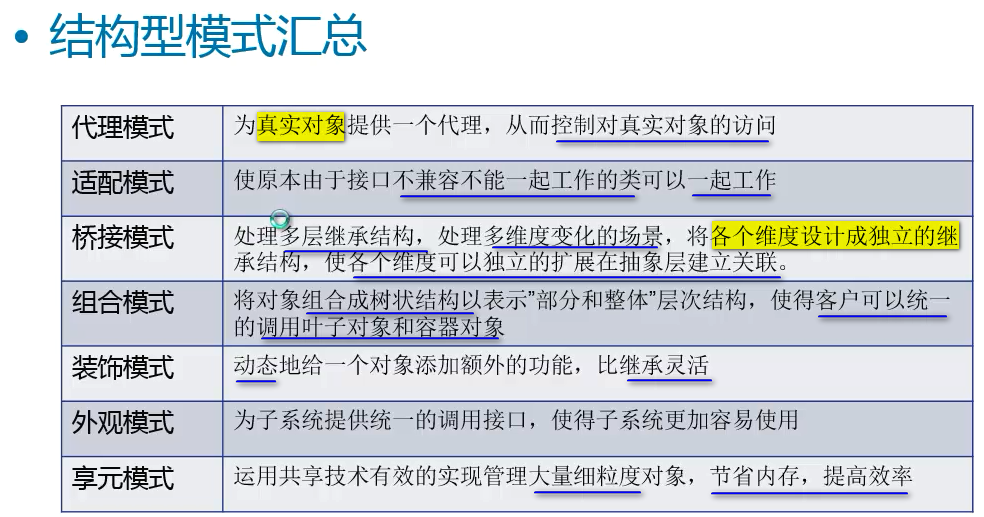
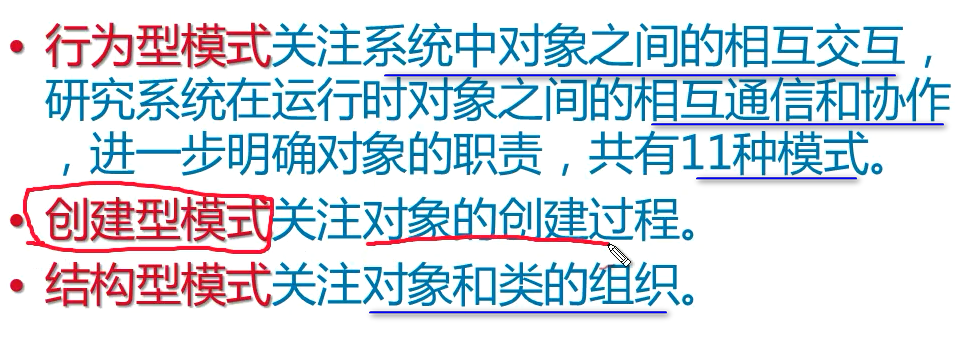
之前的课 结构型的模式都讲完了



