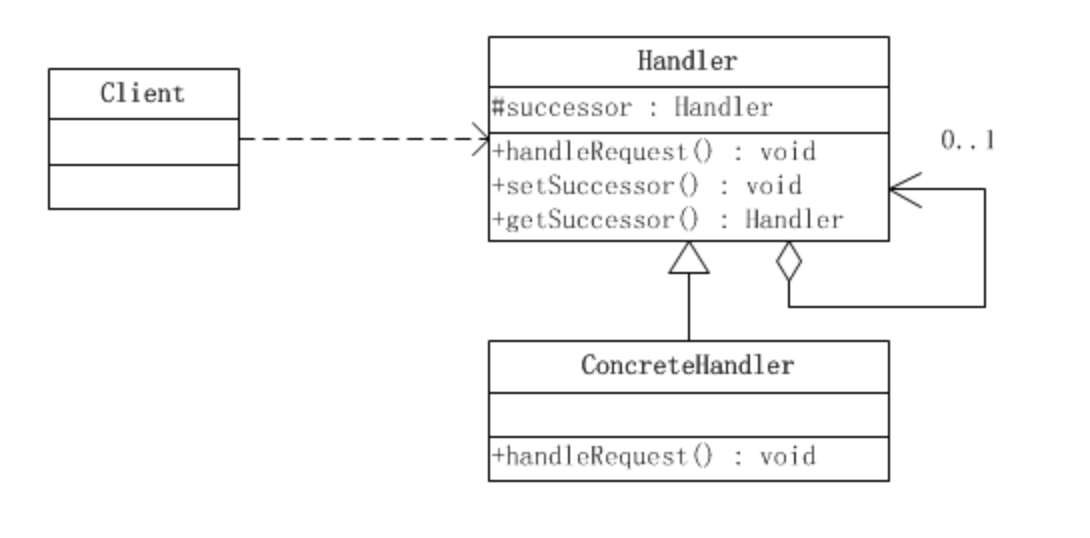
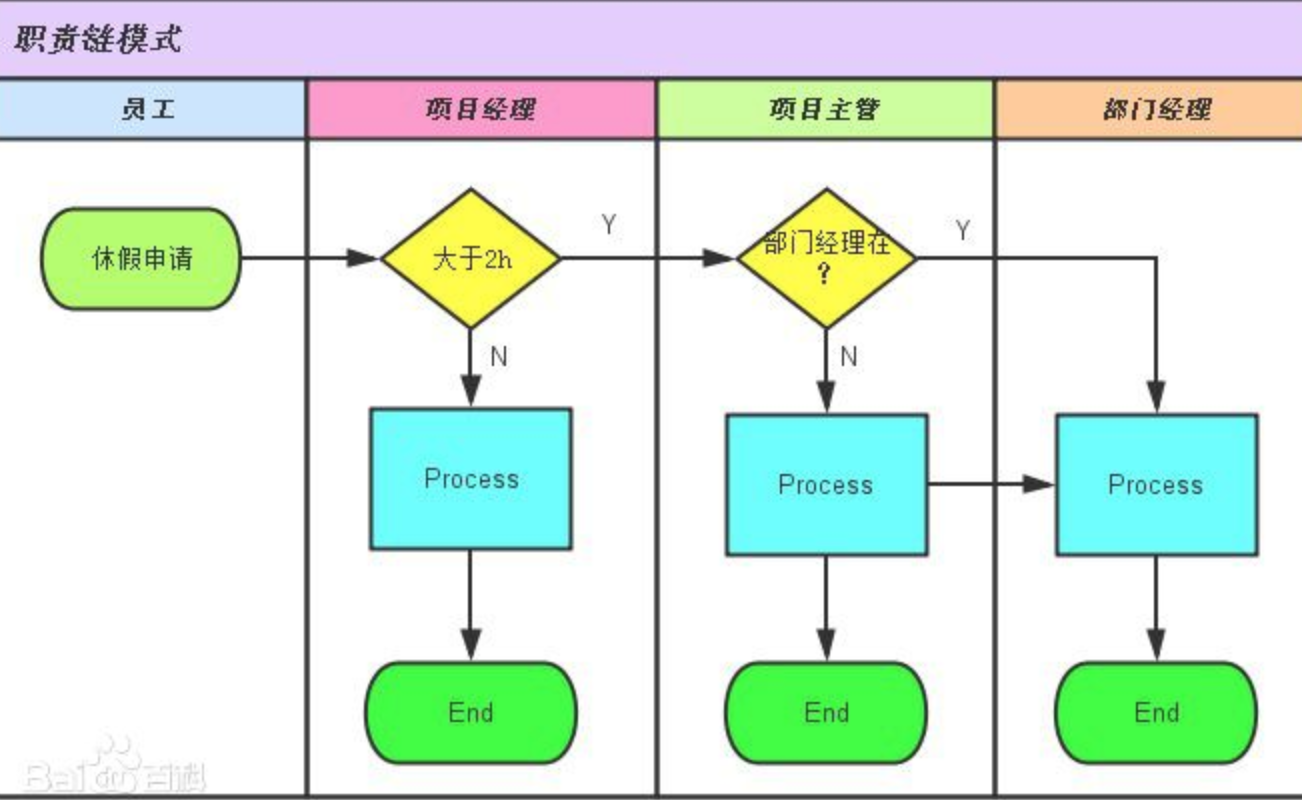
1. 责任链模式：将能够处理同一类请求的对象连成一条链, 所提交的请求沿着链传递, 链上的对象逐个判断是否有能力处理改请求, 如果能则处理, 如果不能则传递给链上的下一个对象
2. 职责链模式通常每个接收者都**包含对另一个接收者的引用**。如果一个对象不能处理该请求，那么它会把相同的请求传给下一个接收者，依此类推
3. 类图



