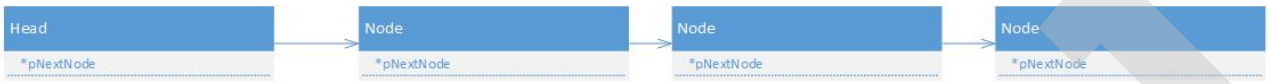
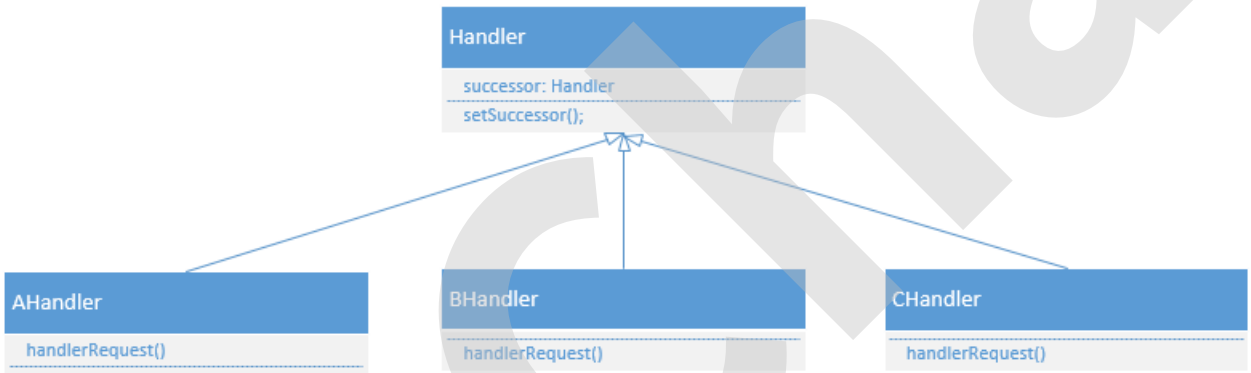
概述

> 责任链模式为某个请求创建一个对象链，每个对象依次检查此请求，并对其进行处理，或者将它传给链中的下一个对象

链表是很常见一种数据结构，链表中的每一个节点都是一个对象，并且该对象中存储着下一个节点的指针。链表的基本结构如下：



责任链模式的结构其实和链表相差无几，唯一的区别就是责任链模式中所有的对象都有一个共同的父类（或接口）：



在责任链模式中，N个Handler子类都处理同一个请求，只不过具体的职责有所差别。

当有一个请求进入时，先经过AHandler的handlerRequest方法，然后再把请求传递给BHandler，B处理完再把请求传递给CHandler，以此类推，形成一个链条。链条上的每一个对象所承担的责任各不相同，这就是责任链模式。

现在我们模拟一个场景：论坛用户发表帖子，但是常常会有用户一些不良的信息，如广告信息，涉黄信息，涉及政治的敏感词等。这时我们就可以使用责任链模式来过滤用户发表的信息。

代码实现

先定义所有责任链对象的父类：

```
/**
 * 帖子处理器
 */
public abstract class PostHandler {

    /**
     * 后继者
     */
    protected PostHandler successor;
```

```
    public void setSuccessor(PostHandler handler){
        this.successor = handler;
    }

    public abstract void handlerRequest(Post post);

    protected final void next(Post post){
        if(this.successor != null){
            this.successor.handlerRequest(post);
        }
    }
}
```

父类Handler主要封装了传递请求等方法，其中要注意的有两点：

1. successor，后继者，这个属性很重要，它保存了责任链中下一个处理器
2. 在next() 方法中（方法名自己随便取），当请求传递到最后一个责任对象时，已经没有后继者继续处理请求了，因此要对successor做判空处理，避免抛出空指针异常。
3. 处理请求的 `handlerRequest` 的入参和返回类型可以根据实际情况修改，可以在该方法中抛出异常来中断请求

广告处理器：

```
/**
 * 广告处理器
 */
public class AdHandler extends PostHandler {

    @Override
    public void handlerRequest(Post post) {
        //屏蔽广告内容
        String content = post.getContent();
        //.....
        content = content.replace("广告","**");
        post.setContent(content);

        System.out.println("过滤广告...");
        //传递给下一个处理器
        next(post);
    }
}
```

涉黄处理器：

```
/**
 * 涉黄处理器
 */
public class YellowHandler extends PostHandler {

    @Override
    public void handlerRequest(Post post) {
        //屏蔽涉黄内容
        String content = post.getContent();
```

```
        //.....
        content = content.replace("涉黄","**");
        post.setContent(content);

        System.out.println("过滤涉黄内容...");
        //传递给下一个处理器
        next(post);
    }
}
```

敏感词处理器：

```
/**
 * 敏感词处理器
 */
public class SensitiveWordsHandler extends PostHandler {

    @Override
    public void handlerRequest(Post post) {
        //屏蔽敏感词
        String content = post.getContent();
        //.....
        content = content.replace("敏感词","**");
        post.setContent(content);

        System.out.println("过滤敏感词...");
        //传递给下一个处理器
        next(post);
    }
}
```

