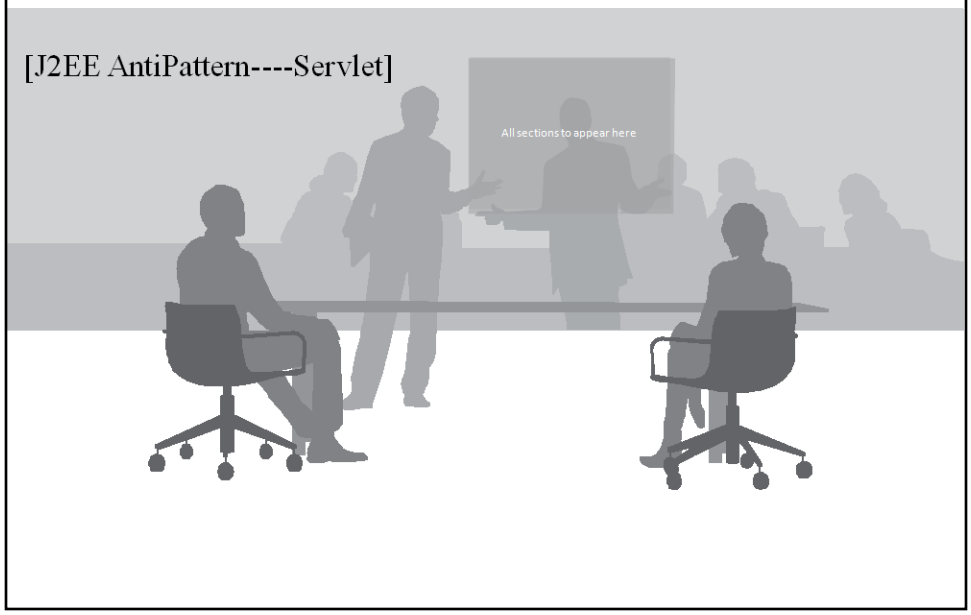
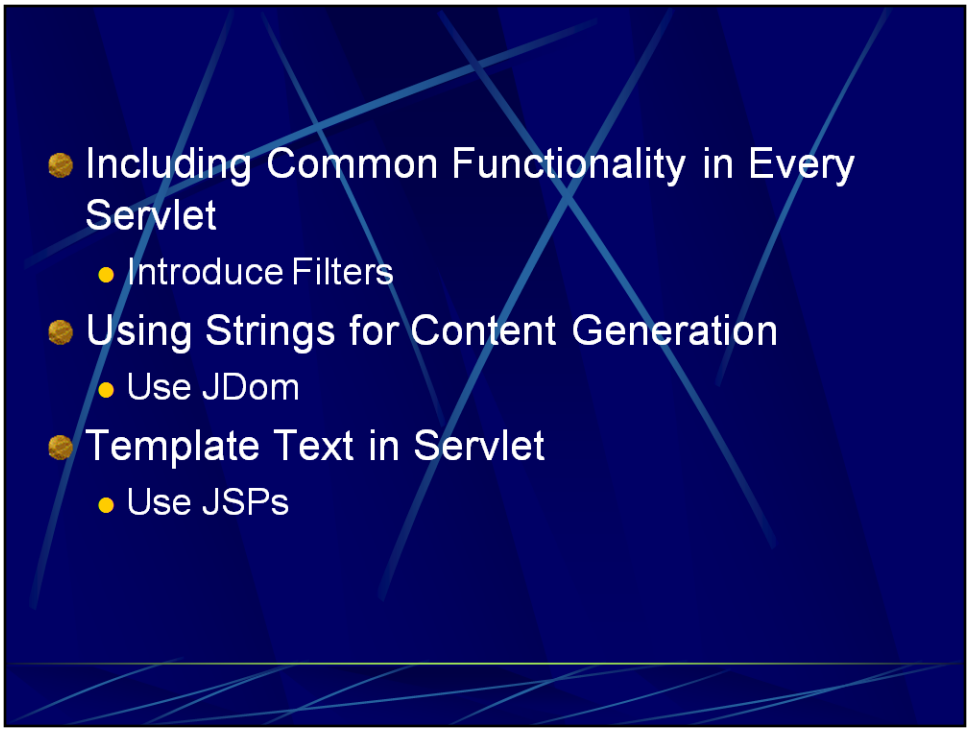
[J2EE AntiPattern----Servlet]