这样 之后的模式 就是行为类型的模式



怎样协作 怎样通信 23中占了一大半



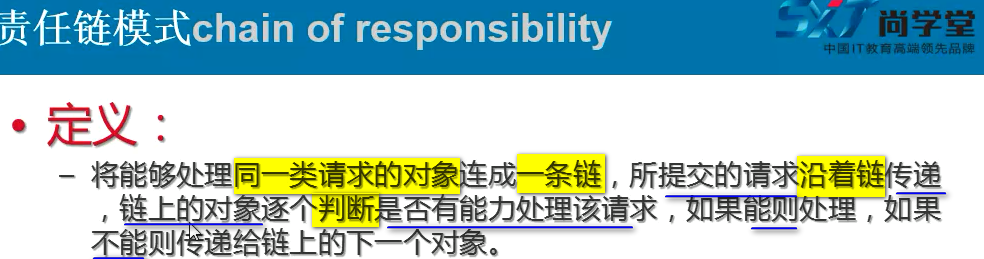
有三个不太常用

解释器模式 ---- 除非用Java构建脚本语言 否则不用

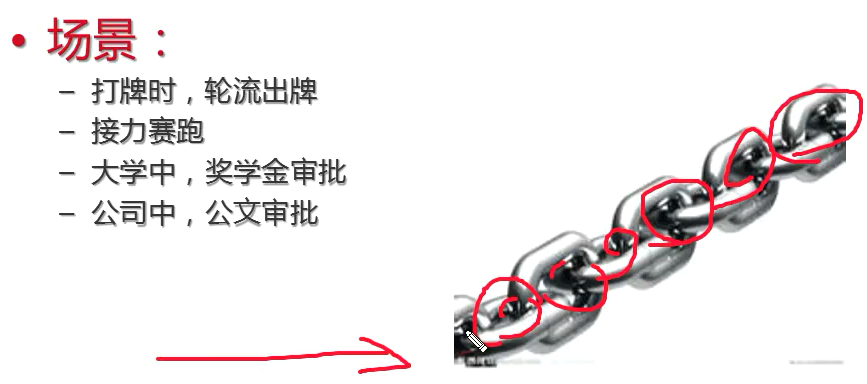
命令模式 --- 底层软件用的多

访问者模式 ---- 用的比较少

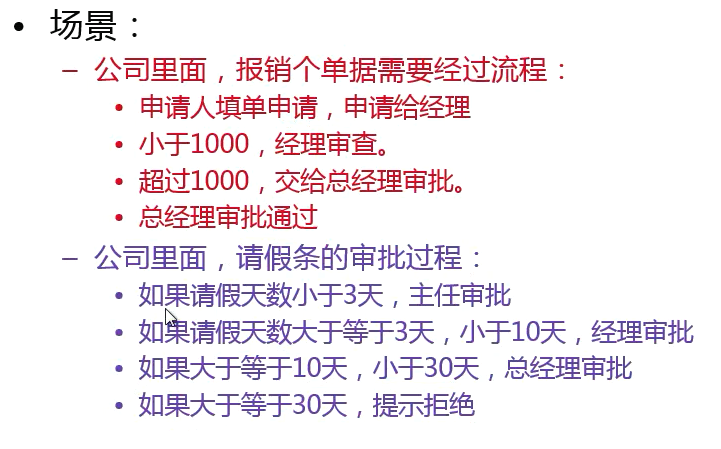
责任链模式 Responsibility Chain Pattern



处理请求的对象组成链 ---- 请求在链上传递 ---- 逐个判断 能处理 处理 不能处理 传递给下一个处理对象



工作中用的很多 审批 接力赛



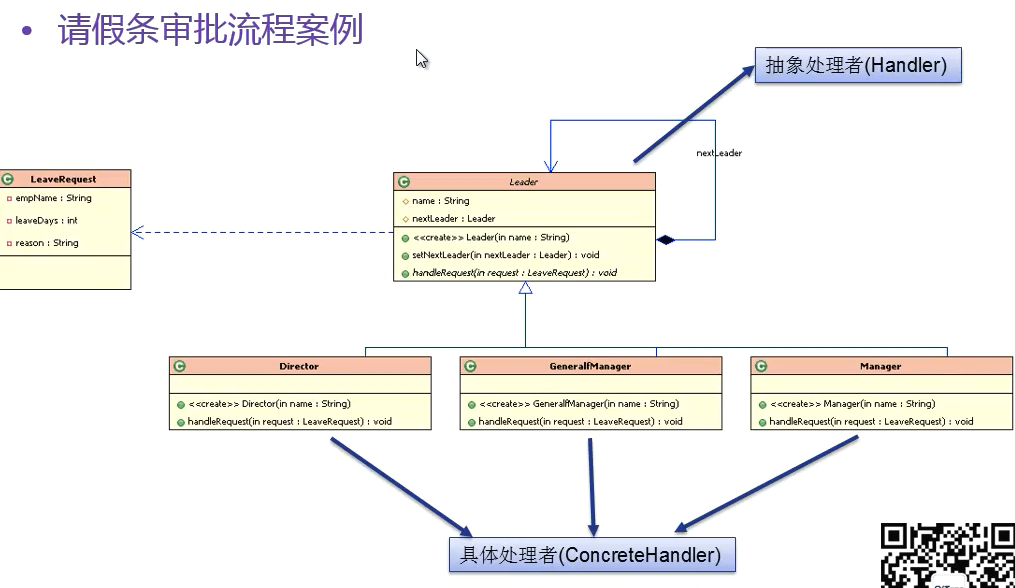
我写一些if else语句不就OK了么

问题在于通过if else语句来做 如果后续流程做改变 比如总经理审批 加一个副经理审批 这样就频繁修改源代码 ---违反了开闭原则

流程复杂的话 整个代码会非常复杂 ---- 不利于后期的维护

需要责任链代替这样的过程





请假条的对象 LeaveRequest 封装请假条的信息