* 1. 抽象处理者(Handler)角色：定义出一个处理请求的接口。如果需要，接口可以定义 出一个方法以set和get对后继者的引用。这个角色通常由一个Java抽象类或者Java接口实现。上图中Handler类的**聚合关系给出了具体子类对后继者的引用**，抽象方法handleRequest()规范了子类处理请求的操作。
  2. 具体处理者(ConcreteHandler)角色: 具体处理者接到请求后，可以选择将请求处理掉，或者将请求传给后继者，从而形成一个责任链

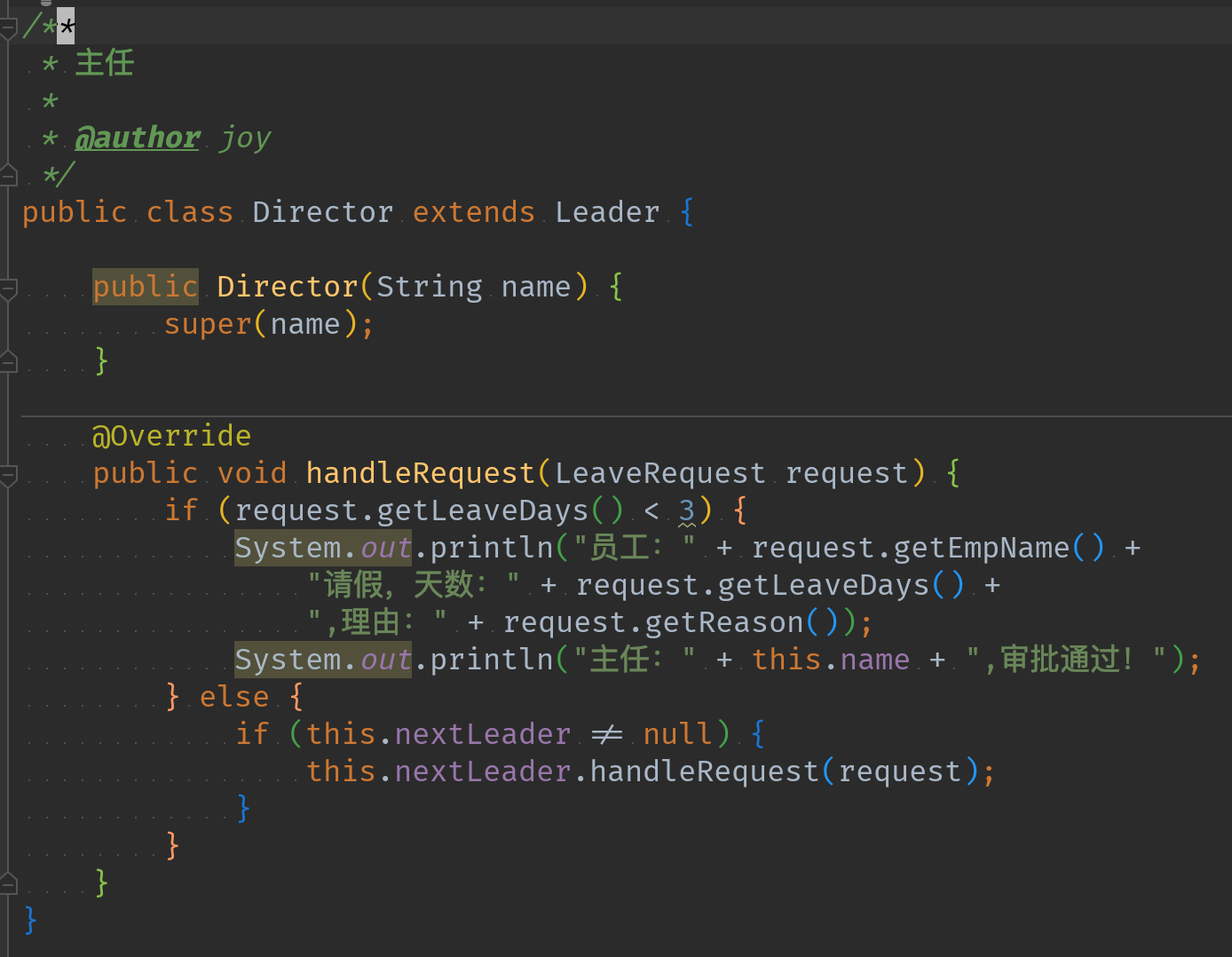
1. 责任链模式应用场合(场景)
   1. 请假审批流程
   2. 奖学金审批
   3. 接力赛跑
2. 简单例子—请假

