# 谈谈Java框架理解

我想针对框架来谈一谈自己的理解。

**从DRY原则开始**
Don't Repeat Yourself，不要重复你的代码。

如果有一些代码我们重复了两次，我们总是要把它们提取出来变成一个方法。
渐渐的我们有了一批方法，我们把它整合成工具类。如DBUtils，我相信很多人都写过。
相应的，工具类还可以整合成类库。类库一般以JAR的形式部署，更容易使用。

以上仍然是自己从底层一点一点的做起，如果有现成的类库可以使用不更好吗？
不要重复造“轮子”，总会有专业造“轮子”的，我们只需要直接使用已有的类库即可。
如Apache Commons项目，这简直成了Java标准库的标准扩展了。

框架也是一样
框架，是为了解决某个特定领域的问题而诞生的。
框架，是为了我们不必总是写相同代码而诞生的。
框架，是为了让我们专注于业务逻辑而诞生的。

框架把我们程序设计中不变的部分抽取出来，让我们专注于与业务有关的代码。
而著名的SSH三大框架就是为了解决Java Web开发中的问题。

**J2EE的发展**
本来网站都是一个个静态HTML的，但很快我们就不满足于这样了。动态网页应运而生。

一开始是Servlet。其代码类似于下面这样。
主要是Java代码，然后用out一点一点输出HTML代码。
当然代码无比丑陋，且美工人员几乎不能理解这样的代码。

```
out.write("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">\r\n");
out.write("<html>\r\n");
out.write("<head>\r\n");
out.write("<title>首页</title>\r\n");
out.write("</head>\r\n");
out.write("<body>\r\n");
out.write("Hello, " + new Date().toLocalString
out.write("</body>\r\n");
out.write("</html>\r\n");
```

所以，很快JSP应运而生。其代码类似于下面这样。
JSP很像是正常的HTML代码中，藏了一些Java代码。
就这样Java Web技术疯狂发展了。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>首页</title>
</head>
<body>
```

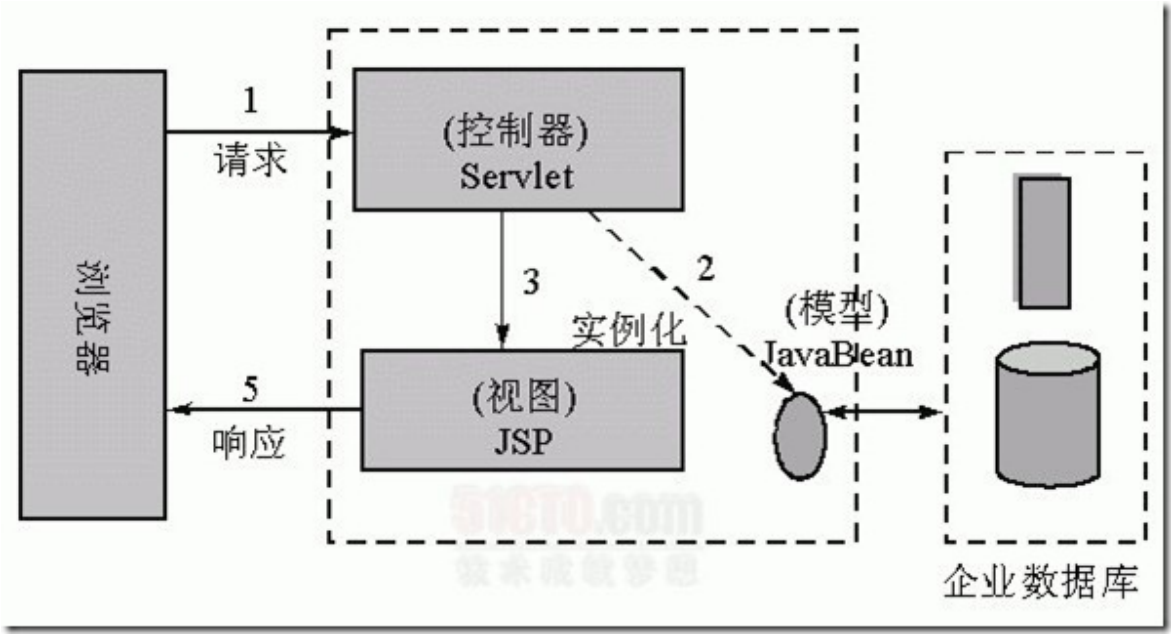
```
<body>