Servlet
J2EE Refactoring Patterns/AntiPatterns

徐迎晓
复旦大学软件学院
xuyingxiao@126.com

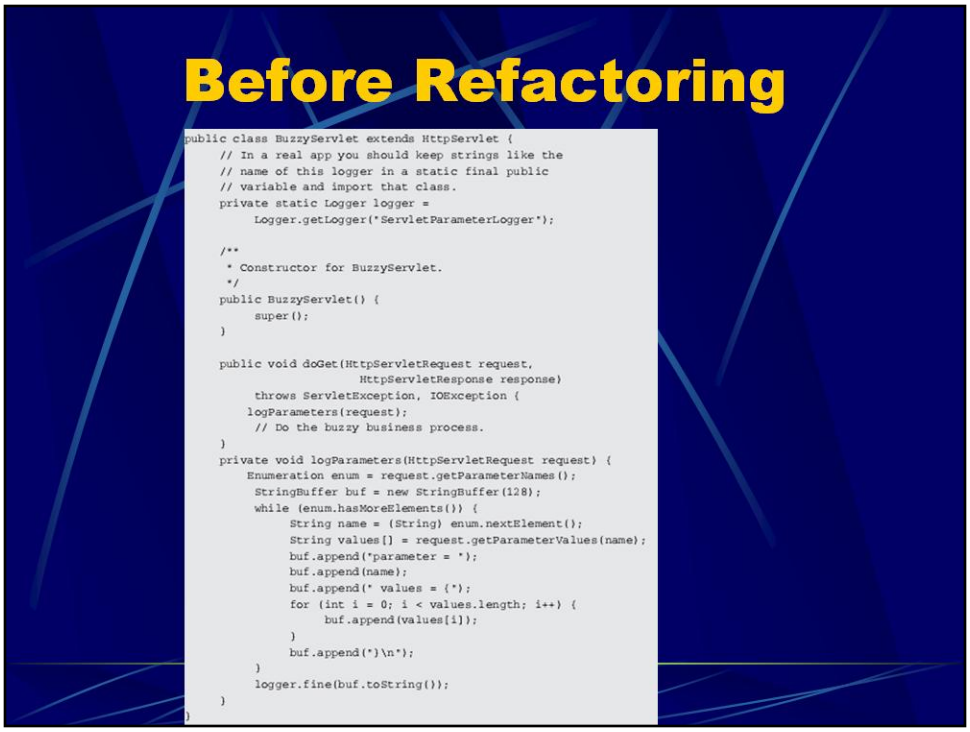


反模式及重构方案

如果尽量Use JSP，如果无法使用JSP，则Use Jdom重构

AntiPattern

- Including Common Functionality in Every Servlet



图中的注释：
//实际项目中Logger.getLogger("ServletParameterLogger");中的字符串应该放在static final public 变量中，并导入(import)含有该变量的类

