


```

package org.springframework.web.servlet;

import ...

1 usage 13 implementations
public interface HandlerInterceptor {

    no usages 11 overrides
    default boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        return true;
    }

    no usages 3 overrides
    default void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {
    }

    no usages 3 overrides
    default void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {
    }
}

```

HandlerInterceptor接口方法的作用及其参数、返回值详解如下：

	作用	参数	返回值
preHandle	对拦截到的请求进行预处理，返回true放行，false不放行	Handler是拦截到的Controller方法处理器	一旦返回false，代表终止向后执行，所有后置方法都不执行，最终方法只执行对应preHandle返回了true的
postHandle	在处理器的方法执行后，对拦截到的请求进行后处理，可以在方法中对模型数据和视图进行修改	Handler是拦截到的Controller方法处理器； modelAndView是返回的模型视图对象	无
afterCompletion	视图渲染完成后(整个流程结束之后)，进行最后的处理，如果请求流程中有异常，可以处理异常对象	Handler是拦截到的Controller方法处理器； ex是异常对象	无

- SpringMVC的拦截器
 - 1. 快速使用
 - 2. 多个拦截器的执行顺序
 - 3. 拦截器原理

1. 快速使用

拦截器需要实现HandlerInterceptor接口，并配置在spring-mvc.xml中，定义拦截器的拦截路径，

```

public class MyInterceptor1 implements HandlerInterceptor {
    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        System.out.println("MyInterceptor1 preHandle...");
    }
}

```

```

        return true;
    }

    @Override
    public void postHandle(HttpServletRequest request, HttpServletResponse
response, Object handler, ModelAndView modelAndView) throws Exception {
        System.out.println("MyInterceptor1 postHandle...");
    }

    @Override
    public void afterCompletion(HttpServletRequest request, HttpServletResponse
response, Object handler, Exception ex) throws Exception {
        System.out.println("MyInterceptor1 afterCompletion...");
    }
}

```

```

<!--配置拦截器-->
<mvc:interceptors>
    <mvc:interceptor>
        <!--* 拦截一级路径, ** 拦截多级路径-->
        <mvc:mapping path="/**"/>
        <bean class="com.example.interceptors.MyInterceptor1"/>
    </mvc:interceptor>
</mvc:interceptors>

```

2. 多个拦截器的执行顺序

再定义一个HandlerInterceptor, 命名为MyInterceptor2, spring-mvc.xml配置如下,

```

<!--配置拦截器-->
<mvc:interceptors>
    <mvc:interceptor>
        <!--* 拦截一级路径, ** 拦截多级路径-->
        <mvc:mapping path="/**"/>
        <bean class="com.example.interceptors.MyInterceptor1"/>
    </mvc:interceptor>
    <mvc:interceptor>
        <mvc:mapping path="/**"/>
        <bean class="com.example.interceptors.MyInterceptor2"/>
    </mvc:interceptor>
</mvc:interceptors>

```

执行后, 输出如下,

```

MyInterceptor1 preHandle...
MyInterceptor2 preHandle...
req2...

```

```
MyInterceptor2 postHandle...
MyInterceptor1 postHandle...
MyInterceptor2 afterCompletion...
MyInterceptor1 afterCompletion...
```

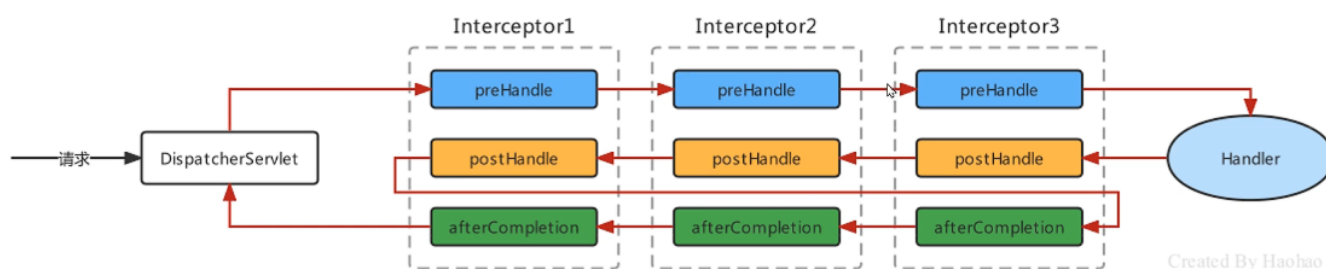
替换两个拦截器的配置顺序，输出如下，

```
MyInterceptor2 preHandle...
MyInterceptor1 preHandle...
req2...
MyInterceptor1 postHandle...
MyInterceptor2 postHandle...
MyInterceptor1 afterCompletion...
MyInterceptor2 afterCompletion...
```

由此可见，**拦截器的执行顺序取决于配置的顺序**。具体方法的执行顺序，有下图，

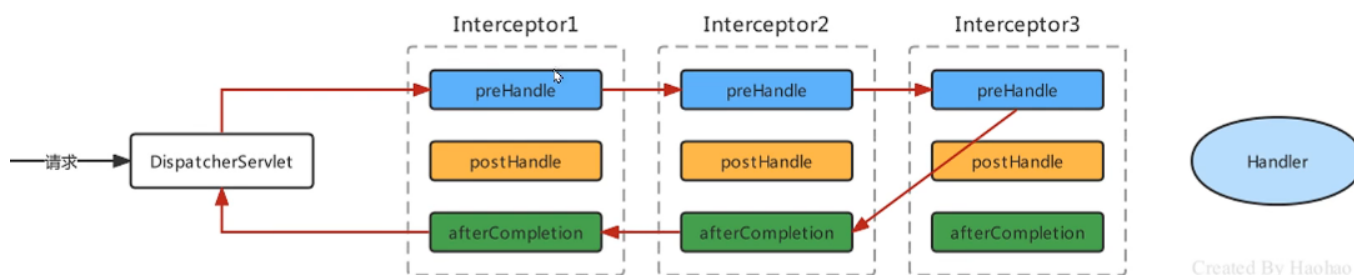
拦截器三个方法的执行顺序

当每个拦截器都是放行状态时，三个方法的执行顺序如下：



拦截器三个方法的执行顺序

当Interceptor1和Interceptor2处于放行，Interceptor3处于不放行时，三个方法的执行顺序如下：



注意，afterCompletion是否执行，取决于它的preHandle是否返回true。

3. 拦截器原理

