外(实际上使用的是 value 属性值),

\* 还可以通过模型属性的对象类型指定哪些模型属性需要放到会话中(实际上使用的是 types 属性值)

\*

\* 注意: 该注解只能放在类的上面. 而不能修饰放方法.

\*/

@RequestMapping("/testSessionAttributes")

**public** String testSessionAttributes(Map<String, Object> map){

User user = **new** User("Tom", "123456", "tom@qq.com", 15);

map.put("user", user);

map.put("school", "laolan");

**return** *SUCCESS*;

}

<a href=*"springmvc/testSessionAttributes"*>Test SessionAttributes</a>

SessionAttributes<br>

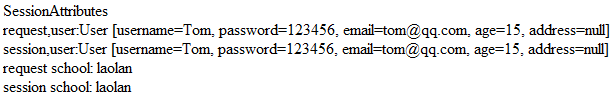
request,user:${requestScope.user }<br>

session,user:${sessionScope.user }<br>

request school: ${requestScope.school }<br>

session school: ${sessionScope.school }<br>

结果：



## 由@SessionAttributes引发的异常

如果在处理类定义处标注了@SessionAttributes(“xxx”)，则尝试从会话中获取该属性，并将其赋给该入参，然后再用请求消息填充该入参对象。如果在会话中找不到对应的属性，则抛出 HttpSessionRequiredException 异常

解决方法：

