

几分钟说在前面的话

Design Patterns

马士兵

Chain Of Responsibility

责任链

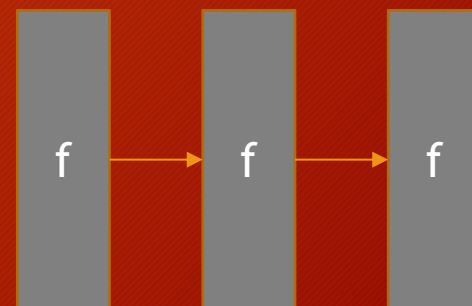
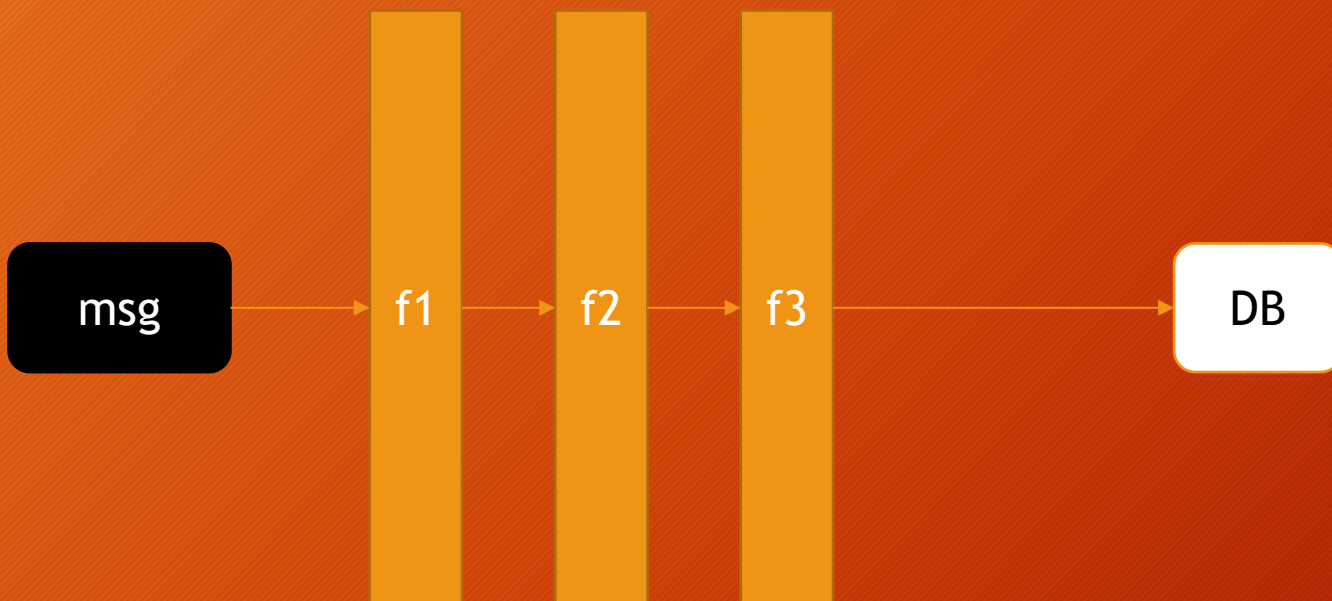
问题:

- 在论坛中发表文章
- 后台要经过信息处理才可以发表或者进入数据库

做法

- 最直观的：定义**MsgHandler**直接处理
- 想扩展性好一些：定义不同的**Filter**
- 继续：将**Filter**们构成链条
 - 小技巧：链式编程
- **FilterChain**其实自己也是一个**Filter**
- 由**FilterChain**中的某一个**Filter**决定链条是否继续

Chain Of Responsibility



应用场景

- command
 - do
 - undo
- servlet Filter
 - doFilter(req,res,chain)
- listener
- tank collider

Singleton

单例

应用场景

- 只需要一个实例
 - 比如各种Mgr
 - 比如各种Factory

Strategy

策略

Comparable

Comparator

一步一步写策略

- int[]
- double ?

工厂系列

简单工厂

静态工厂

工厂方法

抽象工厂

Spring IOC

定义

- 任何可以产生对象的方法或类，都可以称之为工厂
- 单例也是一种工厂
- 不可咬文嚼字，死扣概念
- 为什么有了**new**之后，还要有工厂？
 - 灵活控制生产过程
 - 权限、修饰、日志...

举例

- 任意定制交通工具
 - 继承Moveable
- 任意定制生产过程
 - Moveable XXXFactory.create()
- 任意定制产品一族

思考：

- Factory Method
- vs
- Abstract Method

更好的解决方案

- spring ioc
- bean工厂

?

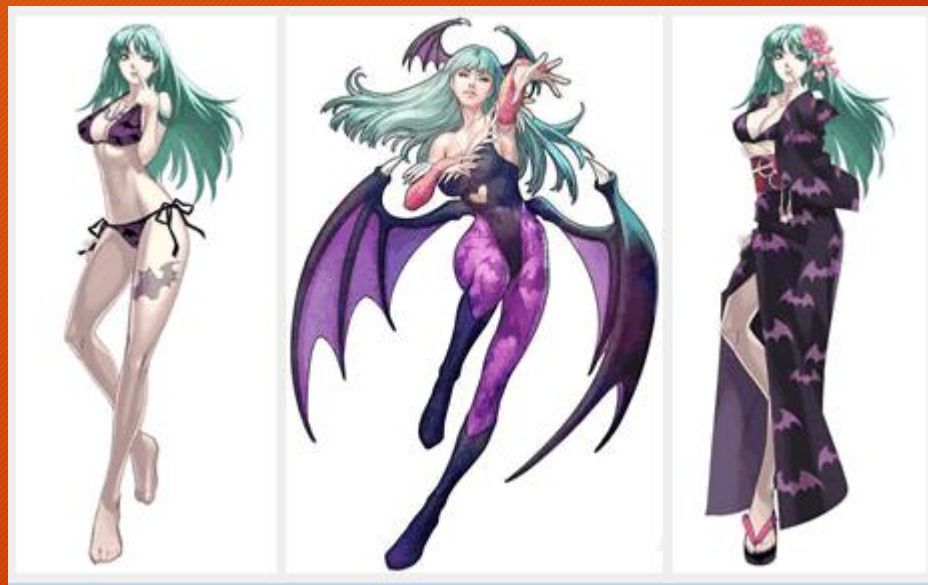
- 抽象类 vs 接口
 - 概念真实存在-抽象类
 - 强调属性-接口
 - 从语义来理解，语法上都可以

装饰

decorator

IO ?

