ModelDriven 和 Preparable 拦截器

[友情提示：学习相关课程后使用此文档效果更佳]

1 提出问题

1.1 如果不使用 ModelDriven 和 Preparable 拦截器技术，Action 类需要包含 JavaBean 中的业务属性，例如：

```
public class StudentAction ──→ Action 类
        extends ActionSupport
        implements RequestAware{

    private static final long serialVersion

    private String stuId;
    private String stuName;
    private String stuAge;
    private String stuSchool;

    private Map<String, Object> requestMap;
    private Dao dao;

public String add(){

    Student student = new Student();
    student.setStuAge(stuAge);
    student.setStuName(stuName);
    student.setStuSchool(stuSchool);
    dao.save(student);

    return SUCCESS;
}
```

⇨ 与 JavaBean 中的属性代码重复，是冗余代码

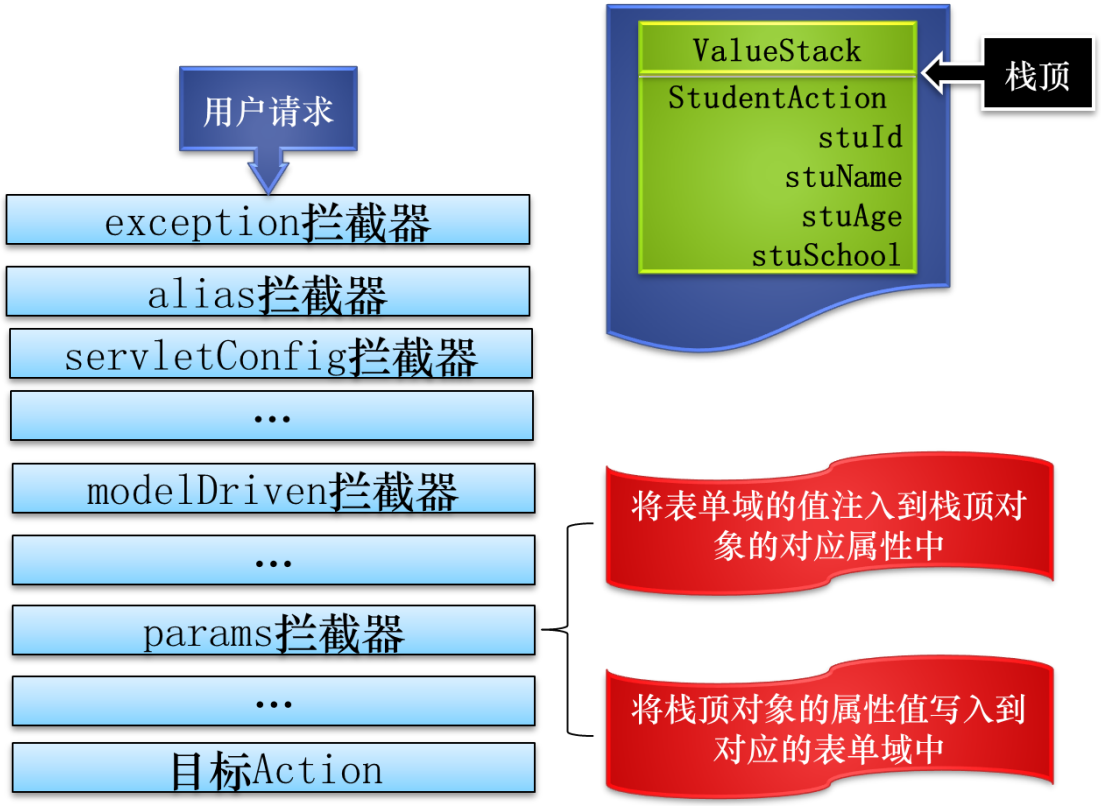
⇨ 面向过程的实现方法

1.2 目标：将冗余代码抽取出来，把操作一个一个散列的属性变为操作一个对象——在这里就是封装了业务信息的领域模型：

JavaBean，以实现模块化编程。

2 分析问题

2.1 当前的情况下，目标 Action 的实例对象会被压入值栈栈顶，params 拦截器会将表单域中的值注入到目标 Action（也就是栈顶对象）对应的属性中。



2.2 我们的需求就是：**如何在 params 拦截器工作之前把 JavaBean 压入值栈栈顶？**

3 解决问题

3.1 使用 ModelDriven 拦截器

3.2 使用方法：

①让我们的 Action 方法实现 **com.opensymphony.xwork2.ModelDriven** 接口，如图所示：

泛型参数指定所使用的 JavaBean 的类型

```
public class StudentAction
        extends ActionSupport
        implements RequestAware,ModelDriven<Student>{
```

②实现 T getModel();方法

[1]首先在 Action 类中声明一个 JavaBean 的引用

```
private Student student;
```

[2]创建一个 JavaBean 的实例对象，在 getModel()方法中返回

```
@Override
public Student getModel() {
    student = new Student();
    return student;
}
```