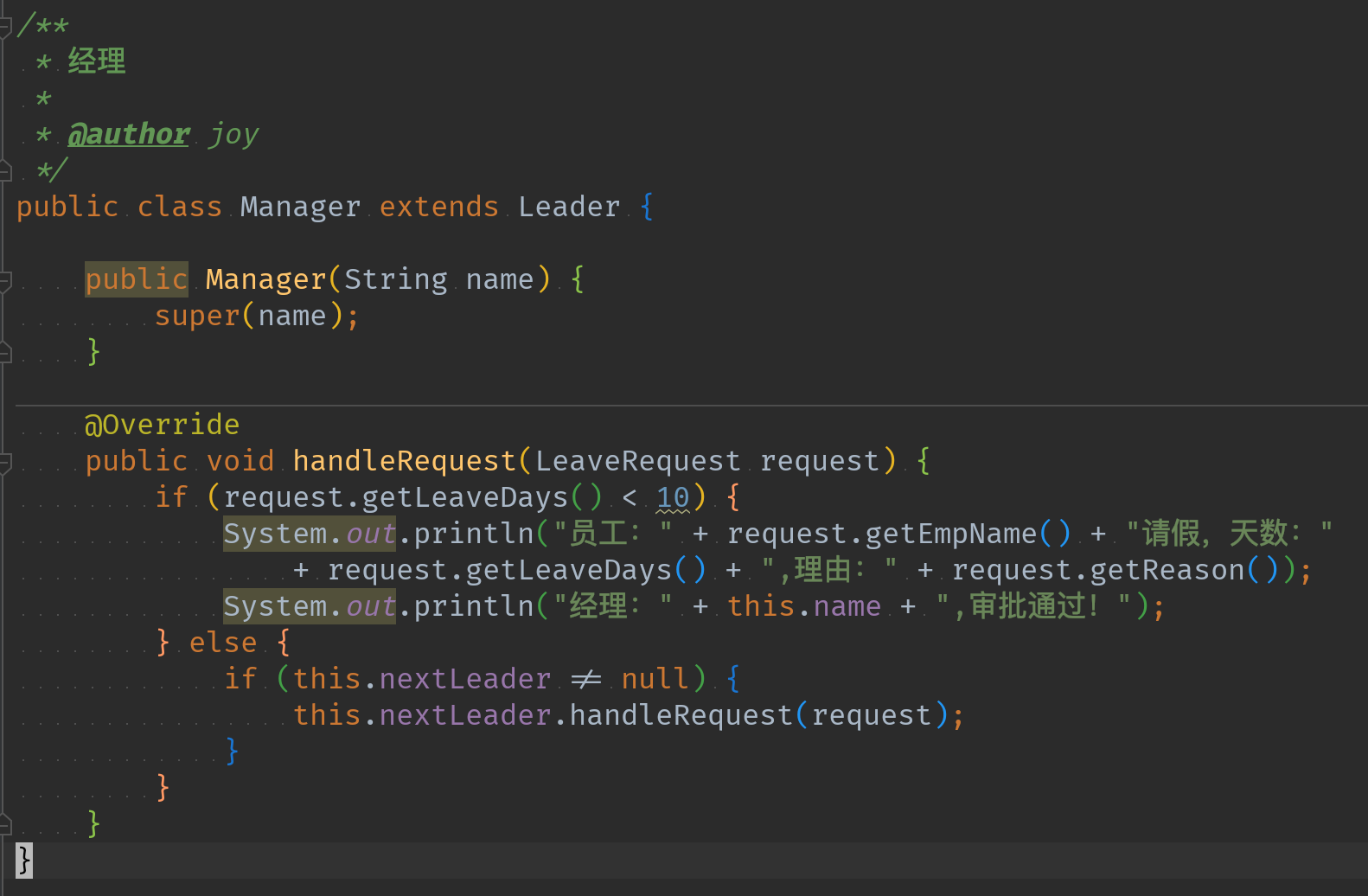


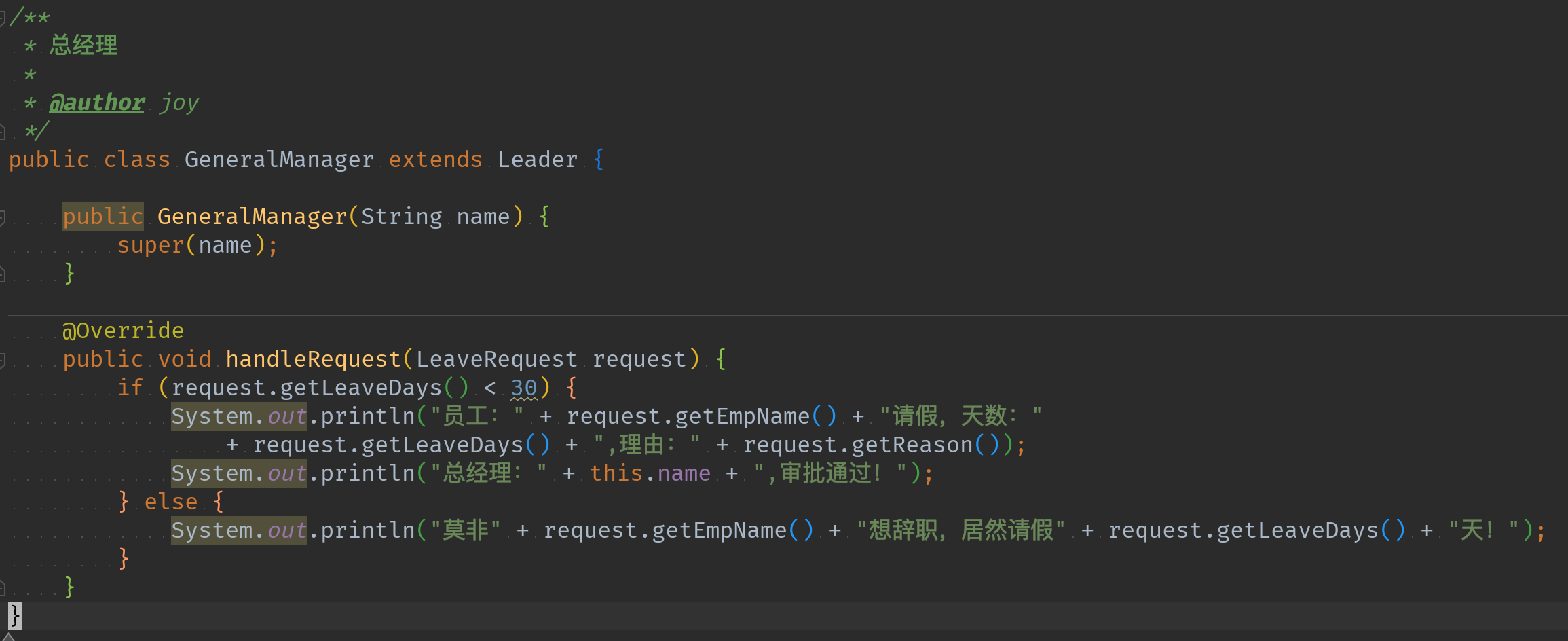
* 1. 抽象处理者



* 1. 具体处理者







* 1. Client



如果可以的话，责任链可以形成『环状』结构，这样请求可以从任一级开始

|  |  |
| --- | --- |
| |  | | --- | | a.setNextLeader(b);  b.setNextLeader(c);  **c.setNextLeader(a);** | |

1. 开发中常见场景
   1. Java异常处理机制, 一个try对应多个catch
   2. Servlet开发中过滤器的链式调用
2. 职责链模式的主要优点
   1. 对象仅需知道该请求会被处理即可，且链中的对象不需要知道链的结构，由客户端负责链的创建，降低了系统的耦合度
   2. 请求处理对象仅需维持一个指向其后继者的引用，而不需要维持它对所有的候选处理者的引用，可简化对象的相互连接
   3. 在给对象分派职责时，职责链可以给我们更多的灵活性，可以在运行时对该链进行动态的增删改，改变处理一个请求的职责
   4. 新增一个新的具体请求处理者时无须修改原有代码，只需要在客户端重新建链即可，符合 "开闭原则"
3. 职责链模式的主要缺点
   1. 一个请求可能因职责链没有被正确配置而得不到处理
   2. 对于比较长的职责链，请求的处理可能涉及到多个处理对象，系统性能将受到一定影响，且不方便调试
   3. 可能因为职责链创建不当，造成循环调用，导致系统陷入死循环
4. 适用场景
   1. 有多个对象可以处理同一个请求，具体哪个对象处理该请求待运行时刻再确定，客户端只需将请求提交到链上，而无须关心请求的处理对象是谁以及它是如何处理的
   2. 在不明确指定接收者的情况下，向多个对象中的一个提交一个请求
   3. 可动态指定一组对象处理请求，客户端可以动态创建职责链来处理请求，还可以改变链中处理者之间的先后次序