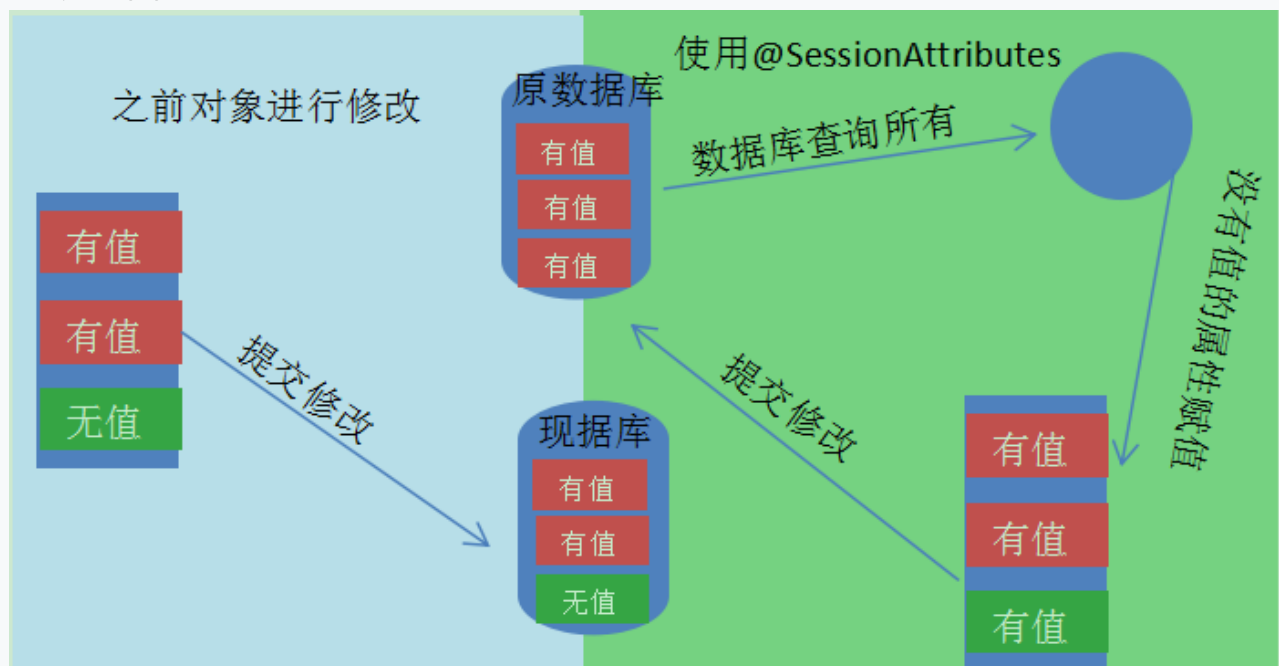


05 SpringMVC-@ModelAttribute

SpringMVC

一:@ModelAttribute描述

- 1、之前传入对象修改数据的时候，当传入的对象其中的某个数据没有值的时候，修改了数据库中某条数据该对应的值就会是null值
- 2、那么出现这种问题怎么处理？
- 3、@ModelAttribute就是用来处理这个中问题的，原理先到数据库中查询出来数据，如传入对象修改的数据其中没有值，则对该进行把查询出来的值对应赋值
- 4、原理图