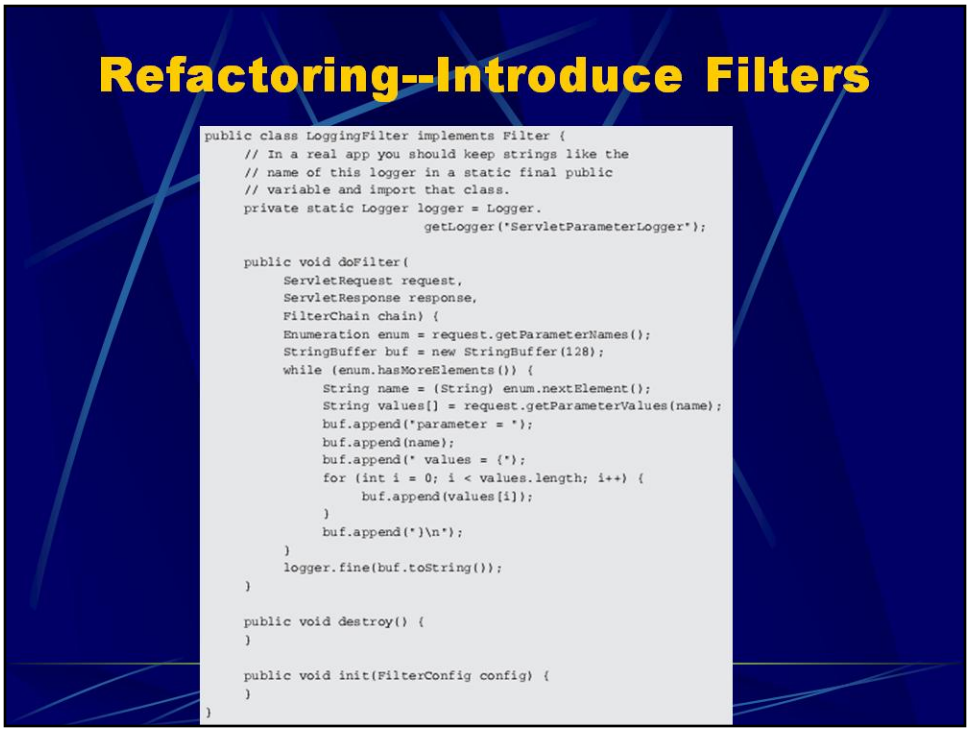
```
public class BuzzyServlet extends HttpServlet {
// In a real app you should keep strings like the
// name of this logger in a static final public
// variable and import that class.
//实际项目中Logger.getLogger("ServletParameterLogger");中的字符串应该放在static final public 变量中，并导入含有该变量的类
                private static Logger logger =
                                Logger.getLogger("ServletParameterLogger");
                /**
                 * Constructor for BuzzyServlet.
                 */
                public BuzzyServlet() {
                                super();
                }
                public void doGet(HttpServletRequest request,
                                                HttpServletResponse response)
                throws ServletException, IOException {
                                logParameters(request);
// Do the buzzy business process.
                }
                private void logParameters(HttpServletRequest request) {
                                Enumeration enum = request.getParameterNames();
                                StringBuffer buf = new StringBuffer(128);
                                while (enum.hasMoreElements()) {
                                                String name = (String) enum.nextElement();
                                                String values[] = request.getParameterValues(name);
                                                buf.append("parameter = ");
                                                buf.append(name);
                                                buf.append(" values = {");
                                                for (int i = 0; i < values.length; i++) {
                                                                buf.append(values[i]);
                                                }
                                                buf.append("}\n");
                                }
                                logger.fine(buf.toString());
                }
}
```

```
public class BuzzyServlet extends HttpServlet {
        private static Logger logger =
                Logger.getLogger("ServletParameterLogger");
        public BuzzyServlet() {
                super();
        }
        public void doGet(...) throws ServletException, IOException {
                logParameters(request);
                        // Do the buzzy business process.
        }
        private void logParameters(HttpServletRequest request) {
              ...
        }
}
```

