Spring AOP ，SpringMVC ，这两个应该是国内面试必问题，网上有很多答案，其实背背就可以。但今天笔者带大家一起深入浅出源码，看看他的原理。以期让印象更加深刻，面试的时候游刃有余。

Spring AOP 原理

简单说说 AOP 的设计：

1. 每个 Bean 都会被 JDK 或者 Cglib 代理。取决于是否有接口。
2. 每个 Bean 会有多个“方法拦截器”。注意：拦截器分为两层，外层由 Spring 内核控制流程，内层拦截器是用户设置，也就是 AOP。
3. 当代理方法被调用时，先经过外层拦截器，外层拦截器根据方法的各种信息判断该方法应该执行哪些“内层拦截器”。内层拦截器的设计就是职责连的设计。

> 是不是贼简单。事实上，楼主之前已经写过一个简单的例子，地址：http://thinkinjava.cn/2018/10/使用-Cglib-实现多重代理/

看完之后更简单。

可以将 AOP 分成 2 个部分来扯，哦，不，来分析。。。 第一：代理的创建；第二：代理的调用。

> 注意：我们尽量少贴代码，尽量用文字叙述，因为面试的时候，也是文字叙述，不可能让你把代码翻出来的。。。所以，这里需要保持一定的简洁，想知道细节，看 interface 21 源码，想知道的更细，看 Spring Framework 最新的 master 分支代码。

代码位置：com.interface21.aop 包下。

开始分析（扯）：

1、代理的创建（按步骤）：

1. 首先，需要创建代理工厂，代理工厂需要 3 个重要的信息：拦截器数组，目标对象接口数组，目标对象。
2. 创建代理工厂时，默认会在拦截器数组尾部再增加一个默认拦截器 —— 用于最终的调用目标方法。
3. 当调用 getProxy 方法的时候，会根据接口数量大余 0 条件返回一个代理对象（JDK or Cglib）。

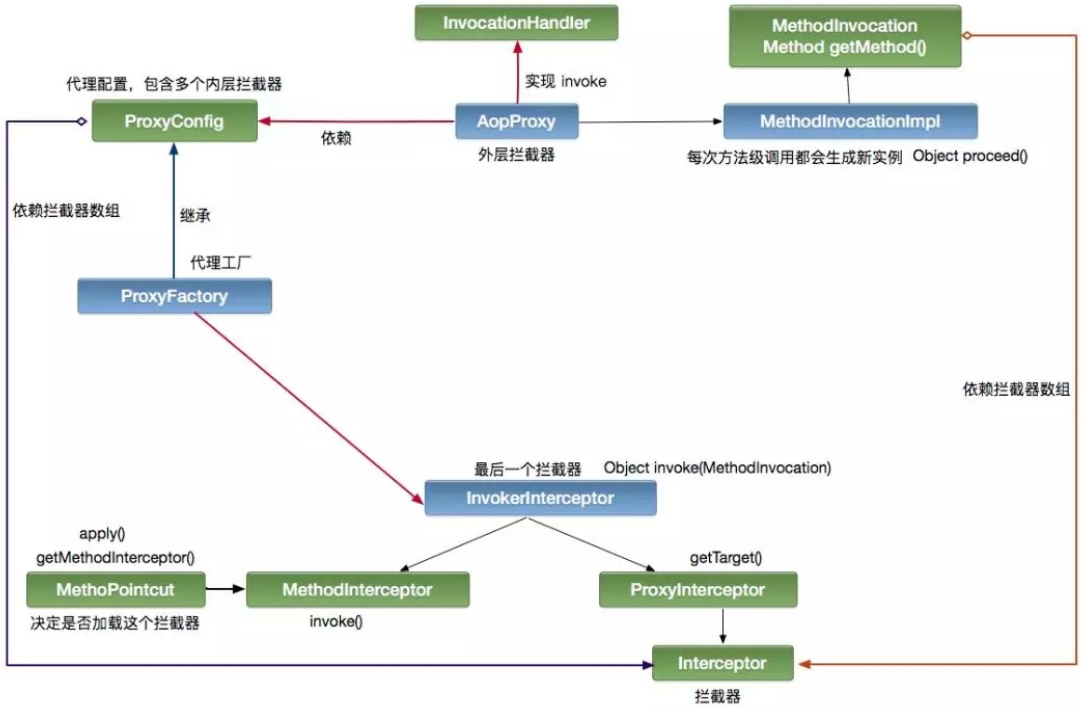
> 注意：创建代理对象时，同时会创建一个外层拦截器，这个拦截器就是 Spring 内核的拦截器。用于控制整个 AOP 的流程。