5、Controller层类

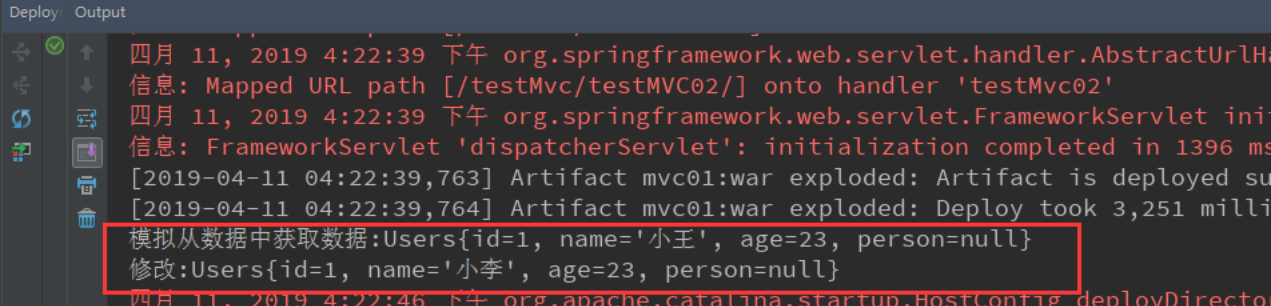
```
1. @SessionAttributes(value = {"sessionKey", "sessionKey1"}, types = {String.class, Date.class})
2. @Controller
3. public class TestMvc01 {
```

```

4.     @ModelAttribute
5.     public void testAttribute(@RequestParam(value = "id" ,required = false) Integer id,
6.                               Map<String, Object> map){
7.         if(id!=null){
8.             //模拟从数据中获取数据
9.             Users users = new Users(1,"小王",23);
10.            System.out.println("模拟从数据中获取数据:"+users);
11.            map.put("users",users);
12.        }
13.    }
14.
15.    //目标传入的方法的第一个字母的字符串必须要和@ModelAttribute修饰方法中，放入map的key一致
16.    @RequestMapping("/testModelAttribute")
17.    public String testModelAttribute(Users users){
18.        System.out.println("修改:"+users);
19.        return "success";
20.    }
21. }
22. =====页面代码=====
23. <FORM ACTION="testModelAttribute" METHOD="post">
24.     <INPUT TYPE="text" VALUE="1" NAME="id"><br/>
25.     <INPUT TYPE="text" VALUE="小李" NAME="name"><br/>
26.     <input type="submit" value="submit">
27. </FORM>

```

6、执行结果



```

四月 11, 2019 4:22:39 下午 org.springframework.web.servlet.handler.AbstractUrlH
信息: Mapped URL path [/testMvc/testMVC02/] onto handler 'testMvc02'
四月 11, 2019 4:22:39 下午 org.springframework.web.servlet.FrameworkServlet ini
信息: FrameworkServlet 'dispatcherServlet': initialization completed in 1396 ms
[2019-04-11 04:22:39,763] Artifact mvc01:war exploded: Artifact is deployed su
[2019-04-11 04:22:39,764] Artifact mvc01:war exploded: Deploy took 3,251 milli
模拟从数据中获取数据:Users{id=1, name='小王', age=23, person=null}
修改:Users{id=1, name='小李', age=23, person=null}
四月 11, 2019 4:22:46 下午 org.apache.catalina.startup.HostConfig deployDirecto

```

二:源码讲解

1、在controller层的 @ModelAttribute注解方法的model传入对象出打上断点

```

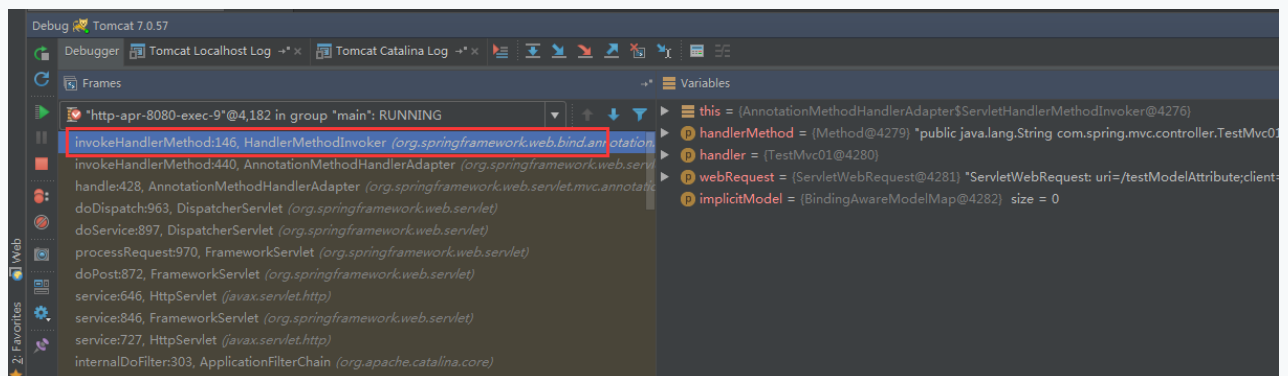
1.     @ModelAttribute
2.     public void testAttribute(@RequestParam(value = "id" ,required = false) Integer id,
3.                               Map<String, Object> map){
4.         if(id!=null){
5.             //模拟从数据中获取数据
6.             Users users = new Users(1,"小王",23);
7.             System.out.println("模拟从数据中获取数据:"+users);
8.             /**
9.              进行DUG打上断点
10.             */
11.             map.put("users",users);
12.         }
13.     }
14.
15.     //访问的地址打上断点
16.     @RequestMapping("/testModelAttribute")
17.     public String testModelAttribute(Users users){
18.         System.out.println("修改:"+users);
19.         /**
20.          进行DUG打上断点
21.         */
22.         return "success";
23.     }