3.3 修改 Action 类

删除冗余代码，修改 Action 方法

add()方法：

```
public String add(){
    dao.save(student);
    return SUCCESS;
}
```

update()方法

```
public String update(){
    dao.update(student);
    return SUCCESS;
}
```

delete()方法

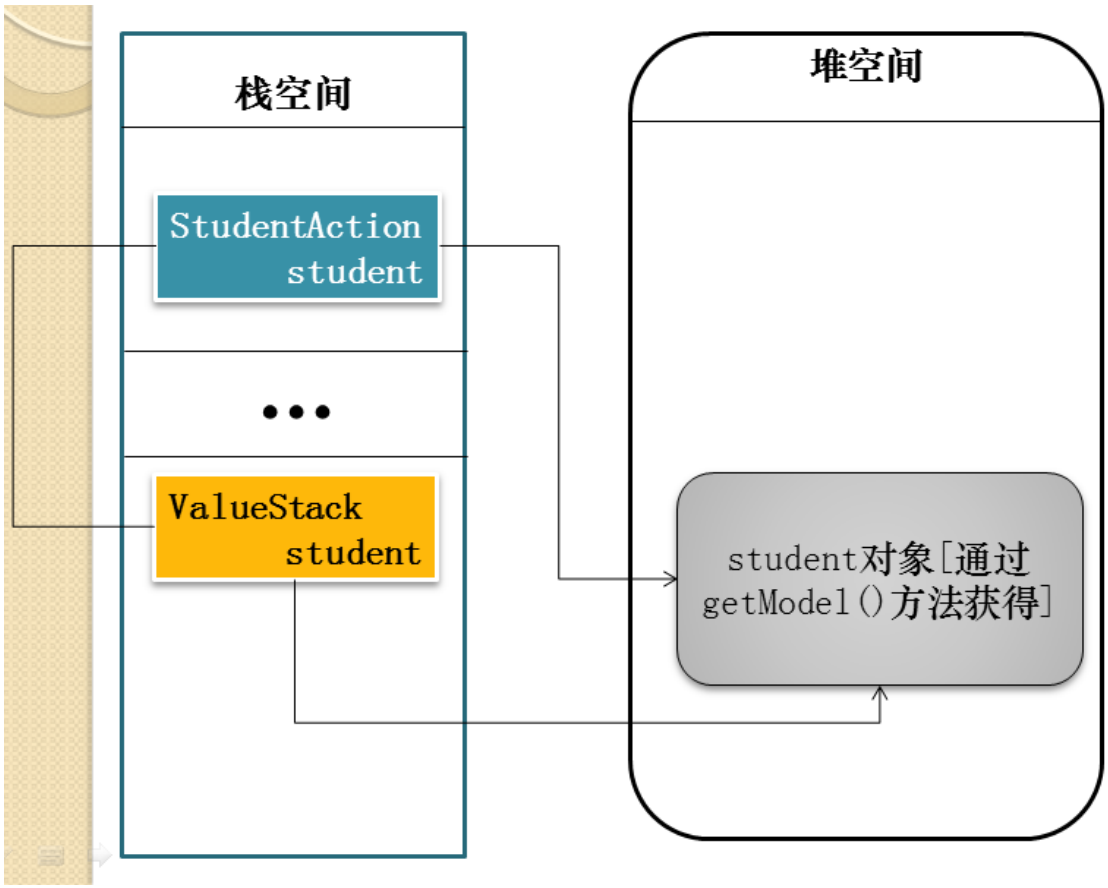
```
public String delete(){
    dao.delete(student.getStuId());
    return SUCCESS;
}
```