三个责任链对象的结构基本一致，只有具体的业务处理逻辑不同。上面代码中将所有不健康内容都用“*” 号代替。

调用：

```
//创建责任对象
PostHandler adHandler = new AdHandler();
PostHandler yellowHandler = new YellowHandler();
PostHandler swHandler = new SensitiveWordsHandler();

//形成责任链
yellowHandler.setSuccessor(swHandler);
adHandler.setSuccessor(yellowHandler);

Post post = new Post();
post.setContent("我是正常内容，我是广告，我是涉黄，我是敏感词，我是正常内容");

System.out.println("过滤前的内容为："+post.getContent());

post = adHandler.handlerRequest(post);

System.out.println("过滤后的内容为："+post.getContent());
```

调用结果：

```
过滤前的内容为：我是正常内容，我是广告，我是涉黄，我是敏感词，我是正常内容
过滤广告...
过滤涉黄内容...
过滤敏感词...
过滤后的内容为：我是正常内容，我是**，我是**，我是**，我是正常内容
```

看到这里，相信你已经基本掌握了责任链模式。但问题来了，我直接将过滤不良信息写在一个方法里不行吗？比如：

```
public class PostUtil {

    public void filterContent(Post post){

        String content = post.getContent();

        content = content.replace("广告","**");
        content = content.replace("涉黄","**");
        content = content.replace("敏感词","**");

        post.setContent(content);
    }
}
```

相比之下，这种方式更简单，仅仅几行代码就搞定了。为什么还要用责任链模式呢？

大家还记得**开闭原则**吗？如果后面要增加其他的功能，过滤其他类型的内容，我们还得修改上面的 `filterContent` 方法，违背了开闭原则。如果你是一个框架开发者，你希望别人修改你框架的源码吗？

因此我们需要使用责任链模式，能够在不修改已有代码的情况下扩展新功能。

案例

servlet 中的 Filter

servlet中的过滤器Filter就是典型的责任链模式，假如我们要给每一次http请求都打印一个log，就可以使用filter过滤器来实现：

创建一个filter实现Filter接口：

```
public class LogFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {

    }


    @Override
```

```
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
FilterChain filterChain) throws IOException, ServletException {

        System.out.println("write log");
        filterChain.doFilter(servletRequest,servletResponse);

    }

    @Override
    public void destroy() {

    }
}
```

然后将这个过滤器配置到web.xml中：

```
<filter>
  <filter-name>LogFilter</filter-name>
  <filter-class>com.zhoujun.filter.LogFilter</filter-class>
</filter>

<filter-mapping>
  <filter-name>LogFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

在上面LogFilter类中，我们可以看到servlet的责任链是通过 `Filter` 来实现的，这是一个接口，在doFilter中还用到了 `FilterChain` ，也是一个接口。通过查找源码，发现了 `FilterChain` 的其中一个实现类：

```
public class PassThroughFilterChain implements FilterChain {
    @Nullable
    private Filter filter;
    @Nullable
    private FilterChain nextFilterChain;
    @Nullable
    private Servlet servlet;

    public PassThroughFilterChain(Filter filter, FilterChain nextFilterChain) {
        Assert.notNull(filter, "Filter must not be null");
        Assert.notNull(nextFilterChain, "'FilterChain must not be null");
        this.filter = filter;
        this.nextFilterChain = nextFilterChain;
    }

    public PassThroughFilterChain(Servlet servlet) {
        Assert.notNull(servlet, "Servlet must not be null");
        this.servlet = servlet;
    }


    public void doFilter(ServletRequest request, ServletResponse response) throws
```

```
ServletException, IOException {
        if (this.filter != null) {
            this.filter.doFilter(request, response, this.nextFilterChain);
        } else {
            Assert.state(this.servlet != null, "Neither a Filter not a Servlet set");
            this.servlet.service(request, response);
        }

    }
}
```

请仔细看这个实现类，你会发现其结构和我们之前的 `PostHandler` 示例代码及其相识，该类中的 `private FilterChain nextFilterChain;` 相当于 `PostHandler` 中的后继者 `successor`

将我们自定义的Filter配置到web.xml中的操作就是将该对象添加到责任链上，servlet开发者帮我们完成了 `setSuccessor()` 的操作。

总结

责任链模式将常用于过滤器，拦截器，事件（鼠标键盘事件，冒泡事件等）等场景

优点

1. 请求者和接收者解耦
2. 可以动态的增加或减少责任链上的对象，或者修改顺序

缺点

1. 调用者不知道请求可能被哪些责任链对象处理，不利于排错
2. 用户请求可能被责任链中途拦截，最终未必被真正执行，这点既是优点也是缺点，我们可以利用它做权限控制拦截器