```

2、运行DUG模式

1、点击invokeHandlerMethod方法

1、点击invokeHandlerMethod方法在HandlerMethodInvoker类中



2、在到该方法中打上对应的断点

```

1.     public final Object invokeHandlerMethod(Method handlerMethod, Object
t handler, NativeWebRequest webRequest, ExtendedModelMap implicitModel)
throws Exception {
2.         /**
3.         进行DUG打上断点
4.         */
5.         Method handlerMethodToInvoke = BridgeMethodResolver.findBridged
Method(handlerMethod);
6.
7.         //中间代码省略.....
8.         while(true) {
9.             Object[] args;
10.            String attrName;
11.            Method attributeMethodToInvoke;
12.            do {
13.                if (!var7.hasNext()) {
14.                    /**
15.                    进行DUG打上断点
16.                    */
17.                    Object[] args =
this.resolveHandlerArguments(handlerMethodToInvoke, handler,
webRequest, implicitModel);
18.                    if (debug) {
19.                        logger.debug("Invoking request handl
er method: " + handlerMethodToInvoke);
20.                    }
21.
ReflectionUtils.makeAccessible(handlerMethodToInvoke);
22.                    /**
23.                    进行DUG打上断点
24.                    */
25.                    return handlerMethodToInvoke.invoke(handler,
args);
26.                }
27.                //中间代码省略.....
28.
ReflectionUtils.makeAccessible(attributeMethodToInvoke
);
29.
30.                /**
31.                进行DUG打上断点
32.                */
33.                Object attrValue = attributeMethodToInvoke.invoke(handl
er, args);
34.                /**
35.                进行DUG打上断点

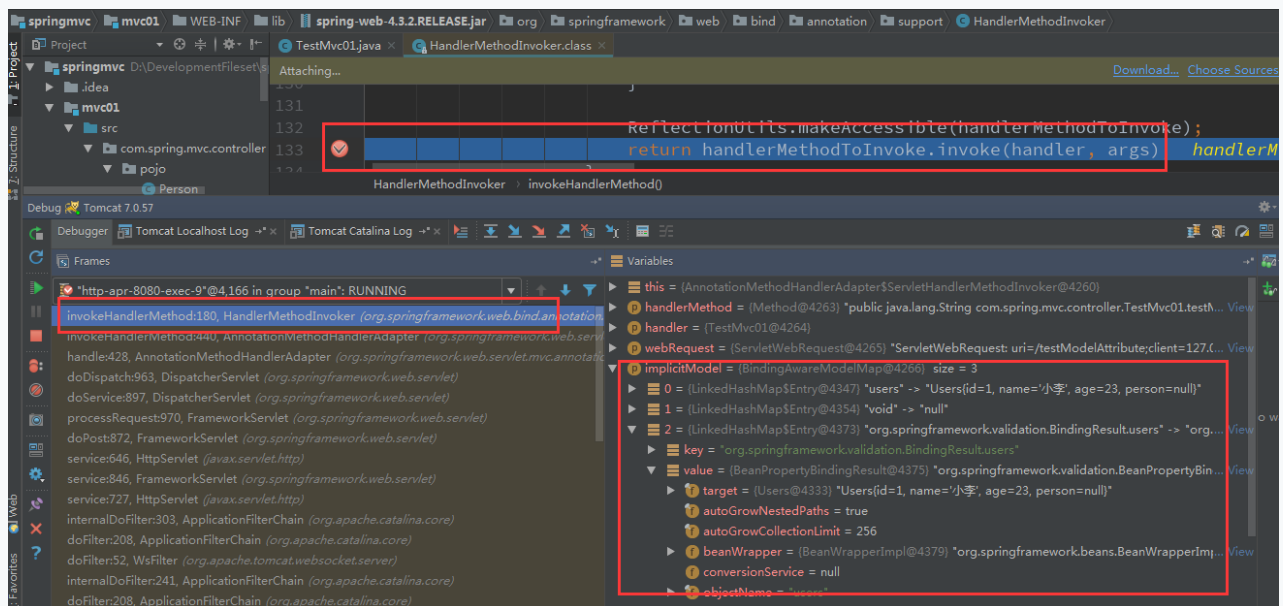
```

```

36.         */
37.         if ("".equals(attrName)) {
38.             Class<?> resolvedType =
GenericTypeResolver.resolveReturnType(attributeMethodToInvoke, handler
.getClass());
39.             attrName = Conventions.getVariableNameForReturnType
(attributeMethodToInvoke, resolvedType, attrValue);
40.         }

