Wangle阅读借鉴：

1. 各种回调通知
2. 类借口齐全，调用方便
3. 可读性高
4. 效率高
5. 模板使用技巧
6. 内部类
7. 工厂模式运用
8. 注重类型安全

二、剖析

1、 各种回调通知

结果（成功，失败，异常）采用回调通知

2、 类提供的借口齐全，方便供外部调用

类提供了多种调用接口的方式，比如Pipeline::addBack(…)中，提供了Pipeline::addBack(H\*), Pipeline::addBack(H&&), Pipeline::addBack(std::shared\_ptr, bool isFront)内部转化为第三种方式调用，所以操作上也比较方便。类似的还有NotificationQueue::putMessage几个接口的集中形式

3、模板

在Pipeline里面用了一套模板，Handler,HandlerContext

Pipeline里面的内部存储的元素设计为PipelineContext\*指针类型（实际为ContextImpl,InboundContextImpl,OutboundContextImpl三种类型）

Front\_和back\_设置为InboundLink\*和OutboundLink\*类型

Pipeline设计的原则应该是处理事件的Handler可以是任意类型对象

但是其内部存储的对象必须支持setNextIn和setNextOut两中操作，但是Handler应该是专门处理事件的对象不应该具有这两种操作。

Pipeline的add方法提供对外的参数为Handler，但实际上Pipeline内部存储的是ContextImpl\*对象，因为实际Pipeline里面存储的对象应该是既能够处理事件也能够转发消息请求的，这里将转发和实际处理的功能分离为HandlerContext和Handler，同样HandlerContext是专门用于转发的对象也不应该具有上述两种操作，所以必须抽象出一个PipelineContext类出来具有setNextIn和setNextOut两种功能用与Pipeline内部的存储对象类型。但是当Pipeline处理请求的时候入口不应该是PipelineContext开始处理（没有这种类型的方法）

所以抽象抽象出了InboundLink和OutboundLink两中抽象类型。

这样实际事件处理时就可以由继承\*boundLink处理。

综合上述两个原因实际存放在Pipeline内的元素应该是同时继承了PipelineContext和\*boundLink的类对象，同时还要具有转发的功能那么应该还需要继承HandlerContext（同时在这个例子中可以看出多重继承可以将多个类的功能聚集到一个类上面），这就是ContextImpl\*的由来

对于Handler和HandlerContext的设计

Handler有三种类型：处理读写请求，处理读请求，处理写请求的。同时这三种类型不应继承与同一个基类，因为三种类型间无可继承关系。另外对于处理的请求类型应该是模板类型的，因为其可以处理任意类型的请求。Handler<Rin, Rout, Win,Wout>, InboundHandler<Rin, Rout>, OutboundHandler<Win,Wout>

同理三种HandlerContext也不应该有继承关系；另外对于其转发类型定义，因为由Handler处理完的消息类型已经和原来不一样了，所以其转发请求的类型也应该是模板类型HandlerContext<Rin, Win >, InboundHandlerContext<Rin>, OutboundHandlerContext< Wout>

讨论清楚这些问题之后可以开始讨论关于ContextImpl\*的设计方法

ContextImpl\*同时继承与\*boundLink,PipelineContext,\*HandlerContext，同时ContextImpl\*内部应该有Handler变量用于由其实际处理问题。由于ContextImpl继承与一个ContextImplBase用于实现setNext\*(In,Out)，这个函数中同样设计到Handler和HandlerContext，而其又没有继承关系，所以用模板即可。