装饰之间的混搭

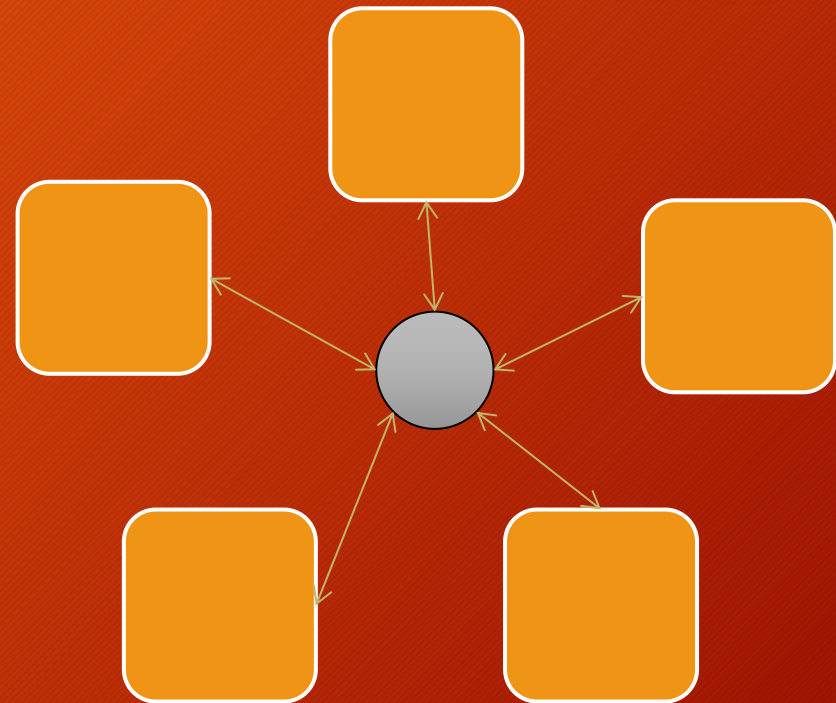
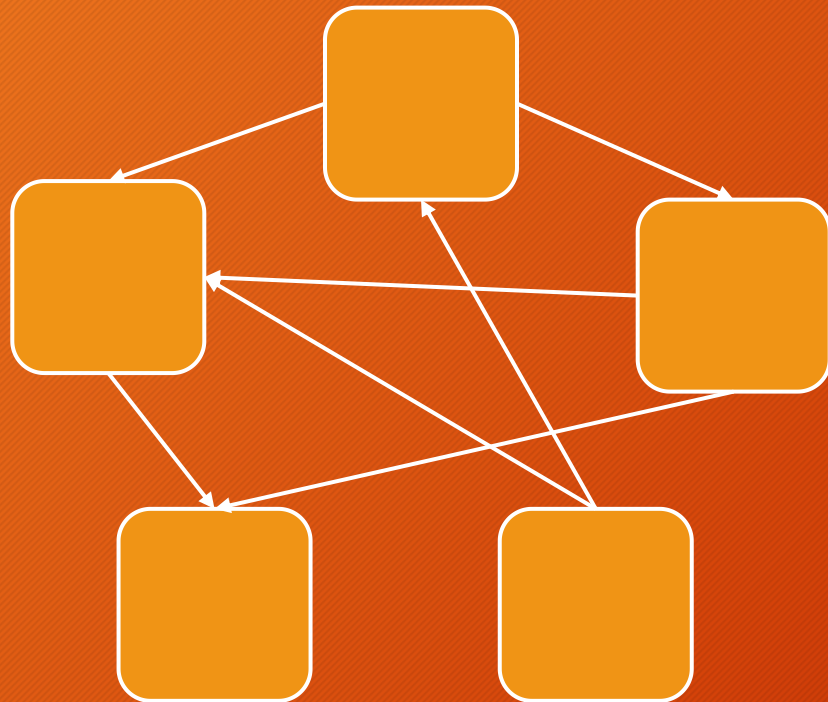