```

3、执行DUG的时候当断点走到 handlerMethodToInvoke.invoke发现对应的值进行赋值了



4、那么这个值是怎么赋值上去的内？

5、在当前方法中this.resolveHandlerArguments有传入个参数implicitModel将对应的值放入在该参数中，改变对应的值

```

1.     Object[] args = this.resolveHandlerArguments(handlerMethodToInvoke,
handler, webRequest, implicitModel);

```

2、点击resolveHandlerArguments对应的类

1、在进行打上对应的断点

```

1.     private Object[] resolveHandlerArguments(Method handlerMethod,

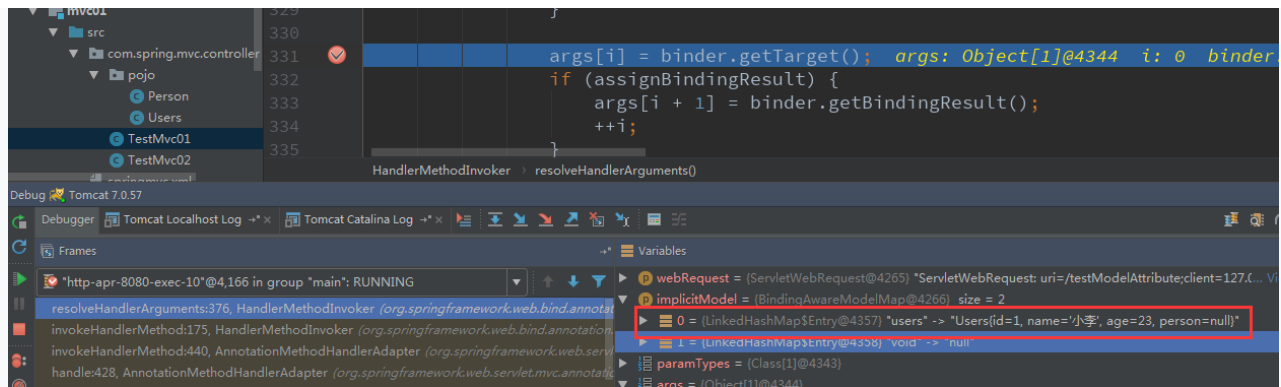
```

```

2.     ....) {
3.         //中间代码省略.....
4.     } else if (attrName != null) {
5.         WebDataBinder binder = this.resolveModelAttribute(attrName, methodParam, implicitModel, webRequest, handler);
6.         boolean assignBindingResult = args.length > i + 1 && Errors.class.isAssignableFrom(paramTypes[i + 1]);
7.
8.         if (binder.getTarget() != null) {
9.             /**
10.                进行DUG打上断点
11.            */
12.            this.doBind(binder, webRequest, validate, validationHints, !assignBindingResult);
13.        }
14.        /**
15.            进行DUG打上断点
16.
17.            当DUG运行到这里，就进行赋值，
18.        */
19.        args[i] = binder.getTarget();
20.        if (assignBindingResult) {
21.            //表单的请求参数赋给了 WebDataBinder 的 target 对应的属性.
22.            args[i + 1] = binder.getBindingResult();
23.            ++i;
24.        }
25.        /**
26.            进行DUG打上断点
27.        */
28.        //会把 WebDataBinder 的 attrName 和 target 给到 implicitModel. 近而传到 request 域对象中
29.        implicitModel.putAll(binder.getBindingResult().getModel());
30.    }
31.    }
32.    /**
33.        进行DUG打上断点
34.    */
35.    return args;
36. }

```

2、DUG运行



3、那么具体是怎么赋值上的内？

4、点击传binder的对象resolveModelAttribute

```

1. //点击resolveModelAttribute
2. WebDataBinder binder = this.resolveModelAttribute(attrName, methodParam, implicitModel, webRequest, handler);