edit()方法

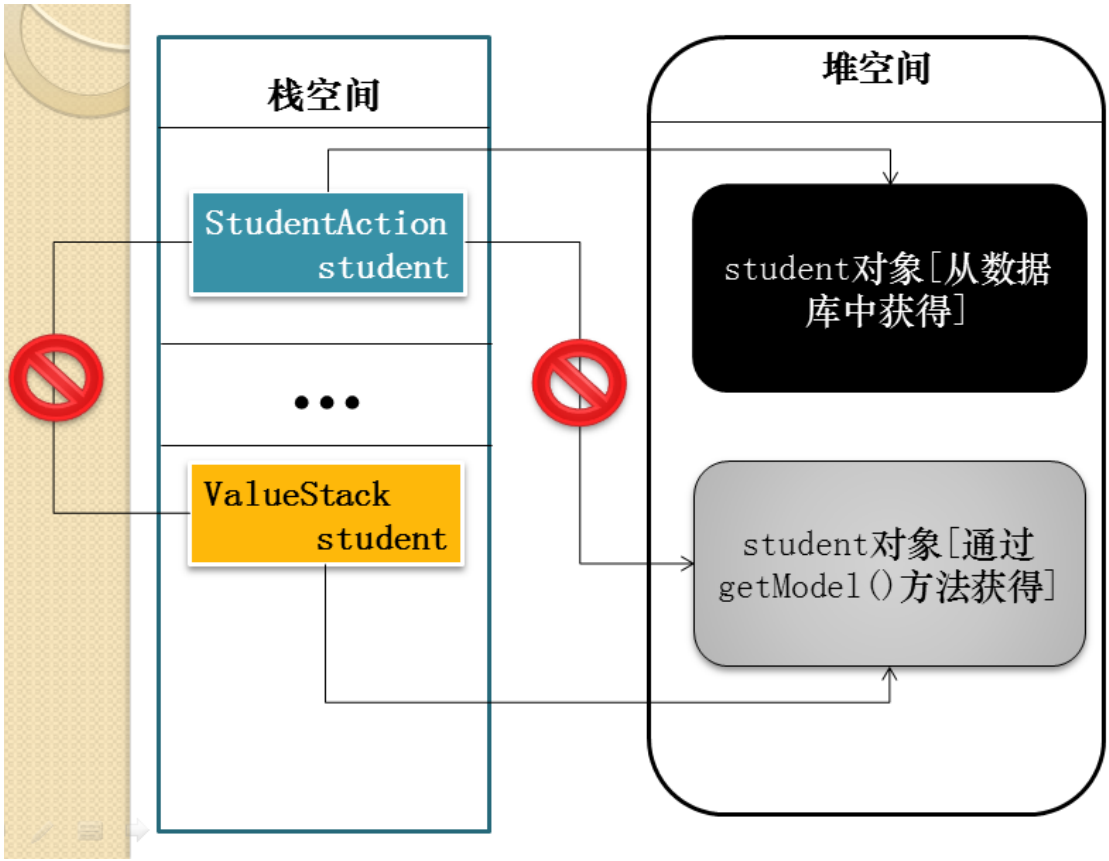
```
//注意：这样写edit()方法无法实现表单回显，为什么？
public String edit(){
    //为什么从数据库中取出的数据没有回显到表单域中？
    student = dao.get(student.getStuId());
    return "editPage";
}
```

原因分析：

①ModelDriven 拦截器工作之后，Action 类中的 JavaBean 和值栈栈顶的 JavaBean 指向的都是 getModel()方法返回的那个对象，如图所示：



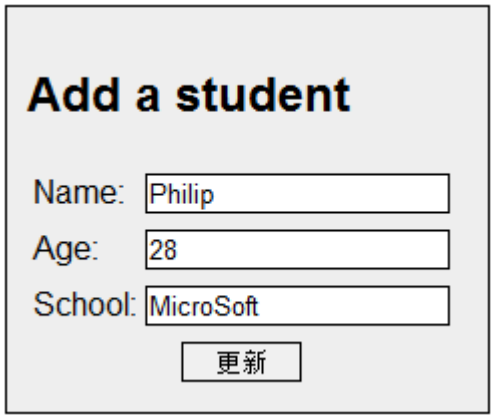
②执行 edit()方法后，Action 类中的 JavaBean 指向的就是从 Dao 中取出的对象，而此时值栈栈顶的对象指向的还是原来那个对象，没有被注入从数据库中取出的数据，而表单回显依靠的是栈顶对象



此时 edit()方法正确的写法是：

```
public String edit(){
    //先将从Dao中取回的对象缓存起来
    Student stuData = dao.get(student.getStuId());
    //然后将每一个属性注入到栈顶对象对应的属性中，显然这也是面向过程的解决方法，需改进
    student.setStuId(stuData.getStuId());
    student.setStuName(stuData.getStuName());
    student.setStuAge(stuData.getStuAge());
    student.setStuSchool(stuData.getStuSchool());
    return "editPage";
}
```