        <ss:a label="首页" href="/index.jsp" />
        <ss:a label="登入" href="/login.jsp" />
        <ss:a label="注册" href="/register.jsp" />
        <ss:a label="登出" href="/logout.do" />
        <hr />

        <jsp:include page="/include/jsp/datetime.jsp" />
        <%
                String user = (String) session.getAttribute("user");
                if (user != null && user.length() > 0) {
                        out.println("Welcome, <b>" + user + "</b>! You are logged in.");
                }
        %>

</body>
</html>
```

随着实际Web应用的使用越来越广泛，Web应用的规模也越来越大，开发人员发现动态Web应用的维护成本也越来越大，即使只需要修改该页面的一个简单按钮文本，或者一段静态的文本内容，也不得不打开混杂的动态脚本的页面源文件进行修改——这是一种很大的风险，完全有可能引入新的错误。

**MVC模式的诞生**

这时候，人们意识到：单纯使用JSP页面充当过多角色是相当失败的选择，这对于后期的维护相当不利，慢慢地开发人员开始在Web开发中使用MVC模式。

Java阵营慢慢的发展出了适合Java Web开的的MVC模式。（看下图）
使用JSP作为视图，Servlet作为控制器，JavaBean作为模型。
通过合理的分层，使得Java程序员更能驾驭大型网站的开发。




使用MVC模式有一个问题，Servlet中获取到的数据如何传递到视图层呢?

```
//一般采取这种办法
//在Servlet中将结果设置到request对象当中
```

```
//在Servlet中将结果设置到request对象当中。
request.setAttribute("result", result);

//在JSP中通过request对象来获取。
<% Object result = request.getAttribute("result"); %>
```

可是JSP既然是视图层，如果嵌入太多JAVA代码的话，对于美工的工作相当不利。
一般来说视图层是不允许使用<% %>来嵌入JAVA代码的。
可是视图层也确实需要负责进行一些逻辑判断，来动态输出页面。

当然，Java提供了一个办法，使用“**自定义标签**”，和“**EL表达式**”。

```
<table>
    <th>
      <td>序号</td>
      <td>姓名</td>
      <td>性别</td>
      <td>年龄</td>
      <td>电话</td>
    </th>
  <c:forEach var="item" items="${request.list}">
    <tr>
      <td>${item.id}</td>
      <td>${item.name}</td>
      <td>${item.sex}</td>
      <td>${item.age}</td>
      <td>${item.phone}</td>
    </tr>
  </c:forEach>
</table>
```

视图层的问题看起来不太大，现在来看看控制层。

```
//首先，你需要在Web.xml中进行配置
<servlet>
  <servlet-name>LoginServlet</servlet-name>
  <servlet-class>com.tee.servlet.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>LoginServlet</servlet-name>
  <url-pattern>/Login</url-pattern>
</servlet-mapping>
```

```
//然后，需要编写一个继承自HttpServlet类的Servlet。
package com.tee.servlet;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletEx
                String username = request.getParameter("username");
```

```
                String password = request.getParameter("password");
                if (UserDao.validate(username, password)) {
                        request.getSession.setAttribute("user", username);
                        response.setContentType("text/html; charset=utf-8");
                        response.sendRedirect("/index.jsp");
                } else {
                        ServletContext sc = getServletContext();
                        RequestDispatcher rd = sc.getRequestDispatcher("/login.jsp");
                        request.setAttribute("errMsg", "login failure");
                        rd.forward(request, response);
                }
        }