```

3、点击resolveModelAttribute

1、打上对应的断点

```

1. private WebDataBinder resolveModelAttribute(String attrName,
2. MethodParameter methodParam, ExtendedModelMap implicitModel,
3. NativeWebRequest webRequest, Object handler) throws Exception {
4.     /**
5.         进行DUG打上断点
6.     */
7.     String name = attrName;
8.     if ("".equals(attrName)) {
9.         name = Conventions.getVariableNameForParameter(methodParam)
10.    ;
11.    }
12.    //中间代码省略.....
13.    /**
14.        进行DUG打上断点
15.    */
16.    WebDataBinder binder = this.createBinder(webRequest, bindObject
17.    , name);
18.    /**
19.        进行DUG打上断点
20.    */
21.    }

```

```
17.         this.initBinder(handler, name, binder, webRequest);
18.         return binder;
```

2、当看到binder是通过this.createBinder所创建的

3、点击createBinder查询源码

1、this.createBinder

1、一步步的点击对象直到对应的实现DataBinder类，当到这里可以看核心的参数就是target，和objectName

```
1.         public DataBinder(Object target, String objectName) {
```

2、返回到resolveModelAttribute中查询target，和objectName是怎么生成的

```
1.
2.         private WebDataBinder resolveModelAttribute(String attrName,
3.             MethodParameter methodParam, ExtendedModelMap implicitModel,
4.             NativeWebRequest webRequest, Object handler) throws Exception {
5.             //attrName可以使用注解@ModelAttribute(value = "users" )进行指定
6.             //没有这事ObjectName为类名的第一个字母小写
7.             String name = attrName;
8.             /**
9.                 进行DUG打上断点
10.            */
11.            if ("".equals(attrName)) {
12.                //获取全类名的,就是@ModelAttribute注解方法的model传入对象的key名称
13.                name = Conventions.getVariableNameForParameter(methodParam);
14.            };
15.            /**
16.                进行DUG打上断点
17.                进行判断implicitModel中是否有name
18.            */
19.            if (implicitModel.containsKey(name)) {
20.                bindObject = implicitModel.get(name);
21.            }
22.            -----
23.            若不存在：则验证当前 Handler 是否使用了 @SessionAttributes 进行修饰，若使用了，
24.            则尝试从 Session 中
```



```

22.  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
23.  -----
24.  若 Handler 没有使用 @SessionAttributes 进行修饰，或 @SessionAttributes 中
    没有使用 value 值指定的 key
25.  和 attrName 相匹配，则通过反射创建了 POJO 对象
26.  -----
27.          */
28.      } else if (this.methodResolver.isSessionAttribute(name, paramType)) {
29.          bindObject =
    this.sessionAttributeStore.retrieveAttribute(webRequest, name);
30.          if (bindObject == null) {
31.              this.raiseSessionRequiredException("Session attribute '"
    " + name + "' required - not found in session");
32.          }
33.          //中间代码省略.....

```

3、String name = attrName;可以在进行指定在请求的方法上

```

1.  @RequestMapping("/testModelAttribute")
2.  public String testModelAttribute(@ModelAttribute(value ="users" ) Users users)

