前端控制器模式 服务定位器模式

拦截过滤器模式

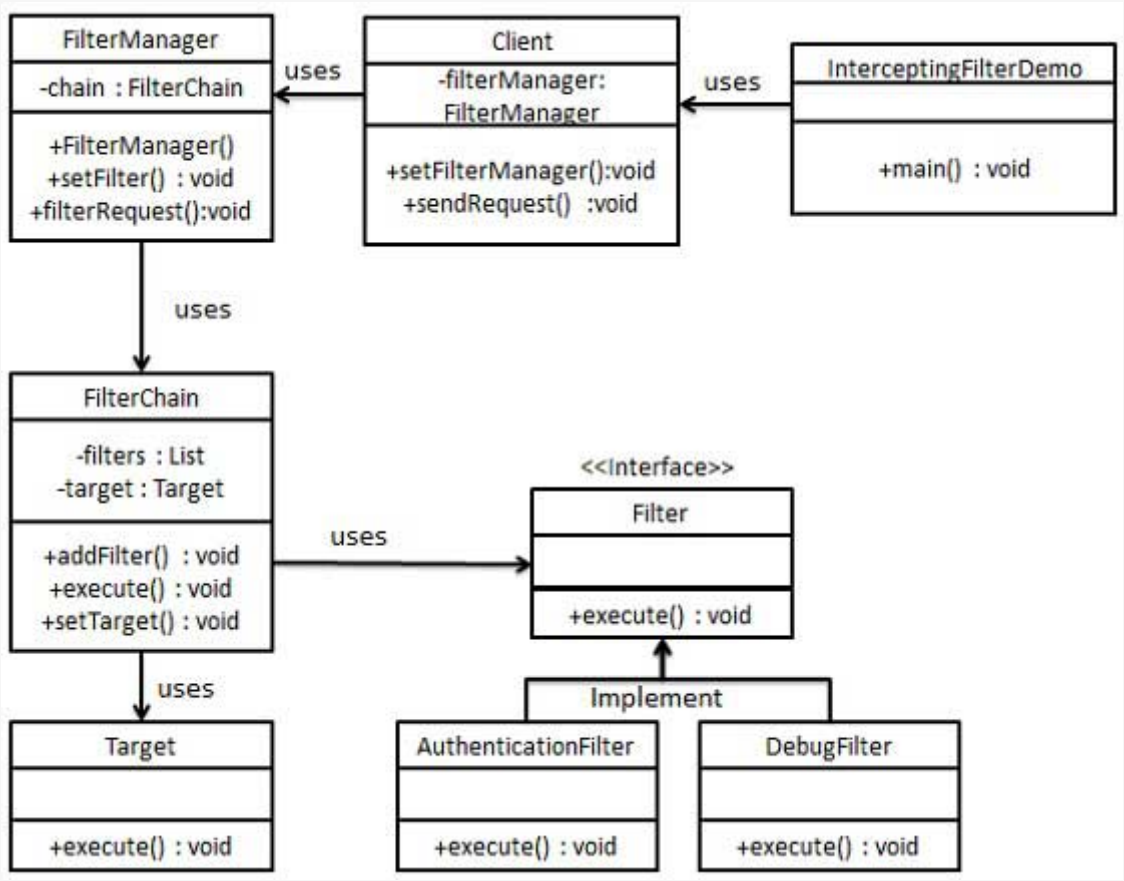
拦截过滤器模式（Intercepting Filter Pattern）用于对应用程序的请求或响应做一些预处理/后处理。定义过滤器，并在把请求传给实际目标应用程序之前应用在请求上。过滤器可以做认证/授权/记录日志，或者跟踪请求，然后把请求传给相应的处理程序。以下是这种设计模式的实体。

- **过滤器（Filter）** - 过滤器在请求处理程序执行请求之前或之后，执行某些任务。
- **过滤器链（Filter Chain）** - 过滤器链带有多个过滤器，并在 Target 上按照定义的顺序执行这些过滤器。
- **Target** - Target 对象是请求处理程序。
- **过滤管理器（Filter Manager）** - 过滤管理器管理过滤器和过滤器链。
- **客户端（Client）** - Client 是向 Target 对象发送请求的对象。

实现

我们将创建 *FilterChain*、*FilterManager*、*Target*、*Client* 作为表示实体的各种对象。*AuthenticationFilter* 和 *DebugFilter* 表示实体过滤器。

InterceptingFilterDemo，我们的演示类使用 *Client* 来演示拦截过滤器设计模式。



步骤 1

创建过滤器接口 Filter。

Filter.java

```
public interface Filter {
public void execute(String request);
}
```

步骤 2

创建实体过滤器。

AuthenticationFilter.java

```
public class AuthenticationFilter implements Filter {
public void execute(String request){
System.out.println("Authenticating request: " + request);
}
}
```

DebugFilter.java

```
public class DebugFilter implements Filter {
public void execute(String request){
System.out.println("request log: " + request);
}
}
```

步骤 3

创建 Target。

Target.java

```
public class Target {
public void execute(String request){
System.out.println("Executing request: " + request);
}
}
```

步骤 4

创建过滤器链。

FilterChain.java

```
import java.util.ArrayList;
import java.util.List;
public class FilterChain {
private List<Filter> filters = new ArrayList<Filter>();
private Target target;
public void addFilter(Filter filter){
filters.add(filter);
}
public void execute(String request){
for (Filter filter : filters) {
filter.execute(request);
```

```
}
target.execute(request);
}
public void setTarget(Target target){
this.target = target;
}
}
```

步骤 5

创建过滤管理器。

FilterManager.java

```
public class FilterManager {
FilterChain filterChain;
public FilterManager(Target target){
filterChain = new FilterChain();
filterChain.setTarget(target);
}
public void setFilter(Filter filter){
filterChain.addFilter(filter);
}
public void filterRequest(String request){
filterChain.execute(request);
}
}
```

步骤 6

创建客户端 Client。

Client.java

```
public class Client {
FilterManager filterManager;
public void setFilterManager(FilterManager filterManager){
this.filterManager = filterManager;
}
public void sendRequest(String request){
filterManager.filterRequest(request);
}
}
```

步骤 7

使用 *Client* 来演示拦截过滤器设计模式。

InterceptingFilterDemo.java

```
public class InterceptingFilterDemo {
public static void main(String[] args) {
FilterManager filterManager = new FilterManager(new Target());
filterManager.setFilter(new AuthenticationFilter());
filterManager.setFilter(new DebugFilter());
```

```
Client client = new Client();
client.setFilterManager(filterManager);
client.sendRequest("HOME");
}
}
```

步骤 8

执行程序，输出结果：

```
Authenticating request: HOME
request log: HOME
Executing request: HOME
```

前端控制器模式 服务定位器模式

点我分享笔记