设计

- Role
- SwingDecorator
- DressDecorator

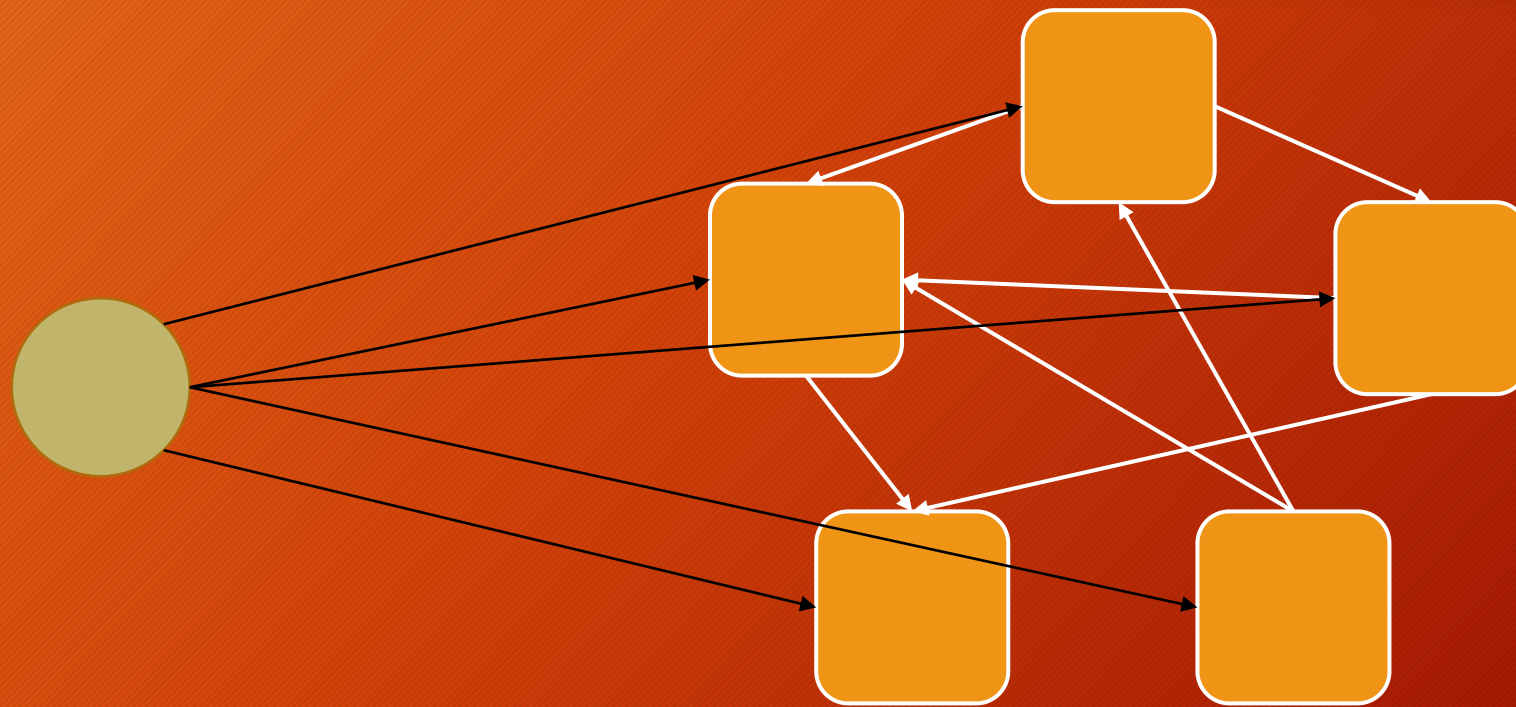
调停者Mediator

图解

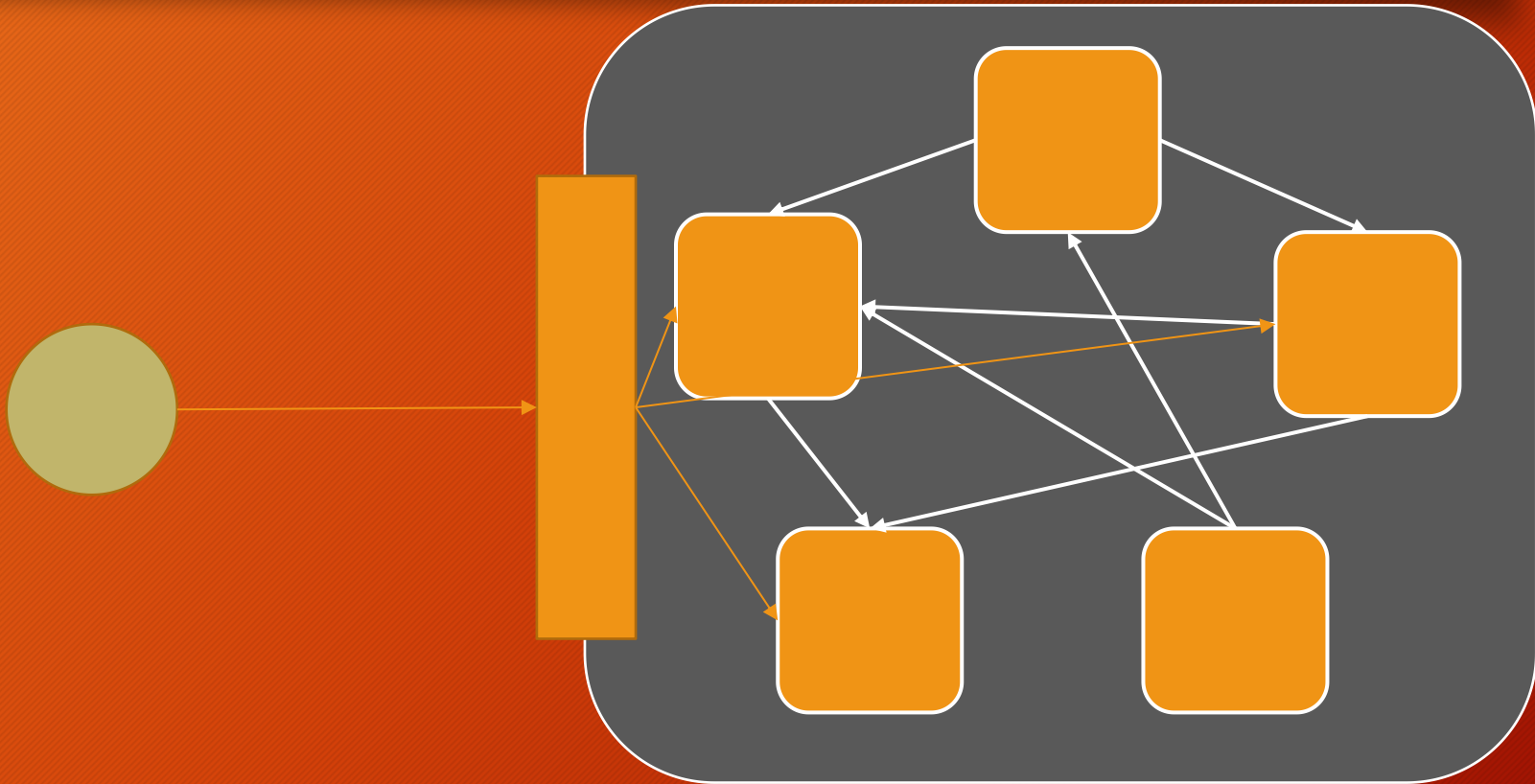


门面Facade

图解



改进



装饰器Decorator

问题:

- 坦克想加一个外壳显示
- 想加一个血条
- 想加一条尾巴
- 子弹想加一条尾巴
- 子弹想加一个外壳
- ...

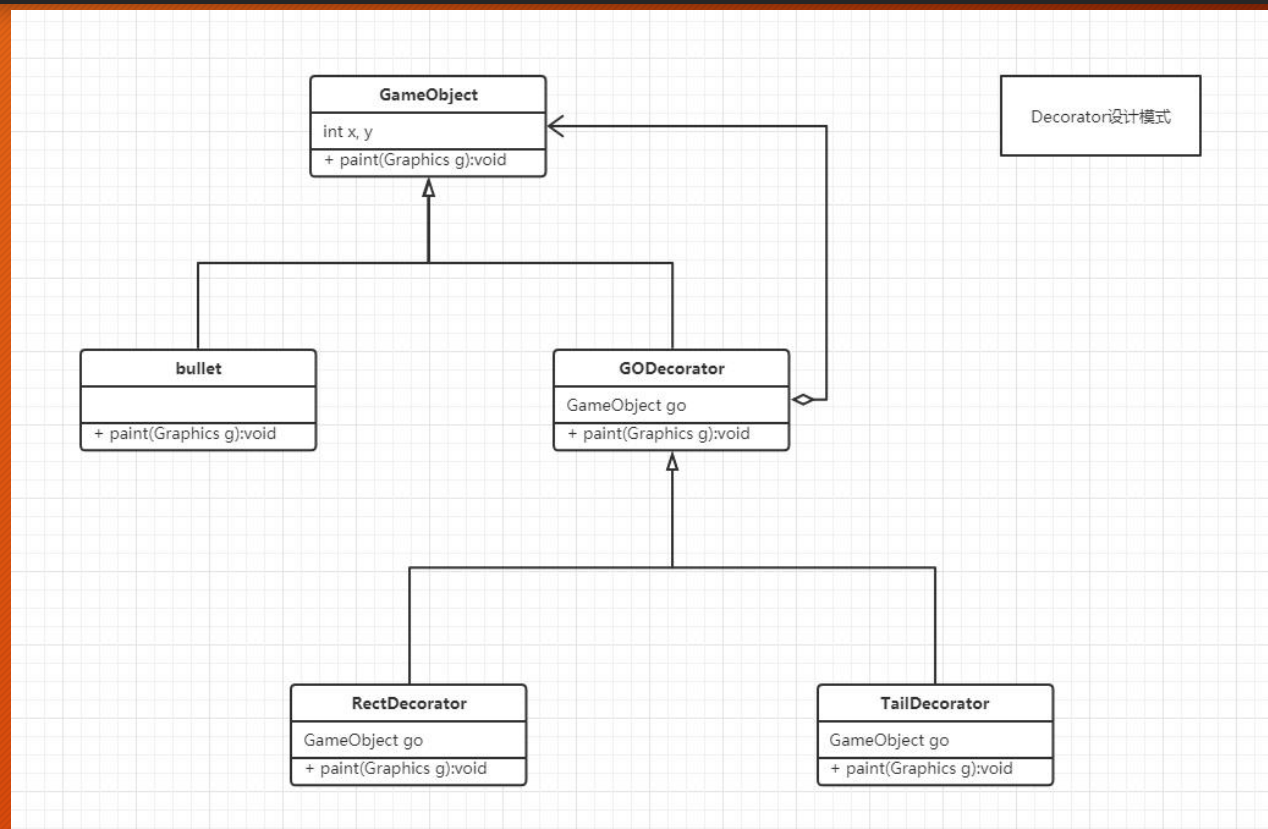
如果用继承:
BloodTank
TailBank
RectTank
BloodTailTank
BloodRectTank
TailBullet
....