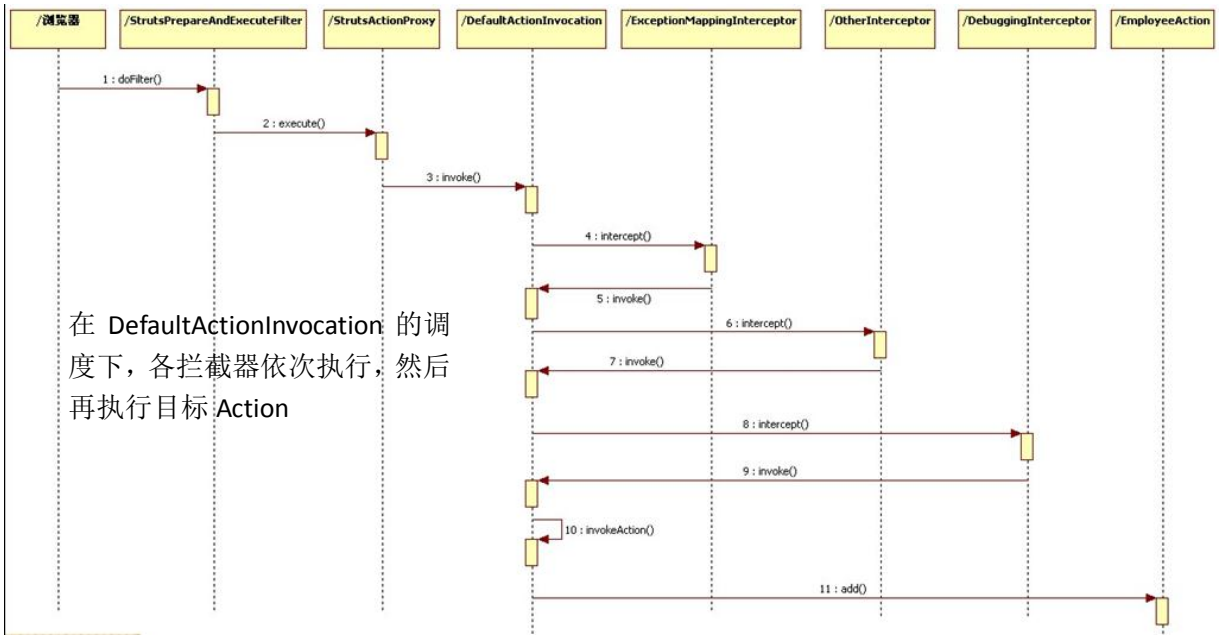
正确回显：



遗留问题：edit()方法过于繁琐，还停留在面向过程的实现方式。如何改进？

4 原理剖析

4.1 Struts2 运行机制



[该图片请放大查看]

4.2 拦截器和拦截器栈

①打个比方：我们去医院体检，需要检查很多个项目，比如：

测血压、胸透、做心电图、测身高、测体重、测肺活量等等，经历这一系列的检查后，最终才能拿到体检报告。

这里体检的项目就相当于我们 Struts2 里面的拦截器，若干个拦截器组织在一起就是一个拦截器栈，最终的体检报告相当于目标 Action。

②在 Struts2 中，默认的拦截器栈是：struts-default.xml 中定义的 defaultStack，这里只列举了其中一部分

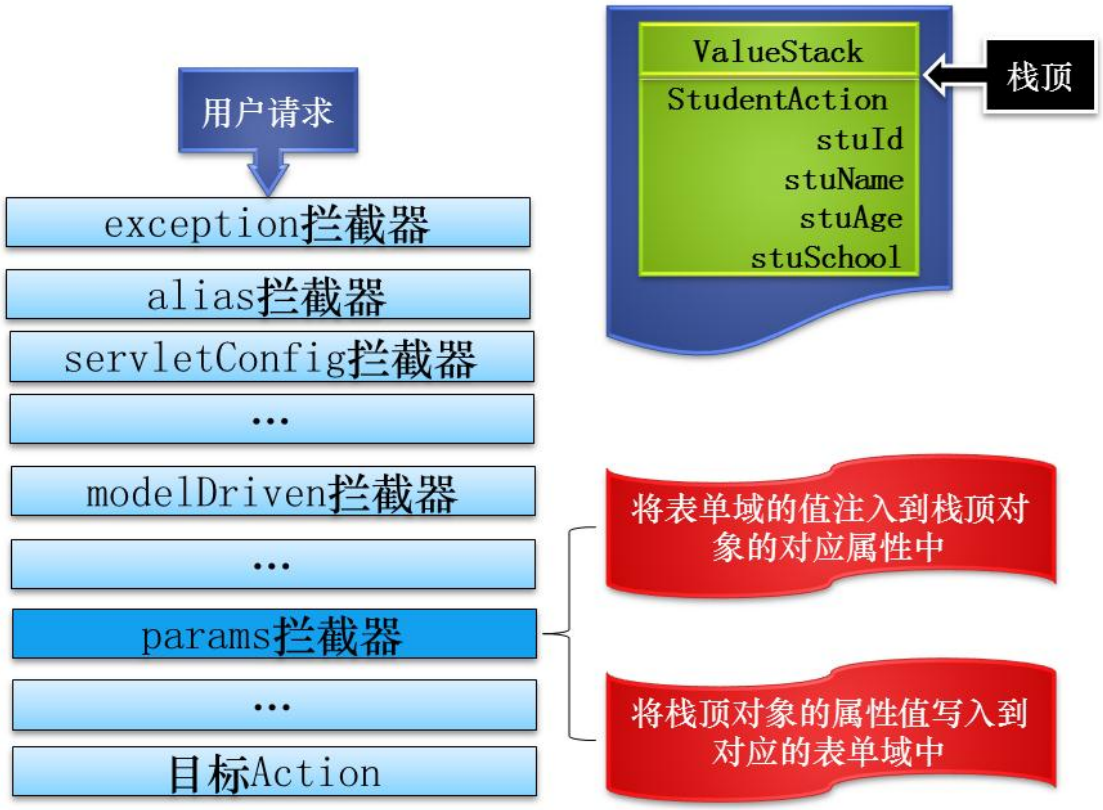
```
<interceptor-stack name="defaultStack">
    <interceptor-ref name="exception"/>
    <interceptor-ref name="alias"/>
    <interceptor-ref name="servletConfig"/>
    <interceptor-ref name="i18n"/>
    <interceptor-ref name="prepare"/>
    <interceptor-ref name="chain"/>
    <interceptor-ref name="scopedModelDriven"/>
    <interceptor-ref name="modelDriven"/>
                ……
    <interceptor-ref name="params">
                ……
</interceptor-stack>
```

