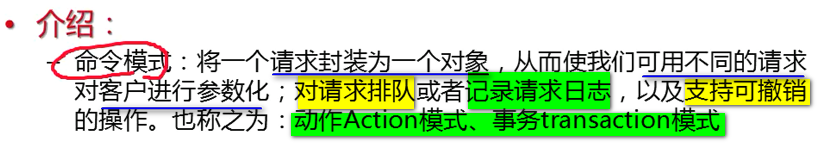
命令模式

实际开发的时候 用到的不多

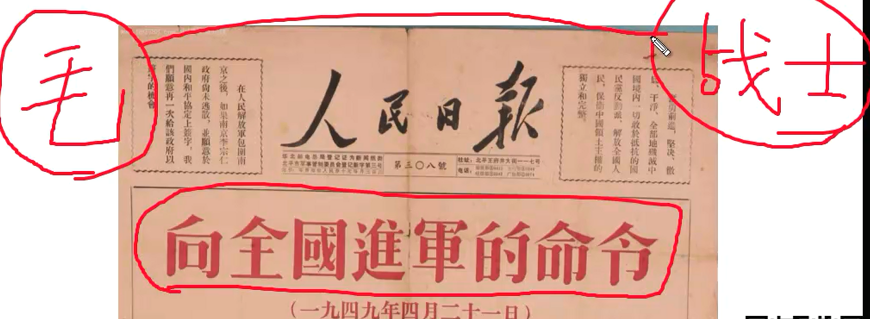


命令模式 ---- 动作模式 ----- 事务模式

毛主席发出命令 最终执行命令的是战士 ---- 可以直接和战士直接交流

比如战士很少 并且命令非常简单 这个没有问题

但是 如果战士非常多 并且 命令非常复杂 怎么进攻 什么进攻战术 这个直接交流就非常麻烦了



在毛主席和战士之间建立命令对象 然后发给战士

发一个请求 以前直接发给他 没有中间处理的过程 到那时 现在有了多个 可以在中间缓存 优化好 保存日志之后 再发给他

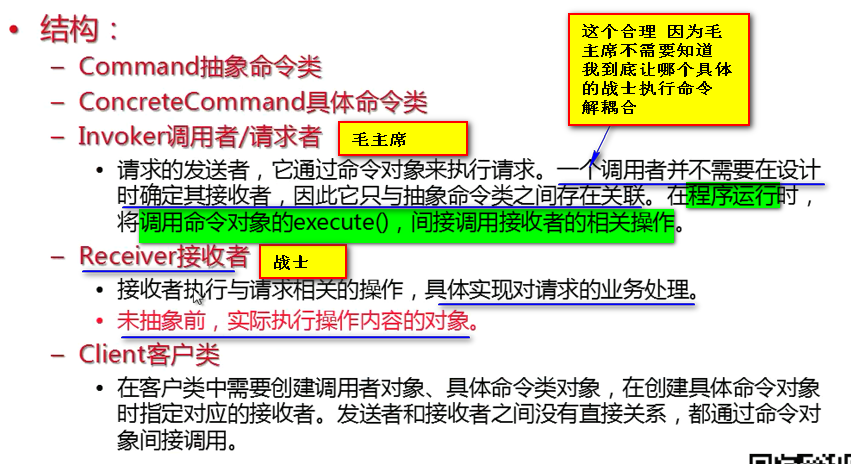
也就是还是分工

对请求动作命令动作 进行缓存批处理

事务模式：数据库中 有一个事务 一批要么执行 要么同时不执行 的SQL

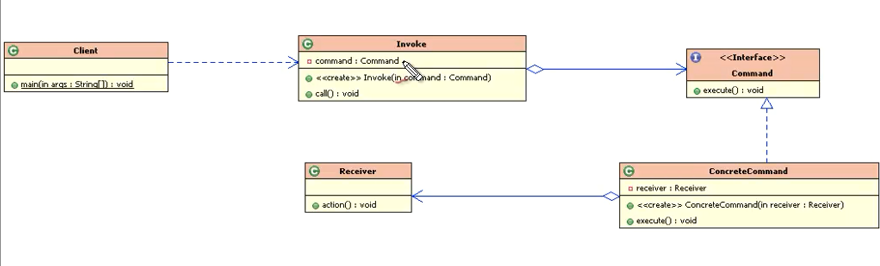
这样 连续发了三条SQL --- 这样在中间缓存一批SQL命令 ----- 通过备忘录模式 把操作之前的状态保存起来 记录好 然后发出命令给你操作 这一批命令全发出去