不灵活: 装饰和被装饰者之间耦合度太高

进一步

- Tank
- TankDecorator
 - tank
 - paint() -> tank.paint() + 装饰
- tank -> GameObject

解决方案



探讨

- IO流
- reader and inputstream
- writer and outputstream

观察者Observer

事件处理模型

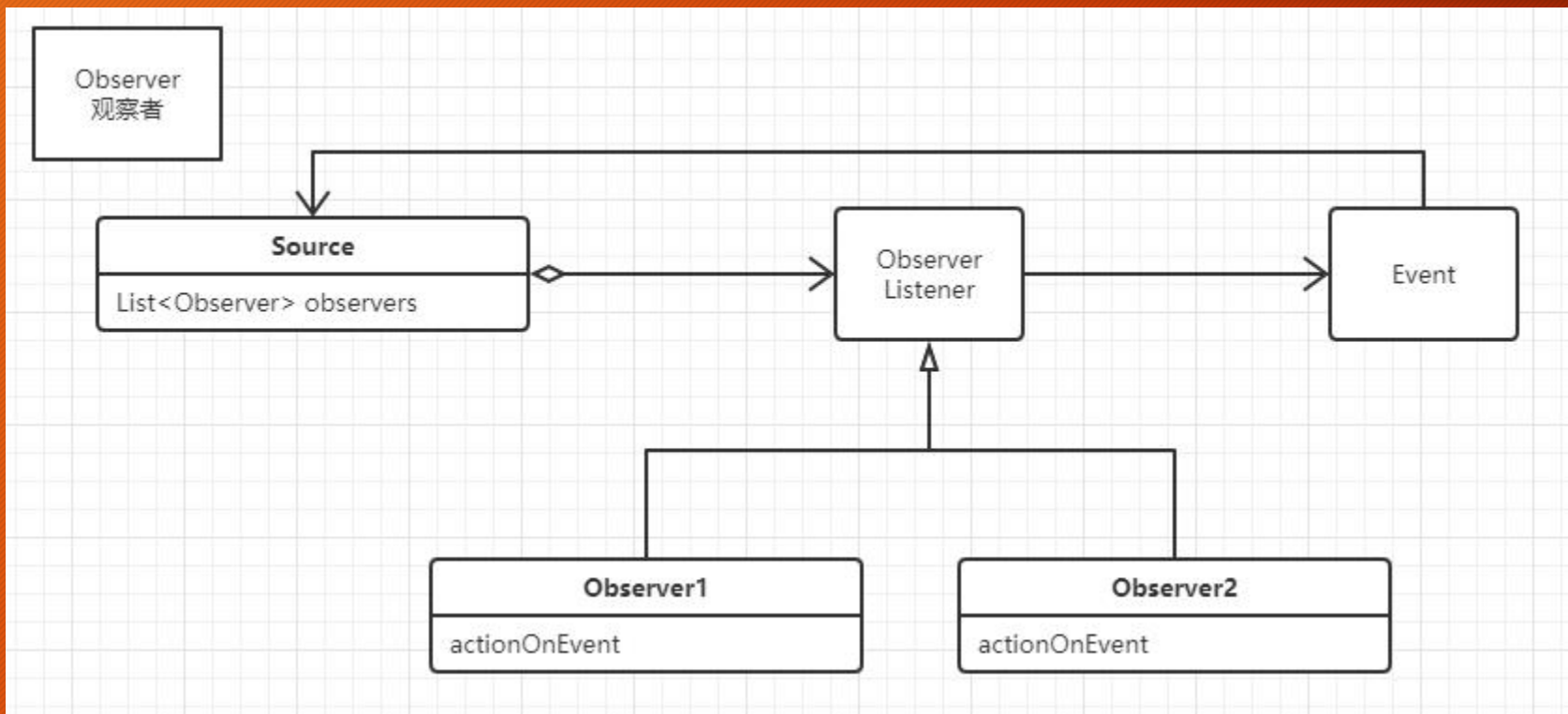
问题



小朋友睡醒了哭，饿！

解决方案

代码



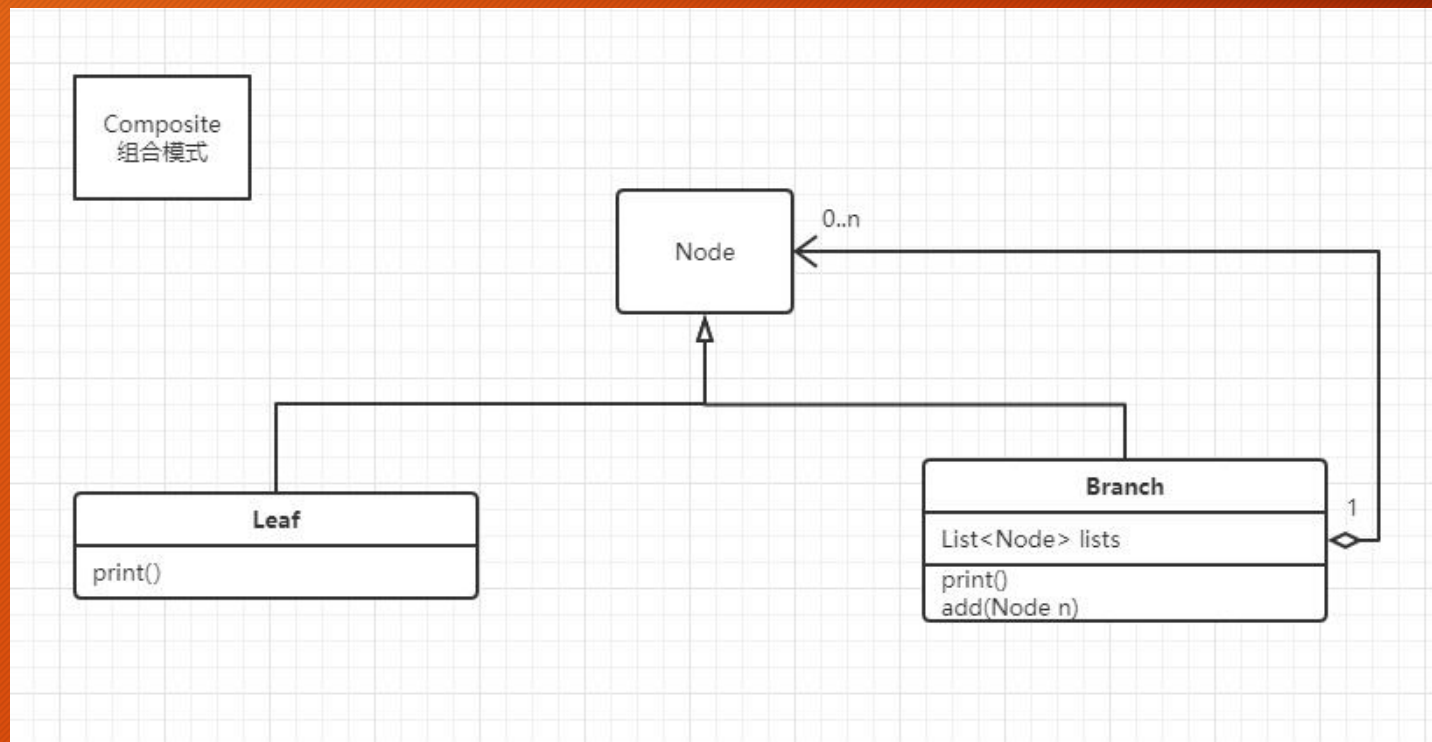
总结

- Observer
- Listener
- Hook
- Callback

组合Composite

树状结构专用模式

图解



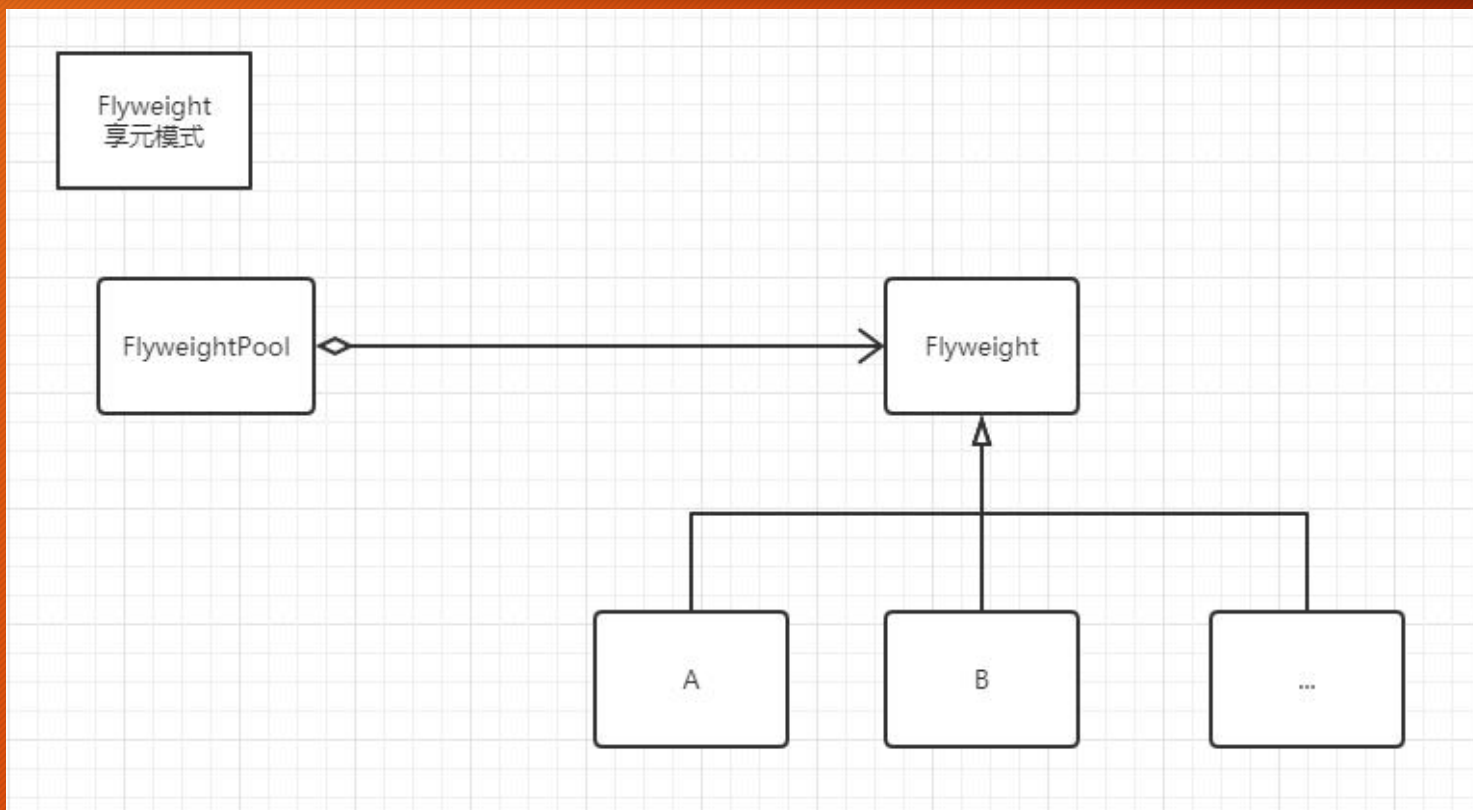
应用场景