将与处理代码——日志记录 嵌在doGet( )方法中了——日志记录是硬编码的，没有灵活性。

要调整日志记录格式怎么办? 其他地方也要进行日志怎么办?——多个地方要用日志的话需要拷贝代码，维护困难

实际应用中,除了日志,还会有很多其他公共功能→引入过滤器，见下页



重构后放在过滤器中,然后构建部署描述文件

重构

1.找出多个Servlet中所共有的预处理和后处理代码

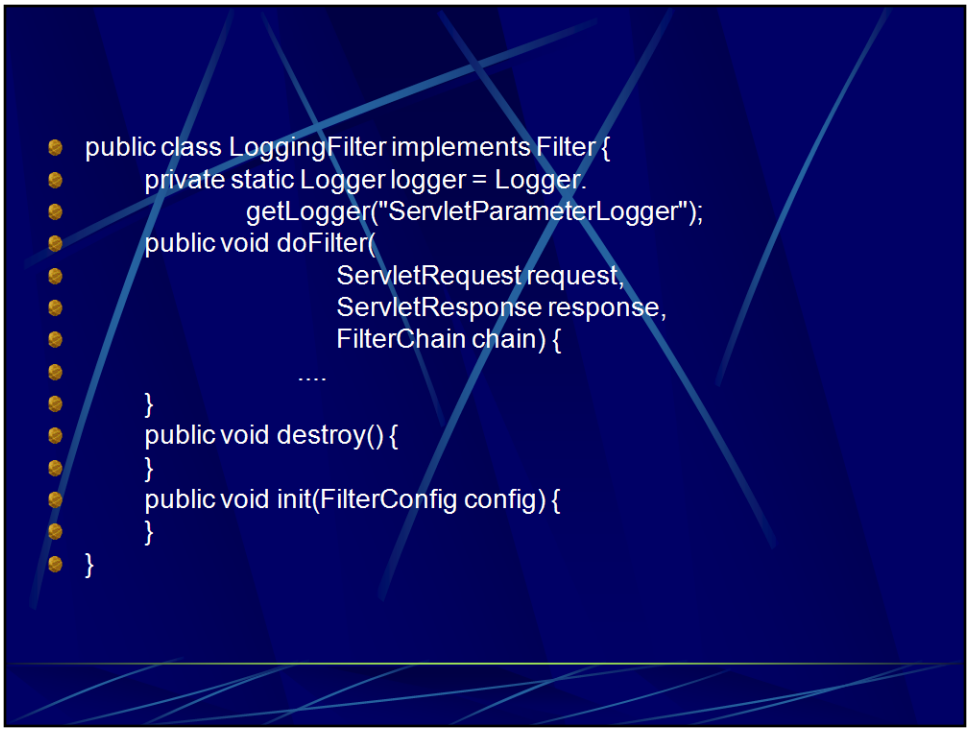
2. 对每个分别

创建过滤器实现类

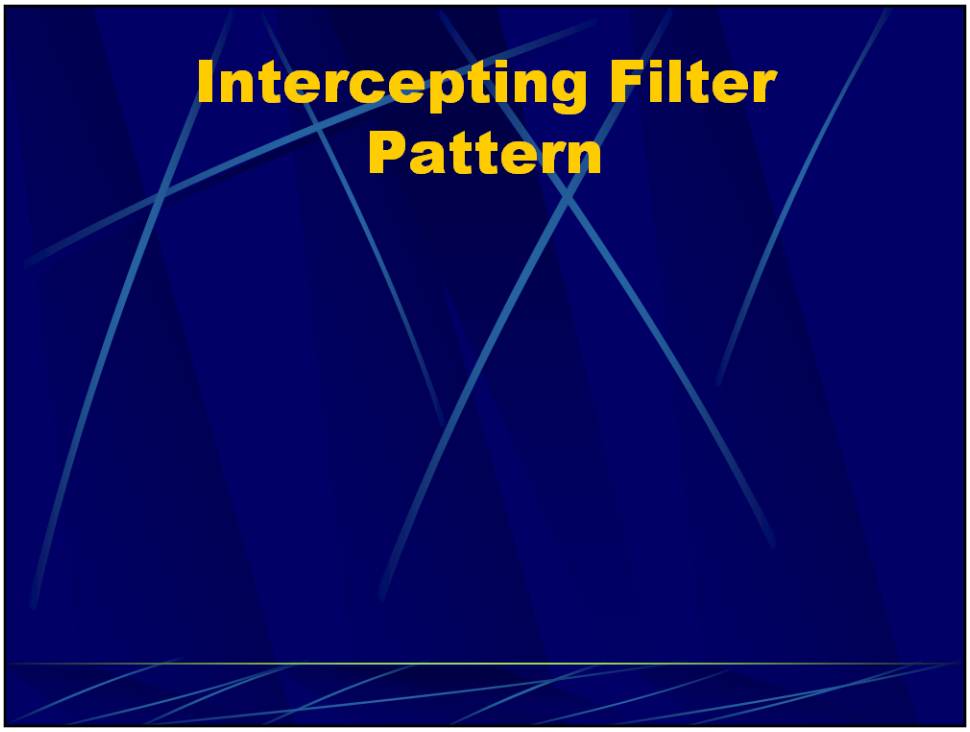
构建部署描述文件

相应Servlet中删除预处理和后处理代码

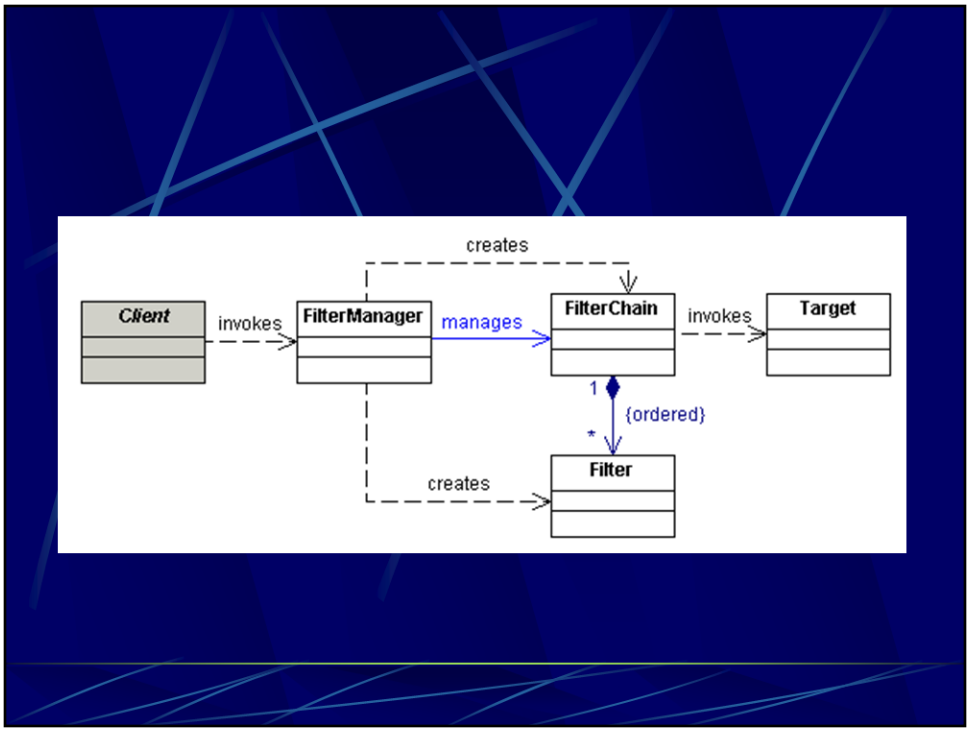
部署和测试

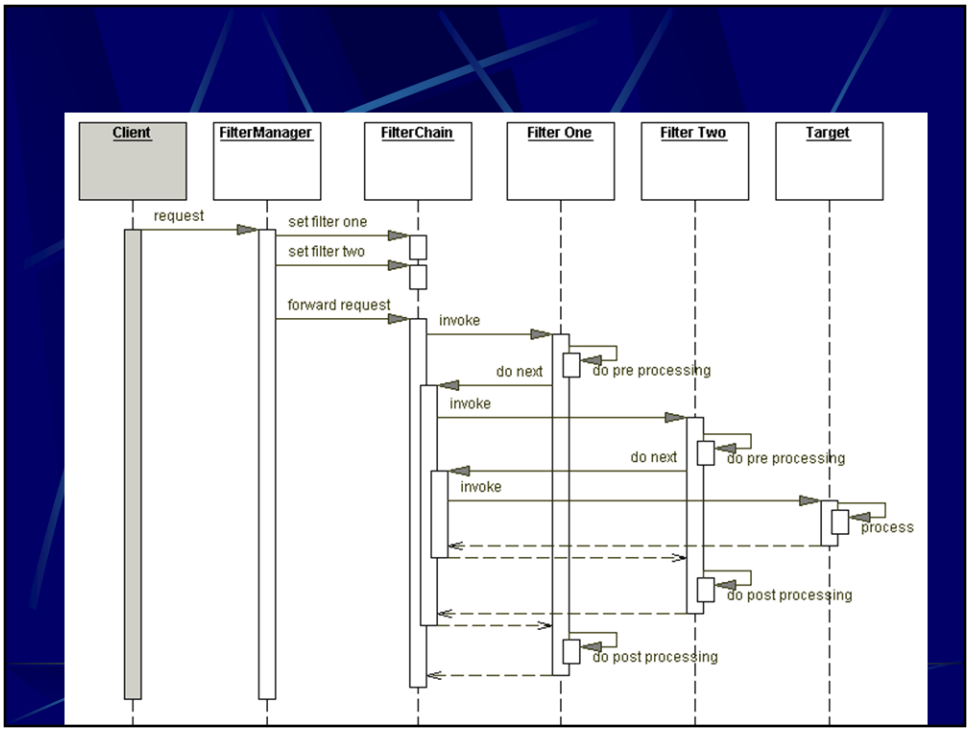



```
public class LoggingFilter implements Filter {
//                  In a real app you should keep strings like the
//                  name of this logger in a static final public
//                  variable and import that class.
                    private static Logger logger = Logger.getLogger("ServletParameterLogger");
                    public void doFilter(
                                                        ServletRequest request,
                                                        ServletResponse response,
                                                        FilterChain chain) {
                                        Enumeration enum = request.getParameterNames();
                                        StringBuffer buf = new StringBuffer(128);
                                        while (enum.hasMoreElements()) {
                                                        String name = (String) enum.nextElement();
                                                        String values[] = request.getParameterValues(name);
                                                        buf.append("parameter = ");
                                                        buf.append(name);
                                                        buf.append(" values = {");
                                                        for (int i = 0; i < values.length; i++) {
                                                                        buf.append(values[i]);
                                                        }
                                                        buf.append("}\n");
                                        }
                                        logger.fine(buf.toString());
                    }
                    public void destroy() {
                    }
                    public void init(FilterConfig config) {
                    }
}
```