所以 事务的底层就是命令模式

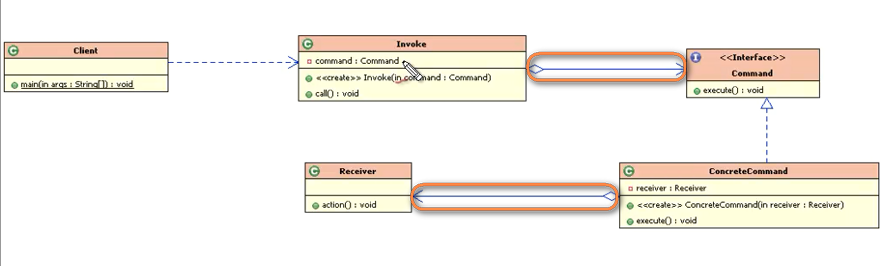


Command出现把Invoker和Receiver进行了解耦合 【有点像门面模式】

Command用来封装我们的命令



Invoker可以定义一个容器包含很多命令对象



【如果画成一排 就是从Invoker到ConcreteCommand到Receiver 逐个持有下一个对象

Invoker持有Command Command持有Receiver】

这个Command有一个execute方法

Invoker找到command command找到receiver ---- receiver可以批处理 是一个容器

【老师说 命令可以是容器 发多个命令 接受者也是多个 容器处理】



Command接口

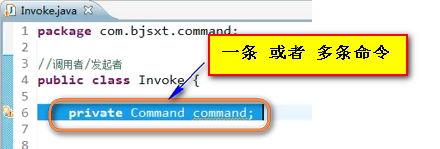


现在实现一个具体的ConcreteCommand

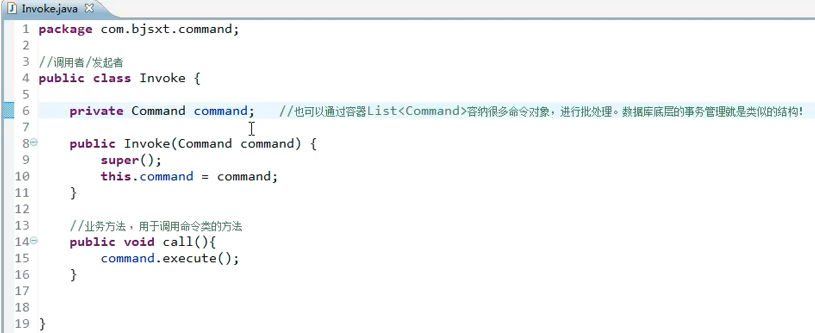


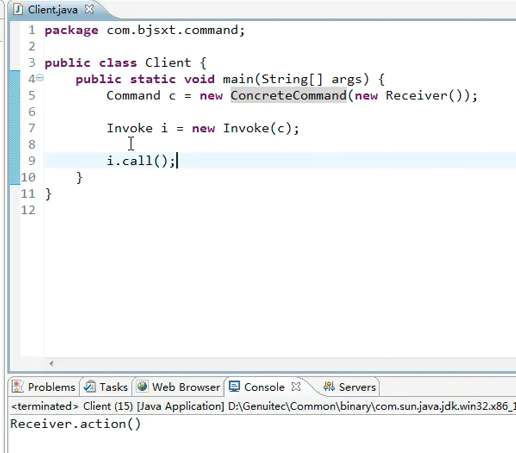
调用前后可以做一些处理

还有就是invoker



多条命令 放到一个容器中 进行批处理 数据库里面的事务管理 就是类似的设计





这样 Command对Invoker和Receiver是解耦合

如果对命令做复杂的处理 这样就可扩展性更强