然后 定一个leader 抽象类 【这个菱形的箭头 直接作用在父类上面了 这样子类一继承 就有了下一任 抽象到父类 因为所有的子类都有这个特点 但是装饰模式的类图就不是这样的了 有的是真实的子类 有的是修饰类 所以 这个箭头在装饰器模式中 只能指向具有装饰功能的Decorator 不能指向父类 --- -因为 不是所有的子类都具有这个 ---- **也就是每一个节点 内部一定嵌套下一个节点** 特点】

【handleRequest用来处理当前节点的请求 那么setNextLeader就是用来进行传递 ---- 形成链条 ----- 内部有一个同类型的引用 就是set的结果 形成链条】

 ---- 这个聚合表示下一个

这里有一个抽象的方法handleRequest 是用来处理请假的核心子类都要实现它

-------

COR chain of responsibility

建立一个假条



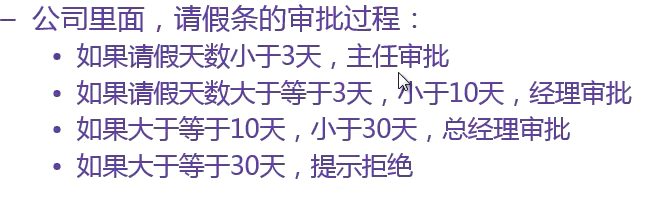
名字给子类 所以protected

抽象类的父类决定不了如何处理



【我们的root其实很多都是流程 我觉得责任链也不错 这样大体上面 使用行为类设计模式 动态组装 有点和装饰器类似 但是 实际上 如果是责任链 需要高并发检测每一个handler】

接下来定义我们的子类





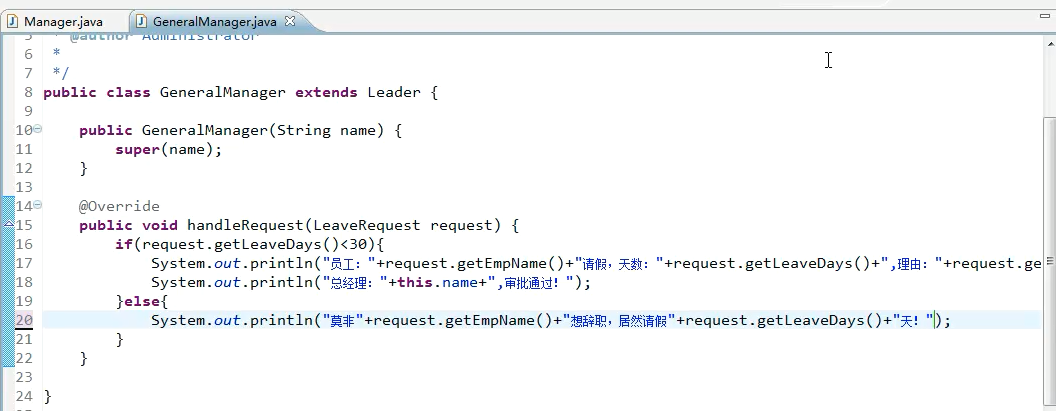
【request可以获取数据 类似于状态模式中的context 参考状态模式中的投票例子】