4.3 params 拦截器

params 拦截器有两项功能

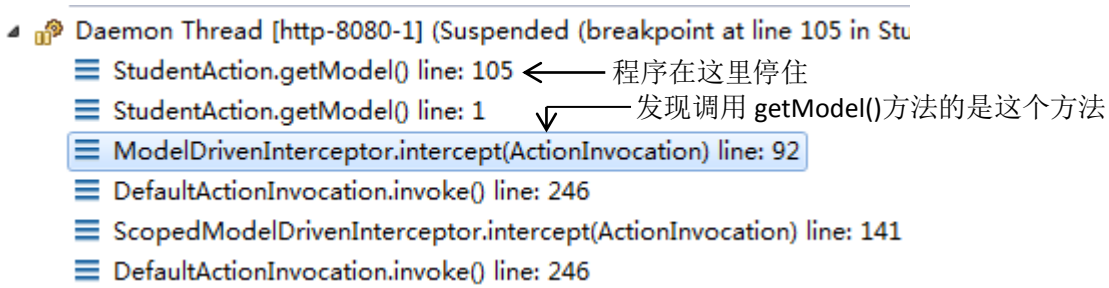
①提交表单时将表单域中的值注入到栈顶对象对应的属性中

②目标页面为表单时，将栈顶对象的属性回显到对应的表单域中



4.4 ModelDriven 拦截器

getModel()方法是由谁来调用的呢？通过在 getModel()方法内设置断点，以 Debug 模式运行项目发现：



找到 ModelDrivenInterceptor.intercept()方法：

```
@Override
public String intercept(ActionInvocation invocation) th
    Object action = invocation.getAction();
    if (action instanceof ModelDriven) {
        ModelDriven modelDriven = (ModelDriven) action;
        ValueStack stack = invocation.getStack();
        Object model = modelDriven.getModel();
        if (model !=  null) {
            stack.push(model);
        }
        if (refreshModelBeforeResult) {
            invocation.addPreResultListener(new Refresh
        }
    }
    return invocation.invoke();
}
```