```

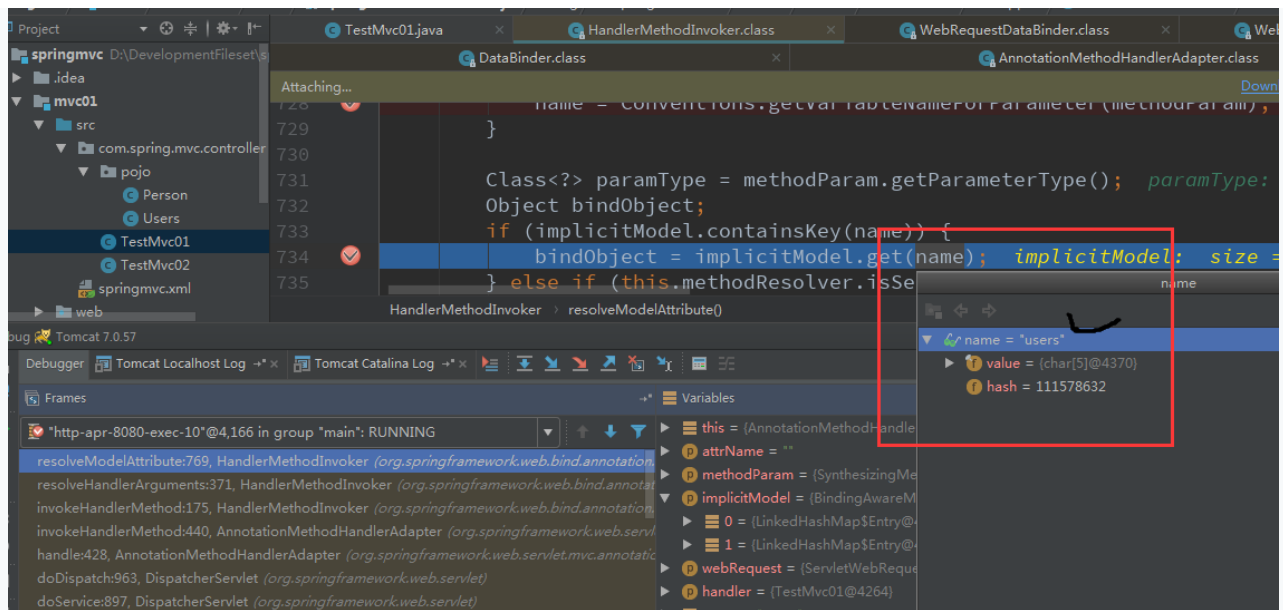
预览在resolveHandlerArguments方法中

```

1.  private Object[] resolveHandlerArguments(Method handlerMethod,.....
2.      //中间代码省略.....
3.  else if (ModelAttribute.class.isInstance(paramAnn)) {
4.      //可以使用注解@ModelAttribute(value ="users" )进行指定
5.          ModelAttribute attr = (ModelAttribute)paramAnn;
6.          attrName = attr.value();
7.          //中间代码省略.....

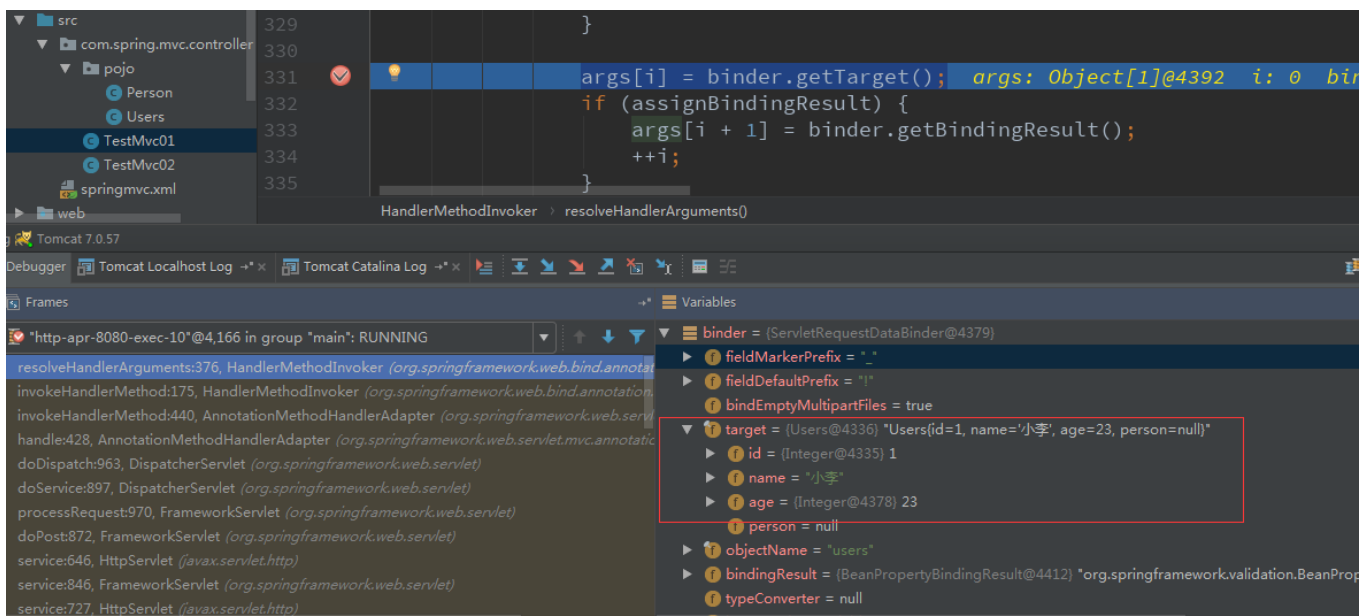
```