【职责链的好处就是 我当前的节点 仅仅需要关心好我的handleRequest就好 然后 】

然后是经理

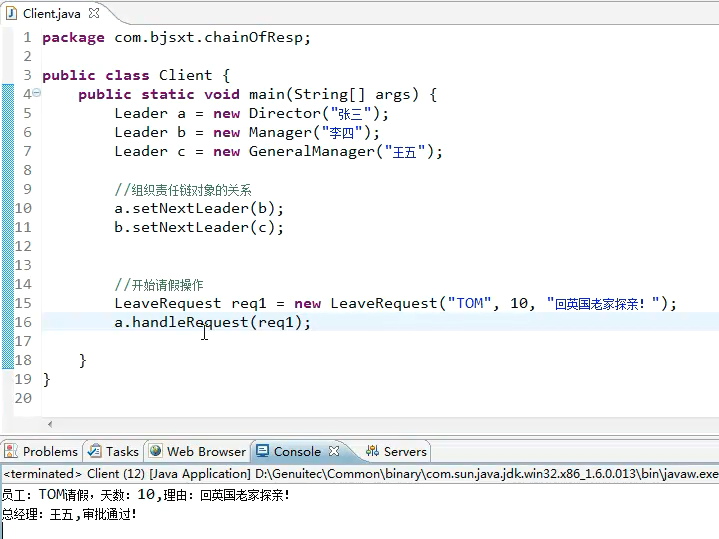


最后一个是总经理



这就是责任链上的类

定义测试



【所以 责任链 就是 内部set一下下一个 仅仅要关系自己handleRequest就好 如果能处理 就处理 如果不能处理 就交给自己持有的下一任即可 ------ 外部需要做的事情 就是 初始化好责任链的每一个节点 然后 设置好每一个节点的下一任 也就是流程安排好 ------ 最后找到起始的节点 调用handleRequest即可启动责任链】