获取目标 Action

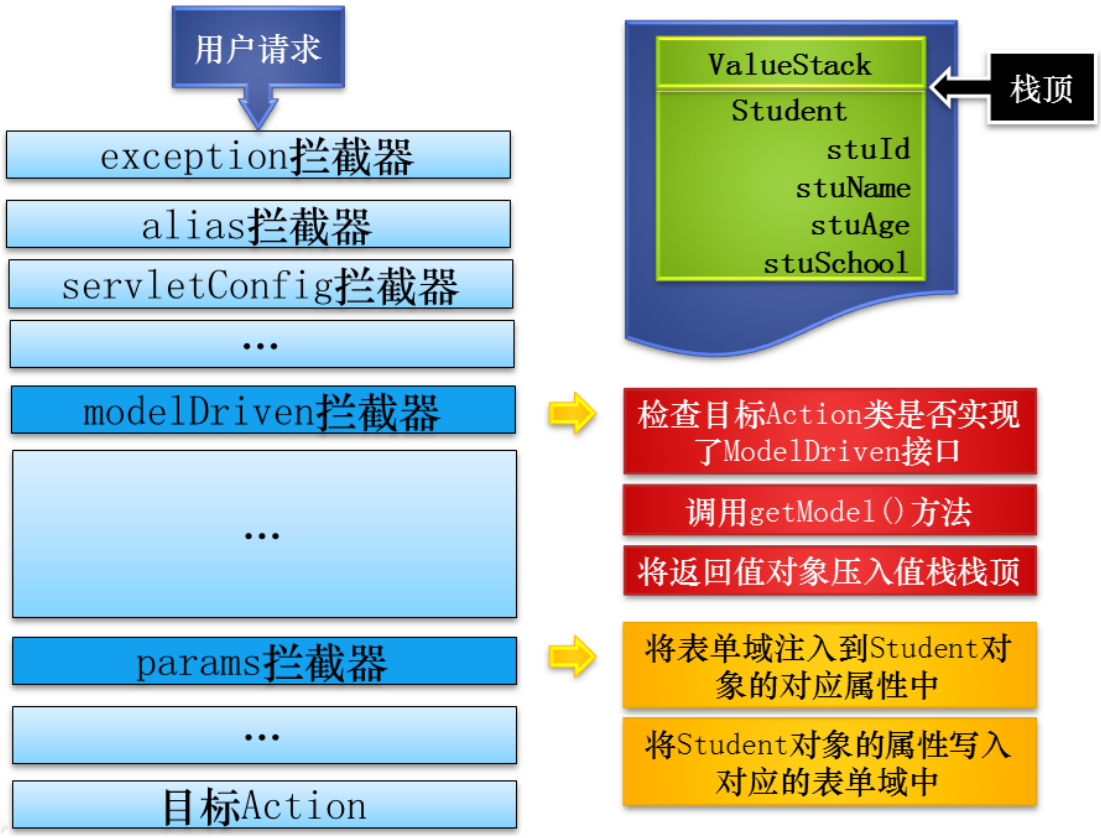
如果目标 Action 实现了 ModelDriven 接口

获取值栈

调用getModel()方法

将返回值压入值栈，最后压入的值一定在栈顶

总体流程：



5 改进 edit()方法

5.1 问题分析

问题产生的原因是：在 getModel()方法中将一个手动创建的空对象提供给 ModelDriven 拦截器，单调死板。

5.2 需求：当目标 action 方法是 edit()时，将从数据库中取出的 JavaBean 压入栈顶，当目标 action 方法是 add()、update()时将一个空的 JavaBean 压入栈顶

5.3 解决问题

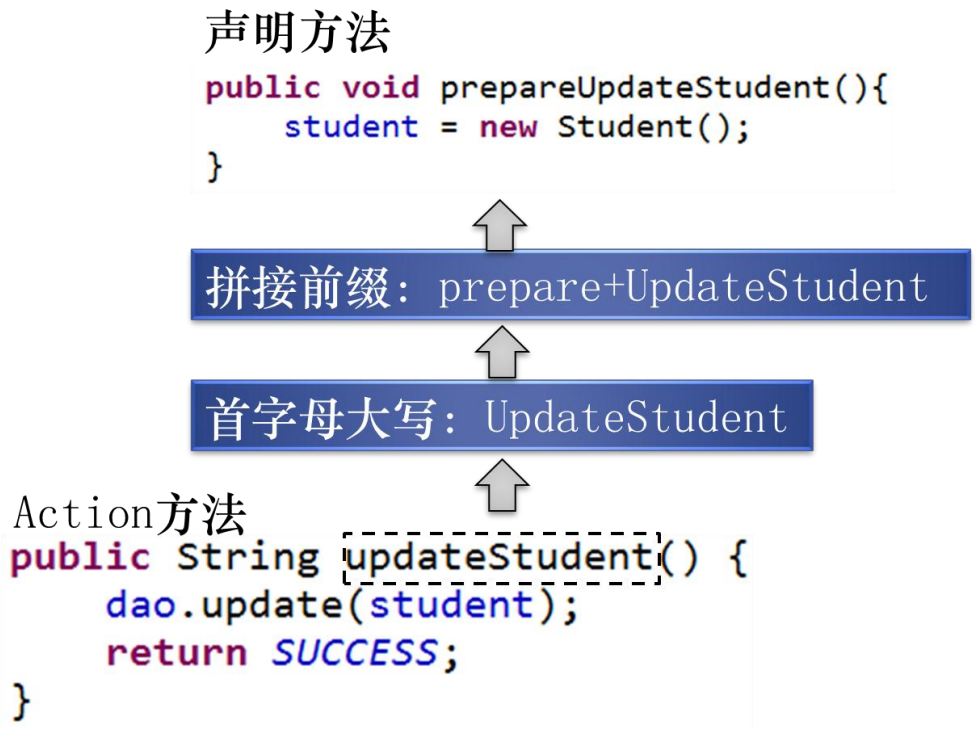
借助 Preparable 拦截器，为 ModelDriven 拦截器“准备”不同 action 方法所需要的不同 JavaBean

5.4 Preparable 拦截器使用方法：

①Action 类实现 com.opensymphony.xwork2.Preparable 接口

②为不同的 action 方法声明专门的 prepare 方法

为特定的Action设定prepare方法



③修改 StudentAction 类代码

[1]prepare 方法

```
//为不同的action方法声明专门的prepare()方法
//1.为addStudent()方法声明prepare方法
public void prepareAddStudent(){
    student = new Student();
}
//2.为updateStudent()方法声明prepare方法
public void prepareUpdateStudent(){
    student = new Student();
}
//3.为updateStudent()方法声明prepare方法
public void prepareEditStudent(){
    student = dao.get(stuId);
}
```