- 树状结构

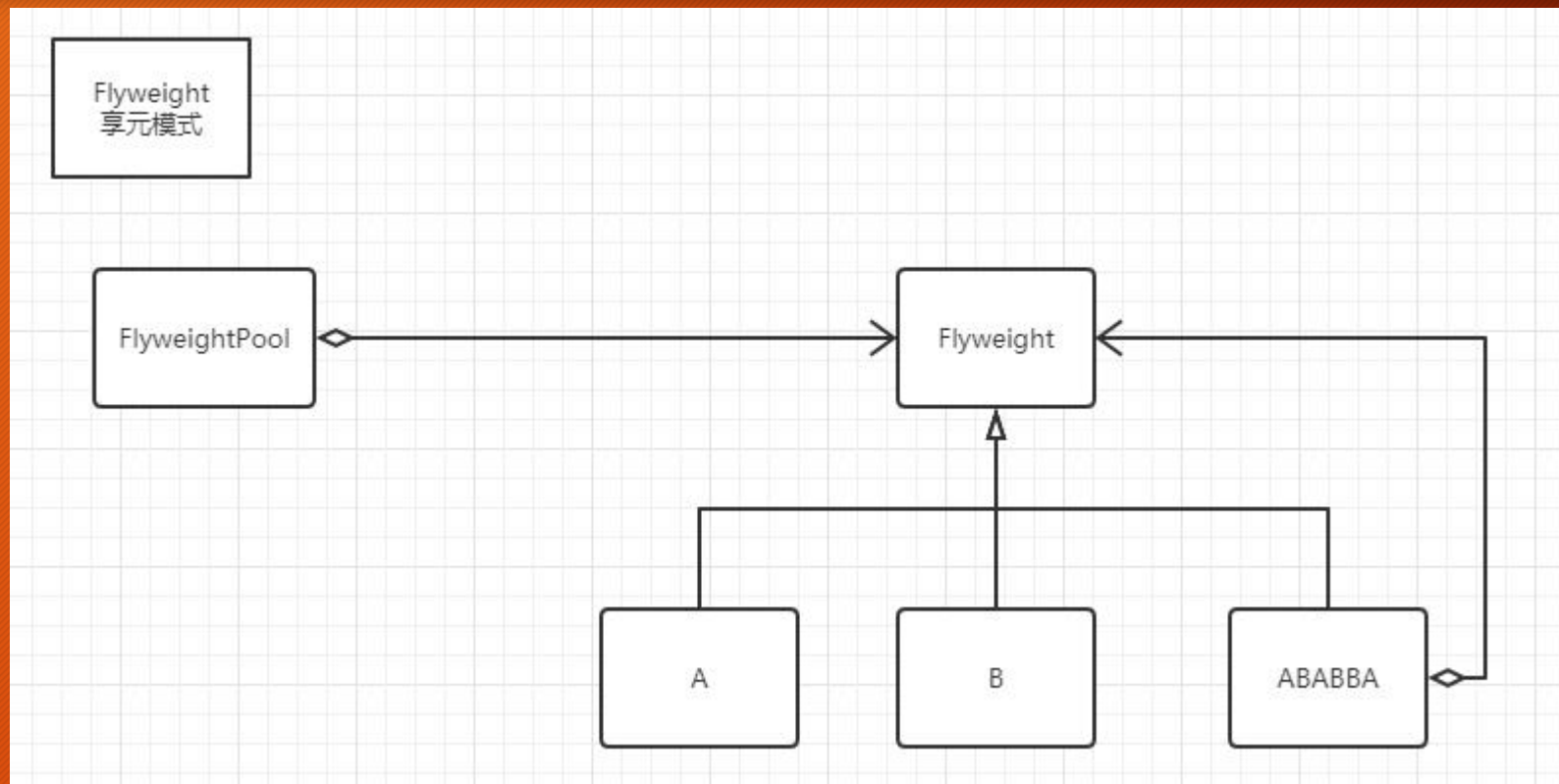
享元Flyweight

重复利用对象

图解



结合Composite



应用场景

- 字处理软件
- **String**类的实现

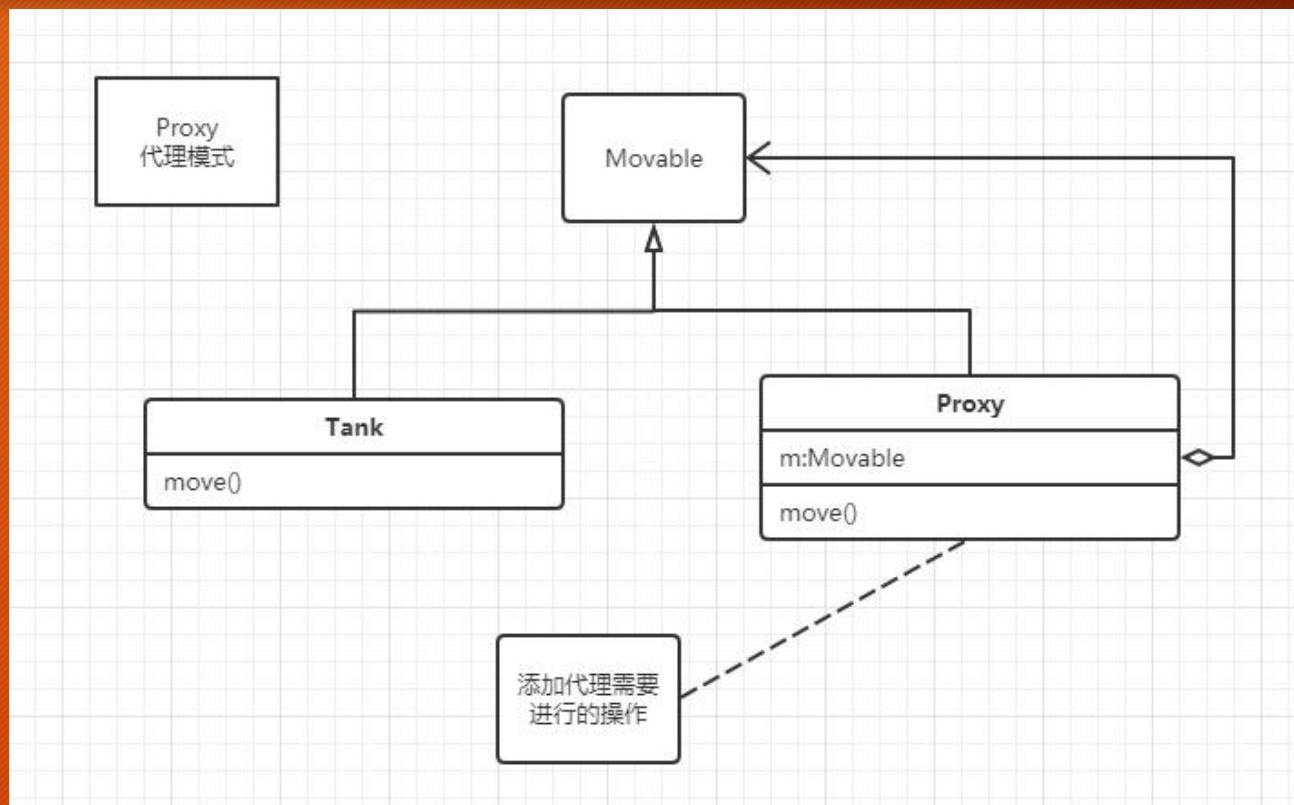
代理Proxy

静态代理

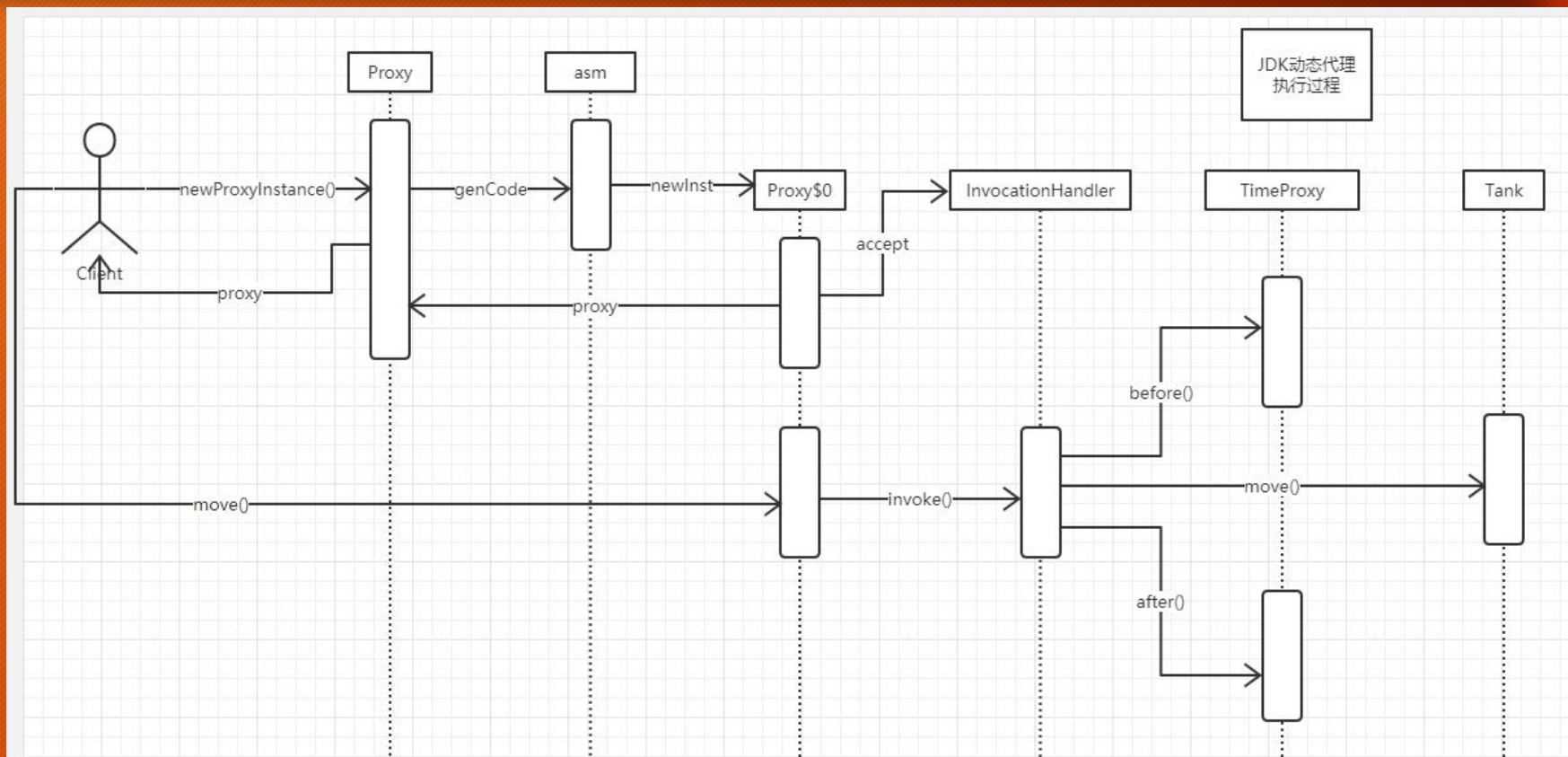