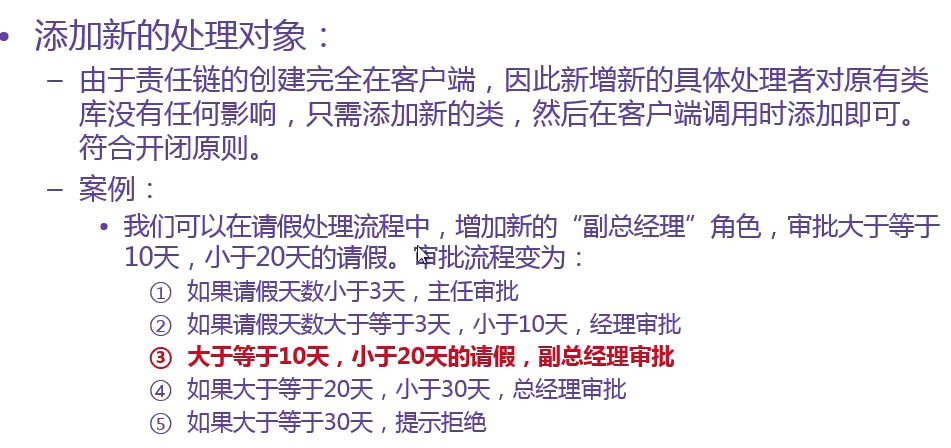


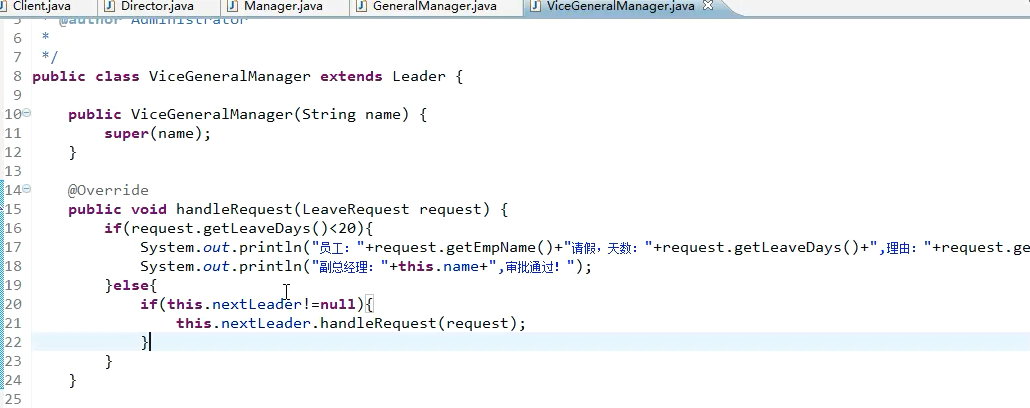
责任链的好处 都在Client定义 后期可以写到配置文件 通过反射进行处理

责任链的关系可以动态去变化

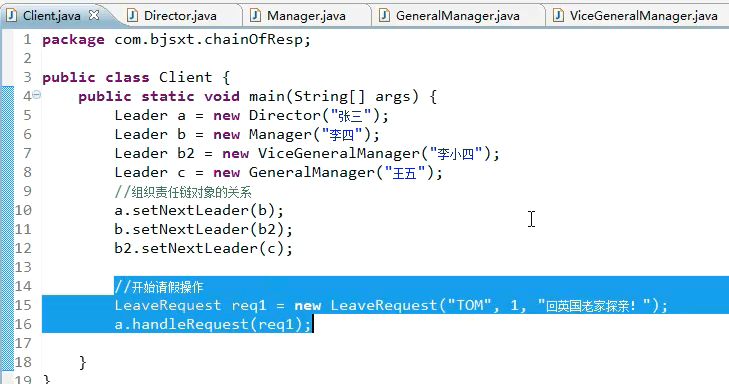
责任链上的对象 可以实时添加

加了一个副总 就是 10-20天 副总审批 ---- 添加新的类 满足开闭原则

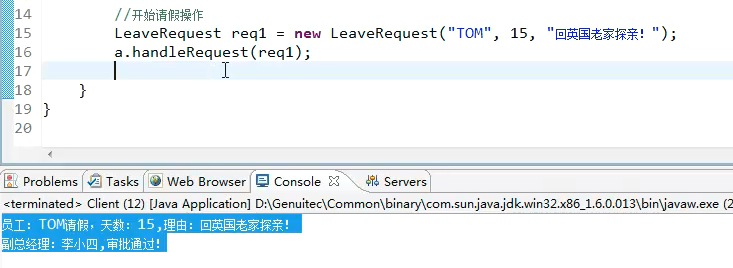




Client中重新修改



处理的代码 都没有变化

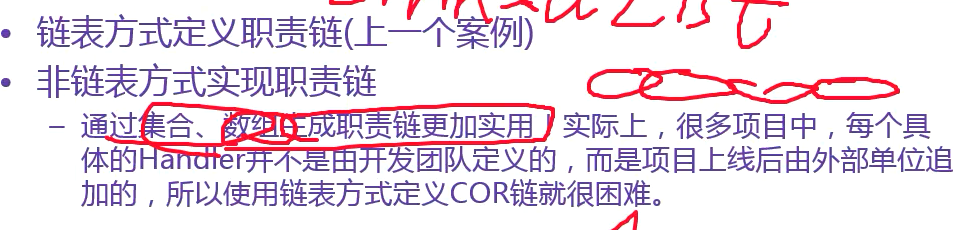


责任链模式很好的逻辑 非常好

链 就是一个容器 ----- LinkedList

和 cor类似的

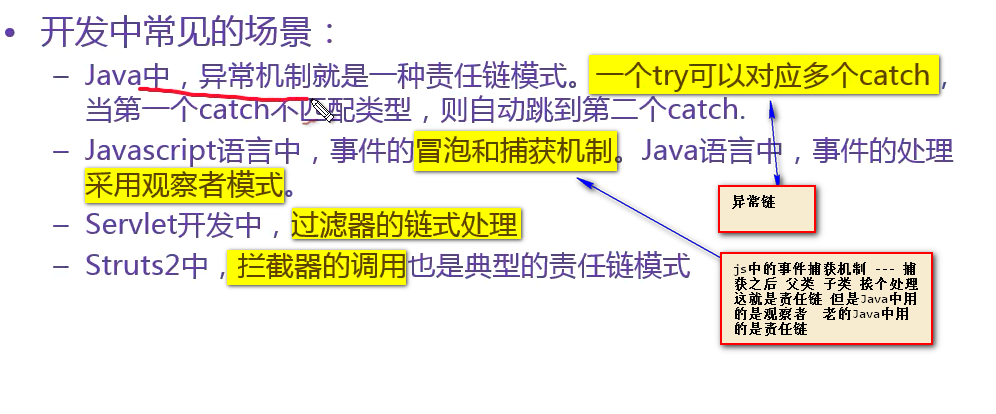
主任对象拉一个经理 ----- 本质上是单向链表



责任链的对象通过容器存储起来 然后在容器中迭代遍历

这里面就不做测试

一般意义 使用责任链 这样就可以了



Spring mvc中的连接器这种也是典型的责任链模式