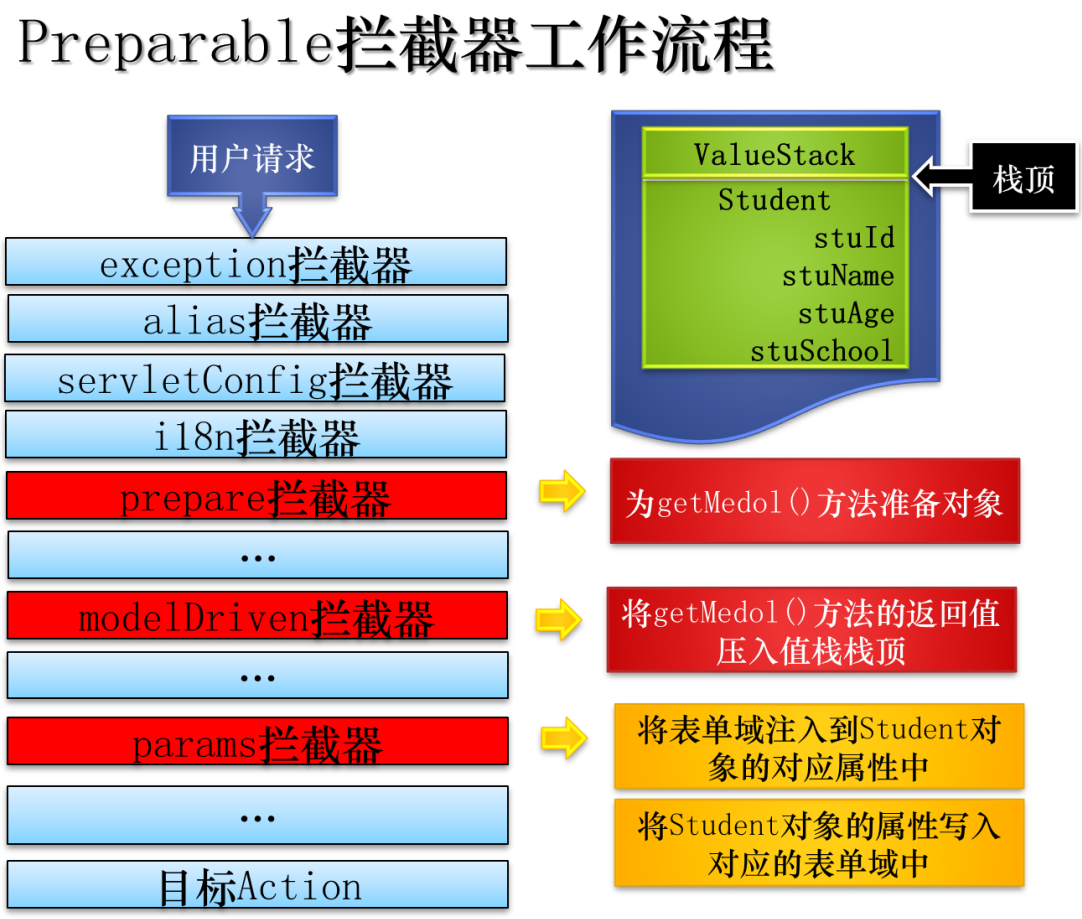
[2]getModel()方法

```
@Override
public Student getModel() {
    return student;
}
```

[3]action()方法

```
//3.为updateStudent()方法声明prepare方法
public void prepareEditStudent(){
    student = dao.get(student.getStuId());
}
```