2、代理的调用

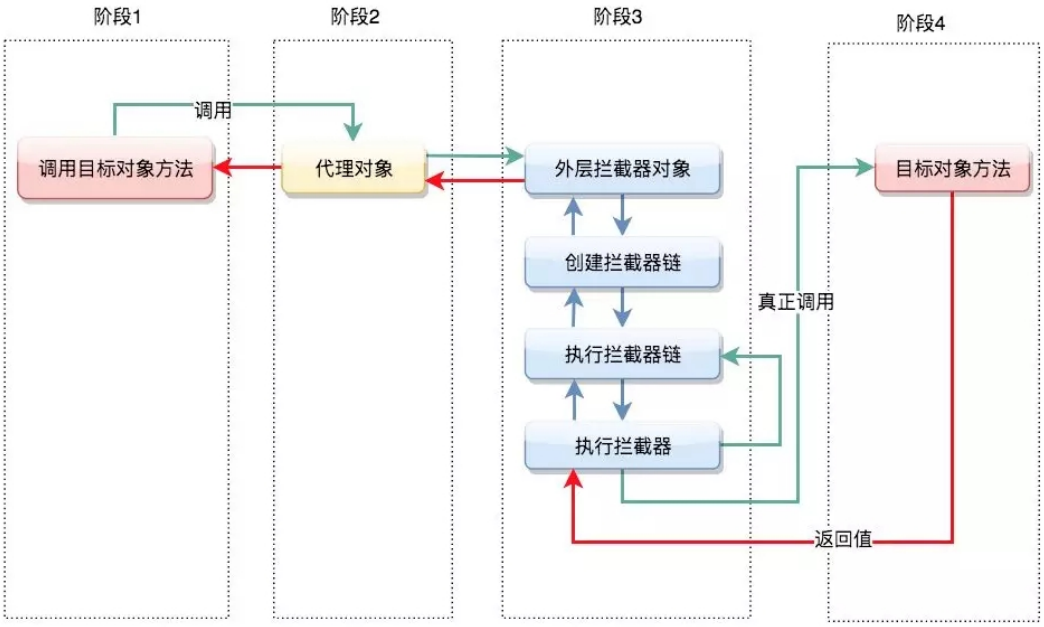
1. 当对代理对象进行调用时，就会触发外层拦截器。
2. 外层拦截器根据代理配置信息，创建内层拦截器链。创建的过程中，会根据表达式判断当前拦截是否匹配这个拦截器。而这个拦截器链设计模式就是职责链模式。
3. 当整个链条执行到最后时，就会触发创建代理时那个尾部的默认拦截器，从而调用目标方法。最后返回。

> 题外话：Spring 的事务也就是个拦截器。

来张不是很标准的 UML 图：



关于调用过程，来张流程图：



大概就是这样子，具体更多的细节，请看源码，如果还不是很明白的话，请咨询本人，本人不确定这个图是否画的很浅显易懂 —— 最起码萌新看得懂才能称之为浅显易懂。

Spring MVC 过程

先来张图：

代码位置：com.interface21.web.servlet.DispatcherServlet#doService

（没错，就是 Spring 1.0 的代码，大道至简，现在的 Spring 经过 15 年的发展，已经太过臃肿，从学习角度来说，interface 21 是最好的代码，不接受反驳）

代码如下：

1.设置属性

```
// 1. 设置属性
// Make web application context available
request.setAttribute(WEB_APPLICATION_CONTEXT_ATTRIBUTE, getWebApplicationContext());

// Make locale resolver available
request.setAttribute(LOCALE_RESOLVER_ATTRIBUTE, this.localeResolver);

// Make theme resolver available
request.setAttribute(THEME_RESOLVER_ATTRIBUTE, this.themeResolver);
```

2.根据 Request 请求的 URL 得到对应的 handler 执行链，其实就是拦截器和 Controller 代理对象。

```
// 2. 找 handler 返回执行链
HandlerExecutionChain mappedHandler = getHandler(request);
```

3.得到 handler 的适配器

```
// This will throw an exception if no adapter is found
// 3. 返回 handler 的适配器
HandlerAdapter ha = getHandlerAdapter(mappedHandler.getHandler());
```

关于这个适配器，作用到底是啥呢？

> HandlerAdapter 注释写到：This interface is not intended for application developers. It is available to handlers who want to develop their own web workflow. 译：此接口不适用于应用程序开发人员。它适用于想要开发自己的 Web 工作流程的处理程序。

也就说说，如果你想要在处理 handler 之前做一些操作的话，可能需要这个，即适配一下这个 handler。例如 Spring 的测试程序做的那样：

```
public ModelAndView handle(HttpServletRequest request, HttpServletResponse response, Obj
ect delegate)
            throws IOException, ServletException {
                    // 你可能需要 doSomething.......
            ((MyHandler) delegate).doSomething(request);
            return null;
        }
```

4.循环执行 handler 的 pre 拦截器

```
// 4. 循环执行 handler 的 pre 拦截器
for (int i = 0; i < mappedHandler.getInterceptors().length; i++) {
    HandlerInterceptor interceptor = mappedHandler.getInterceptors()[i];
    // pre 拦截器
    if (!interceptor.preHandle(request, response, mappedHandler.getHandler())) {
        return;
```

```
    }
}
```

这个没什么好讲的吧？

5.执行真正的 handler，并返回 ModelAndView(Handler 是个代理对象，可能会执行 AOP）

```
// 5. 执行真正的 handler，并返回  ModelAndView(Handler 是个代理对象，可能会执行 AOP )
ModelAndView mv = ha.handle(request, response, mappedHandler.getHandler());
```

6.循环执行 handler 的 post 拦截器

```
// 6. 循环执行 handler 的 post 拦截器
for (int i = mappedHandler.getInterceptors().length - 1; i >=0 ; i--) {
    HandlerInterceptor interceptor = mappedHandler.getInterceptors()[i];
    // post 拦截器
    interceptor.postHandle(request, response, mappedHandler.getHandler());
}
```

7.根据 ModelAndView 信息得到 View 实例

```
View view = null;
if (mv.isReference()) {
    // We need to resolve this view name
    // 7. 根据 ModelAndView 信息得到 View 实例
    view = this.viewResolver.resolveViewName(mv.getViewName(), locale);
}
```

8.渲染 View 返回

```
// 8. 渲染 View 返回
view.render(mv.getModel(), request, response);
```