4、运行DUG



5、当程序执行到binder.getTarget()时候在resolveHandlerArguments方法中,会发现Target进行赋值了

```
1. private Object[] resolveHandlerArguments(Method handlerMethod,
2.     ....) {
3.     //中间代码省略.....
4.     args[i] = binder.getTarget();
```



6、当程序执行到implicitModel.putAll(binder.getBindingResult().getModel())时进行传入了参数，点击.getModel()

```

1. private Object[] resolveHandlerArguments(Method handlerMethod,
2. ....) {
3.     //中间代码省略.....
4.     implicitModel.putAll(binder.getBindingResult().getModel());
5. }

```

1、implicitModel.putAll(binder.getBindingResult().getModel())

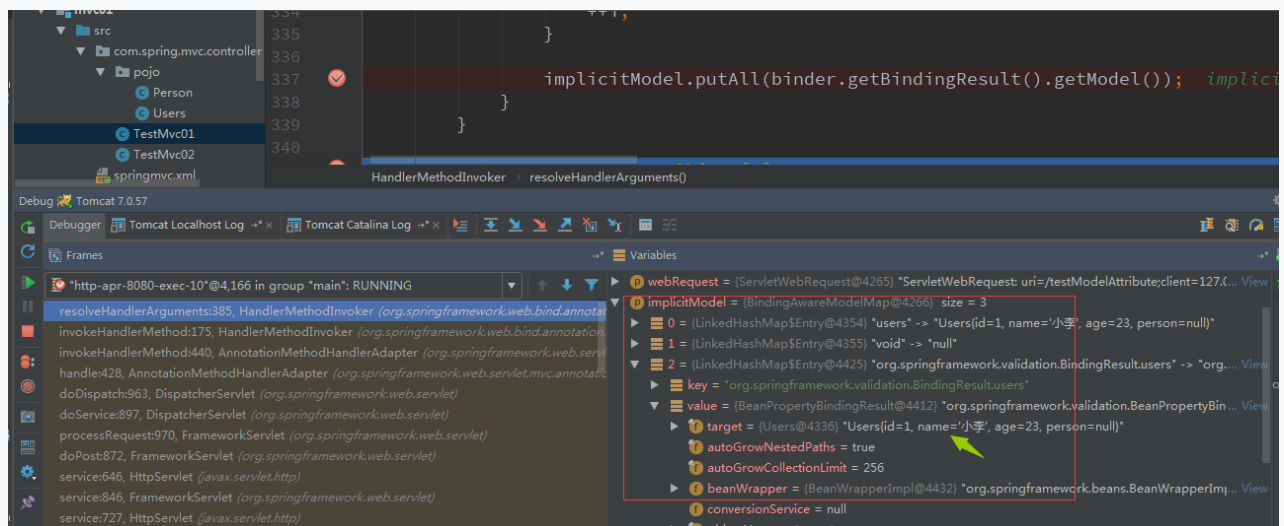
1、点击.getModel()

2、直到实现类(Ctrl+Alt+B)AbstractBindingResult

```

1. public abstract class AbstractBindingResult extends AbstractErrors
   implements BindingResult, Serializable {
2.     private final String objectName;
3.     public Map<String, Object> getModel() {
4.         //这里将binder中的ObjectName和Target进行返回了
5.         Map<String, Object> model = new LinkedHashMap(2);
6.         model.put(this.getObjectName(), this.getTarget());
7.         model.put(MODEL_KEY_PREFIX + this.getObjectName(), this);
8.         return model;
9.     }

```



4、入参的过程

1、SpringMVC 确定目标方法 POJO 类型入参的过程

1) 、 确定一个 key

- 1、若目标方法的 POJO 类型的参数木有使用 @ModelAttribute 作为修饰, 则 key 为 POJO 类名第一个字母的小写
- 2、若使用了 @ModelAttribute 来修饰, 则 key 为 @ModelAttribute 注解的 value 属性值

2) 、 implicitModel 中查找 key

- 1、在 implicitModel 中查找 key 对应的对象, 若存在, 则作为入参传入
- 2、若在 @ModelAttribute 标记的方法中在 Map 中保存过, 且 key 和 @ModelAttribute 确定的 key 一致, 则会获取到.
- 3、若 implicitModel 中不存在 key 对应的对象, 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 没有标识 @SessionAttributes 注解或 @SessionAttributes 注解的 value 值中不包含 key, 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