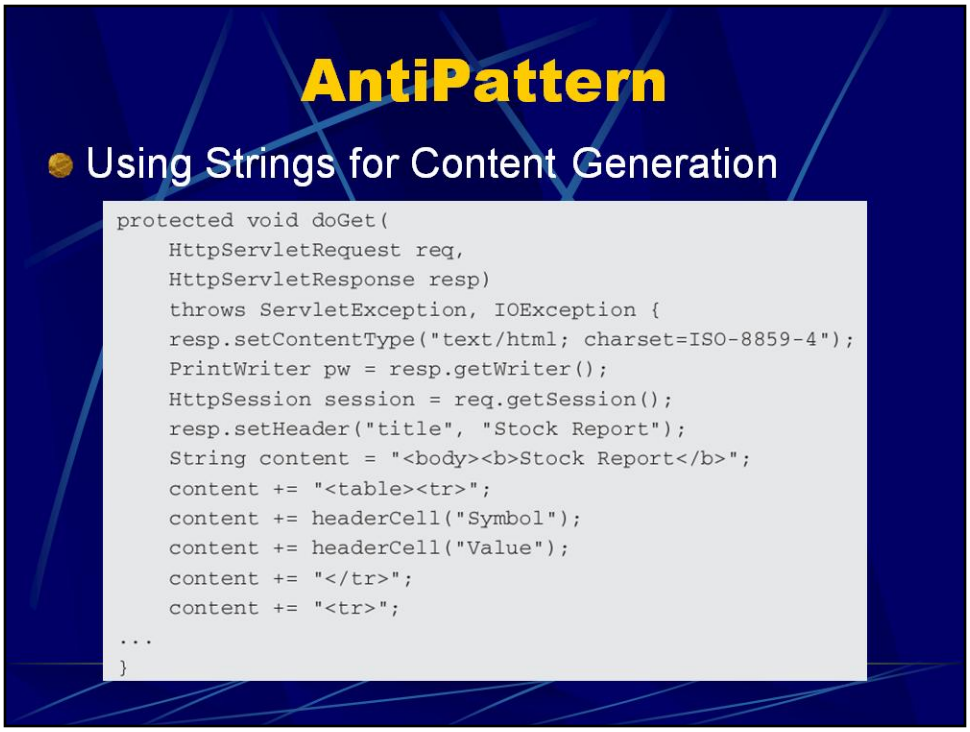


Core J2EE Patterns 拦截过滤器模式


creates
Client
invokes
FilterManager
manages
FilterChain
invokes
Target
1
{ordered}
*
Filter
creates


Client
FilterManager
FilterChain
Filter One
Filter Two
Target
request
set filter one
set filter two
forward request
invoke
do next
do pre processing
invoke
do next
do pre processing
invoke
process
do post processing
do post processing

Servlet 2.3中的标准机制



Servlet中硬编码HTML，标签有问题很难调试。通过Jdom自动生成XHTML比较方便

当然，如果Servlet中能避开HTML最好，可以使用以后介绍的“使用JSP”

例外情况：只有几行HTML

Refactoring--Use JDom

```
protected void doGet(HttpServletRequest req,
                     HttpServletResponse resp)
                     throws ServletException, IOException {
    resp.setContentType("text/html; charset=ISO-8859-4");
    PrintWriter pw = resp.getWriter();
    HttpSession session = req.getSession();
    resp.setHeader("title", "Stock Report");
    Element body = new Element("body");
    Element title = new Element("b");
    title.addContent("Stock Report");
    body.addContent(title);
    Element table = new Element("table");
    Element row = rowCell(table);
...
}
```

重构后

Antipattern

- Template Text in Servlet

```
protected void doGet(HttpServletRequest req,
                     HttpServletResponse resp)
        throws ServletException, IOException {
        resp.setContentType("text/html; charset=ISO-8859-4");
        PrintWriter pw = resp.getWriter();
        HttpSession session = req.getSession();
        // Build the headers and top-level stuff.
        pw.print("<!DOCTYPE HTML PUBLIC \"");
        pw.print("-//W3C//DTD HTML 4.0 Transitional//EN\"");
        pw.print("\"http://www.w3.org/TR/REC-html40/loose.dtd\">");
        pw.print("<html>\n<head>\n");
        pw.print("<title>Template Ski Reports</title>");
        pw.print("</head>");
        // Start the body.
        pw.print("<body>");
        pw.print("<b>Template Text Ski Report</b>");
        pw.print("<table border=\"1\">");
...
}
```

静态HTML比业务逻辑代码多得多。其实由于JSP的出现，Servlet已经不再是生成HTML的最佳技术了。

Refactoring-Use JSPs

```
protected void doGet(
    HttpServletRequest req,
    HttpServletResponse resp)
    throws ServletException, IOException {
    List reports = getReports();
    HttpSession session = req.getSession();
    session.setAttribute("ski.reports", reports);
    session
        .getServletContext()
        .getRequestDispatcher("iBank/SkiReport.jsp")
        .forward(req, resp);
}
```