动态代理

Spring AOP

图解



JDK动态代理执行过程



SpringAOP

- IOC + AOP
- bean工厂 + 灵活装配 + 动态行为拼接，成就spring在java框架中的一哥地位

迭代器Iterator

容器与容器遍历

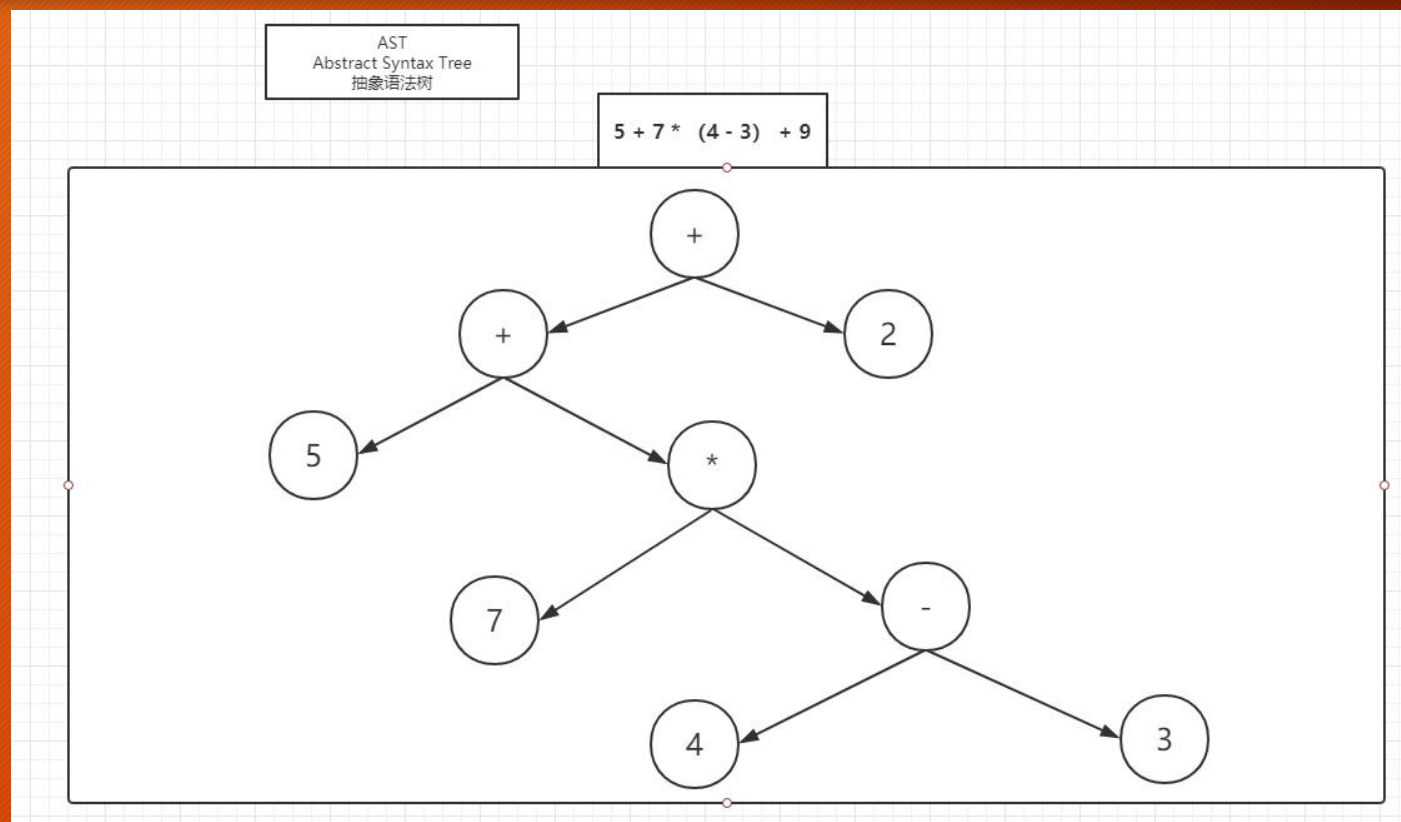
- 构建动态扩展的容器
 - List.add()
- 数组的实现
- 链表的实现