        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletE
                doGet(request,response);
        }
}
```

这样一来，首先Web.xml会越来越大，越来越难以维护。
且修改Web.xml的风险很大，只要一个错误，整个网站都有可能无法访问。

上面的例子没有进行传入参数的检验，如是否为空，是否有是规定长度，是否是规定的字符。也没有涉及到参数转换，如转换成Date类型，转换成int类型。一旦发生错误又该如何传递给前端，前端又该如何接受错误信息，如何展示出来。

因为Servlet必须继承自HttpServlet，且内部与Servlet Api高度耦合，不但难以测试，而且难以重用。更不要说我们总是需要书写大量与业务逻辑无关的代码。

**MVC框架的诞生**
同样的代码写两次就是罪过，所以我们需要什么东西来让我们解放出来。
很快人们就发展出了一套MVC框架，特别是适合于Java Web的MVC框架。

MVC框架能帮我们做什么？

> 这里引用“四刘”的回答：
> 框架和普通库的几点区别里，我认为最重要的一点是控制反转。
> 框架**规定了开发者写哪些代码 / 不写哪些代码，怎么写代码——这就是框架主要解决的问题。**

MVC框架实现了MVC模式。什么意思？
意思是只要你根据框架的要求填充代码，你就能够很简单的实现MVC模式。

谁来响应用户请求？框架可能告诉你，Action就是用来响应用户请求的。
不用再继承HttpServlet，代码中也可以完全脱离Servlet Api。复用度高，可单元测试。

谁负责生成响应界面？框架可能告诉你，可以用一个JSP文件来生成界面。
也可以用其他视图技术，JfreeChart，FreeMarke，JasperReports，JSF，Tiles，Vlocit等。

如何将网址匹配到Action？框架可能告诉你，在XML文件中配置，且可以分模块配置。
如何确定该返回那个JSP？框架可能告诉你，在XML文件中配置好了，Action返回SUCCESS，INPUT，LOGIN等等即可。

Action如何接受参数？框架可能告诉你，写个Setter方法，就可以接受相应参数。不用再从request获取，且类型可自动转换。根据配置处理编码问题。
Action如何与视图交互？框架可能告诉你，Action自动与视图绑定，在Action写一个Getter方法，试图层就可以用自定义标签获取其值。

输入校验如何进行？写个validate方法，有错误就调用addActionError方法，自动返回配置中，INPUT指定的页面，页面用<s:actionerrors/>标签就可以自动输出错误信息。
也可以使用addFieldError方法添加特定field的错误，使用<s:fielderror/>输出特定field的错误信息。
甚至使用<s:textfield>方法生成的输入框，可以自动显示该字段的错误信息。

**使用框架的必要性**
需要指出的是，只有在相当规模的程序中应用这些框架才能体会到好处。
如果你的网站、程序很小，那么没有必要杀鸡用牛刀。

**需要学习框架吗**
那么新手需要学习框架吗？我不建议新手一上来就学习框架。
就好像新手不要一上来就使用IDE，一定要用JDK自己编几回代码后才能知道IDE帮我们做了什么。

那么新手也不要一开始就使用框架，否则你就不知道框架帮你做了什么，而这并不利于程序员的成长。在学习阶段，先试试不用框架我该如何做，然后会更明白框架的意义。

如果你自己不经历过大型程序的开发，很难理解IoC框架存在的必要性。
我直接new一个不就行了吗？AOP有什么用？声明式事务有什么好处？

如果你不懂AOP，不懂编程式事务，不懂ThreadLocal，那你就很难理解基于注解的声明式事务，你只会感觉到他很神奇，你就很难理解为什么要使用sf.getCurrentSession()，而不是sf.openSession()。很难理解Hibernate经常遇到的懒加载异常。

**最后再谈一谈 框架 和 类库的不同。**

先说说控制反转
一般程序，程序该如何运转，是由我们程序员全盘掌握的。

程序的一切资源的初始化、资源的销毁、资源的定位查找都是我们自己写代码实现的。
程序的一且流程都可以从我们的代码中依次找到，只要你顺着我的程序主入口开始查找。

但是应用框架后的控制权已经交给了框架。如Java Web框架，我们一般都要在Web.xml文件中配置框架的核心控制器、拦截器。这就是把控制权交出的过程。

就好比说原先你是老板，一切工作都亲力亲为，一切问题都是先提交到你这，然后你分配出去。
有了框架之后，就好比是找了个总经理，一切问题他先处理，需要你处理的才来请示你。
就算是要请示你，也先把材料都整理过了，以你最舒服的形式提交给你。

再说说类库和框架的不同
类库就好像是负面清单，类库里实现的代码你就不用写了，程序需要的其他代码你来写。
关键就是其他代码有多少？类库没告诉你，需要你程序员心理有数。

框架就好像是正面清单，框架直接告诉你，你只需要把这些代码写完，程序就可以正常运行了。

框架就好像建筑里的框架结构一样，有了框架，这个建筑就完成了大半，剩下的就是装修问题了。

Intopass，**Full Stack Developer** 

来源： <http://www.zhihu.com/question/25654738/answer/31302541>