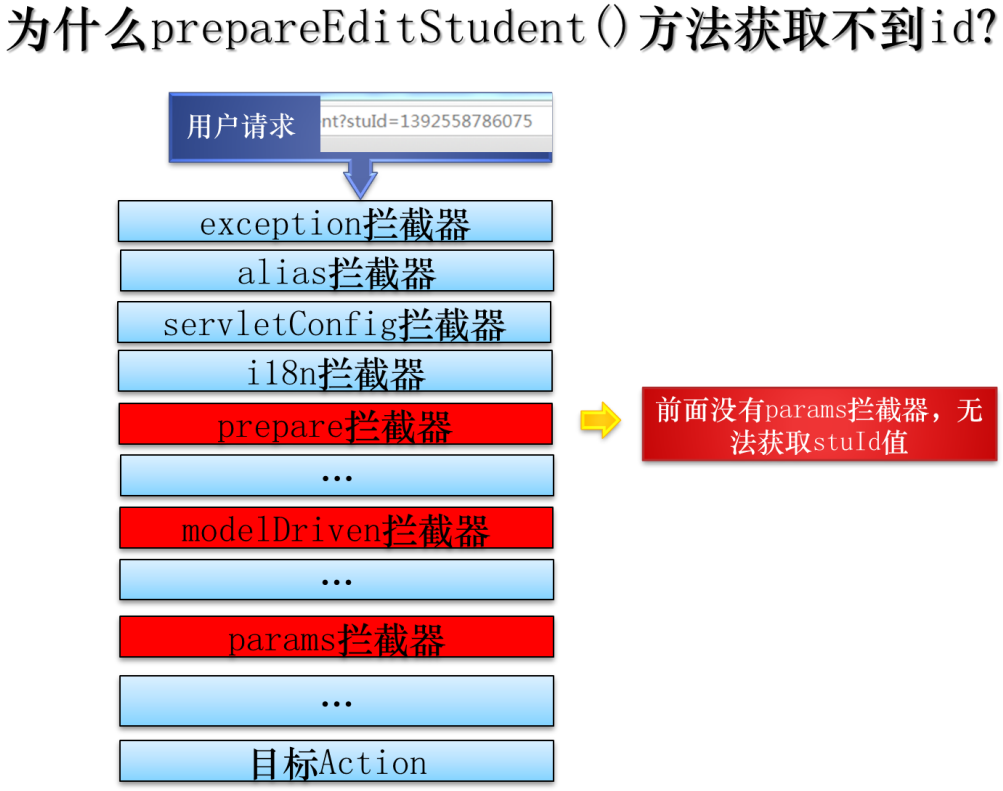
5.5 Preparable 拦截器工作流程



5.6 自定义 prepare 方法的命名格式有什么玄机？

PrepareInterceptor拦截器工作原理：图示

5.7 深情的呼唤：急需在 prepare 拦截器之前有一个 params 拦截器



5.8 *paramsPrepareParamsStack* 拦截器栈

paramsPrepareParamsStack 从字面上理解来说， 这个 stack 的拦截器调用的顺序为：首先 params，然后 prepare，接下来 modelDriven，最后再 params

Struts 2.0 的设计上要求 modelDriven 在 params 之前调用，而业务中 prepare 要负责准备 model，准备 model 又需要参数，这就需要在 prepare 之前运行 params 拦截器设置相关参数，这个也就是创建 paramsPrepareParamsStack 的原因。

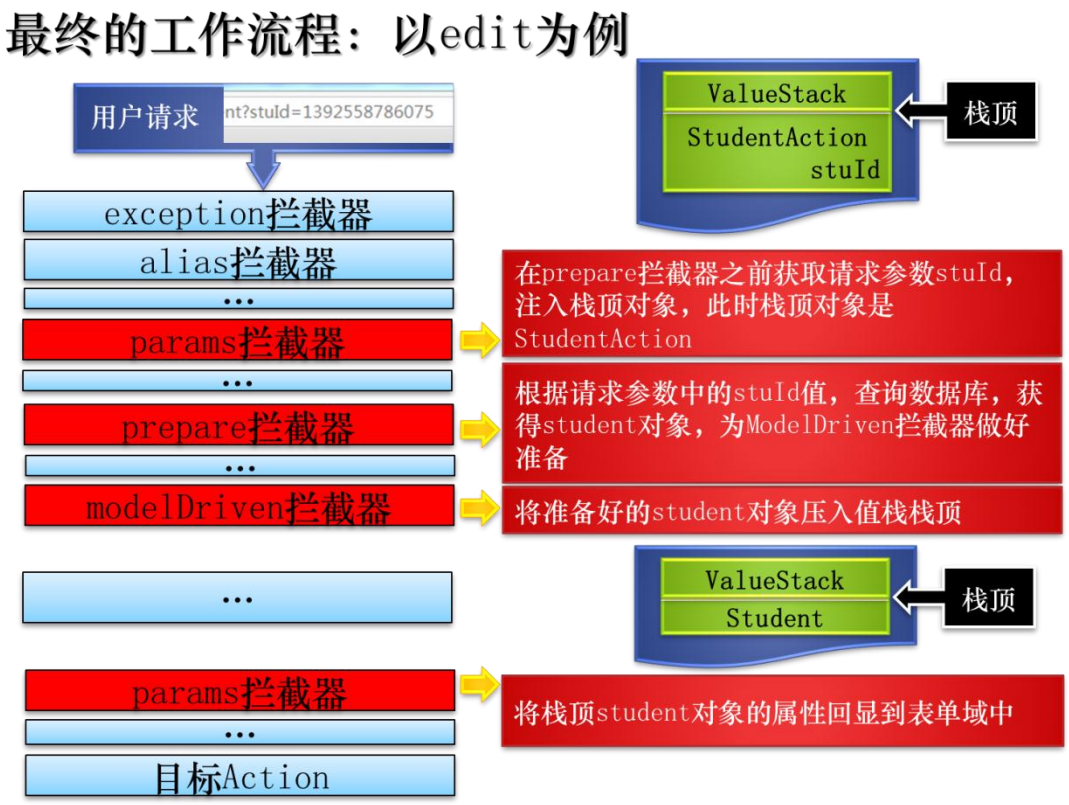
配置方法

在 struts.xml 文件中，package 标签下添加

```
<default-interceptor-ref name="paramsPrepareParamsStack">
</default-interceptor-ref>
```

5.9 paramsPrepareParamsStack 拦截器栈工作流程

5.10 最终的工作流程



6 补充：如何取消 prepare()方法的执行

在 struts.xml 文件中 action 标签内配置：

```
<!-- 覆盖 prepare 拦截器的 alwaysInvokePrepare 参数值为 false -->
<interceptor-ref name="paramsPrepareParamsStack">
    <param name="prepare.alwaysInvokePrepare">false</param>
</interceptor-ref>
```