- 数组 vs 链表
 - 插入(中间)
 - 添加 尾部
 - 删除
 - 随机访问
 - 扩展

访问者Visitor

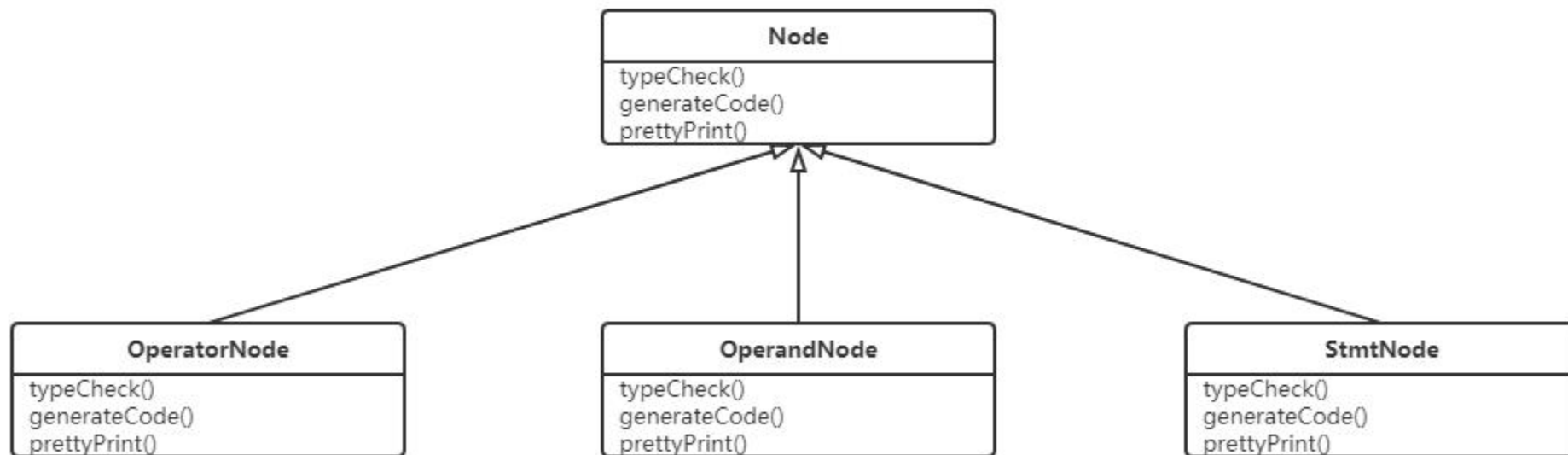
在结构不变的情况下动态改变对于内部元素的动作

GOF典型案例

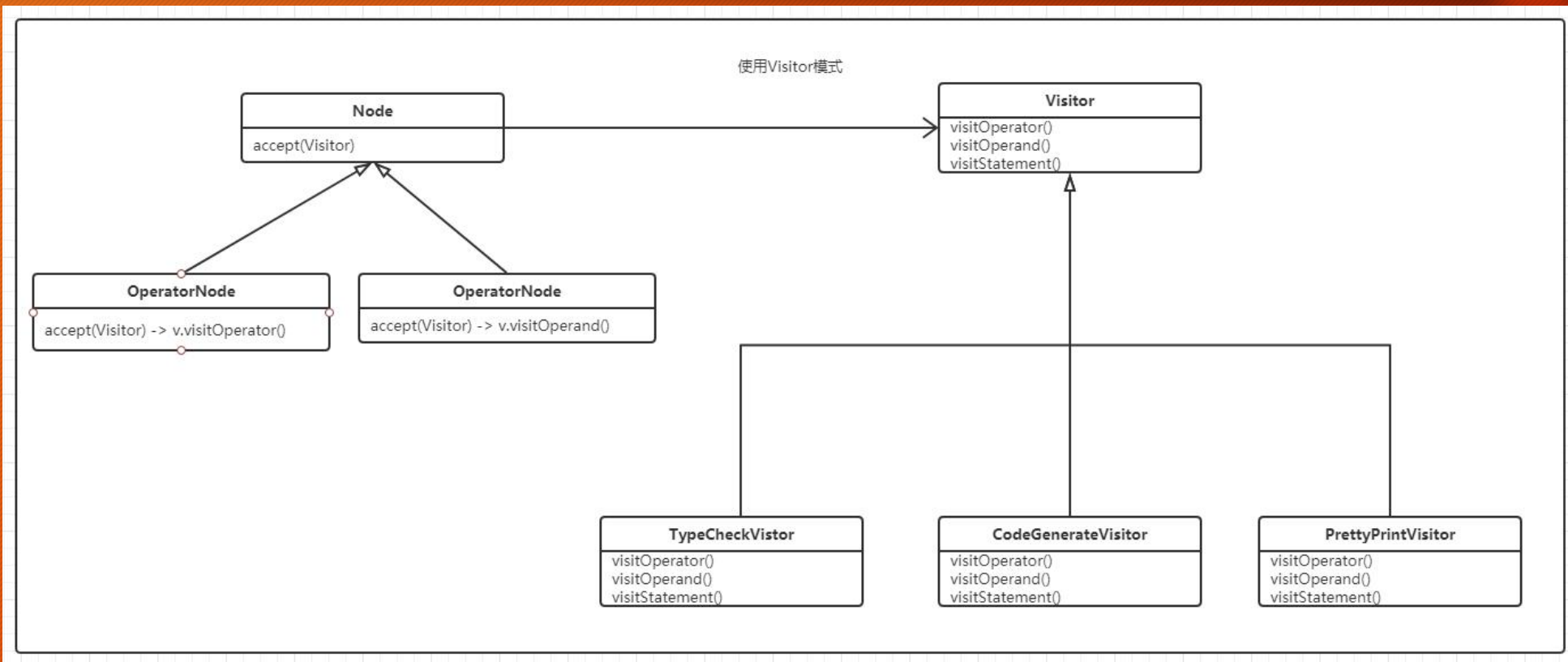


传统解决方案

传统编程模式



使用Visitor模式



ASM

- <http://asm.ow2.io>
- the ASM name does not mean anything: it is just a reference to the `__asm__` keyword in C, which allows some functions to be implemented in **AsSeMbly** language.

java class file

- **Magic:** 该项存放了一个 Java 类文件的魔数 (magic number) 和版本信息。一个 Java 类文件的前 4 个字节被称为它的魔数。每个正确的 Java 类文件都是以 0xCAFEBAE 开头的, 这样保证了 Java 虚拟机能很轻松的分辨出 Java 文件和非 Java 文件。
- **Version:** 该项存放了 Java 类文件的版本信息, 它对于一个 Java 文件具有重要的意义。因为 Java 技术一直在发展, 所以类文件的格式也处在不断变化之中。类文件的版本信息让虚拟机知道如何去读取并处理该类文件。
- **Constant Pool:** 该项存放了类中各种文字字符串、类名、方法名和接口名称、final 变量以及对外部类的引用信息等常量。虚拟机必须为每一个被装载的类维护一个常量池, 常量池中存储了相应类型所用到的所有类型、字段和方法的符号引用, 因此它在 Java 的动态链接中起到了核心的作用。常量池的大小平均占到了整个类大小的 60% 左右。
- **Access flag:** 该项指明了该文件中定义的是类还是接口 (一个 class 文件中只能有一个类或接口), 同时还指明了类或接口的访问标志, 如 public, private, abstract 等信息。
- **This Class:** 指向表示该类全限定名称的字符串常量的指针。

- **Super Class:** 指向表示父类全限定名称的字符串常量的指针。
- **Interfaces:** 一个指针数组, 存放了该类或父类实现的所有接口名称的字符串常量的指针。以上三项所指回的常量, 特别是前两项, 在我们用 ASM 从已有类派生新类时一般需要修改: 将类名称改为子类名称; 将父类改为派生前的类名称; 如果有必要, 增加新的实现接口。
- **Fields:** 该项对类或接口中声明的字段进行了细致的描述。需要注意的是, fields 列表中仅列出了本类或接口中的字段, 并不包括从超类和父接口继承而来的字段。
- **Methods:** 该项对类或接口中声明的方法进行了细致的描述。例如方法的名称、参数和返回值类型等。需要注意的是, methods 列表里仅存放了本类或本接口中的方法, 并不包括从超类和父接口继承而来的方法。使用 ASM 进行 AOP 编程, 通常是通过调整 Method 中的指令来实现的。
- **Class attributes:** 该项存放了在该文件中类或接口所定义的属性的基本信息。

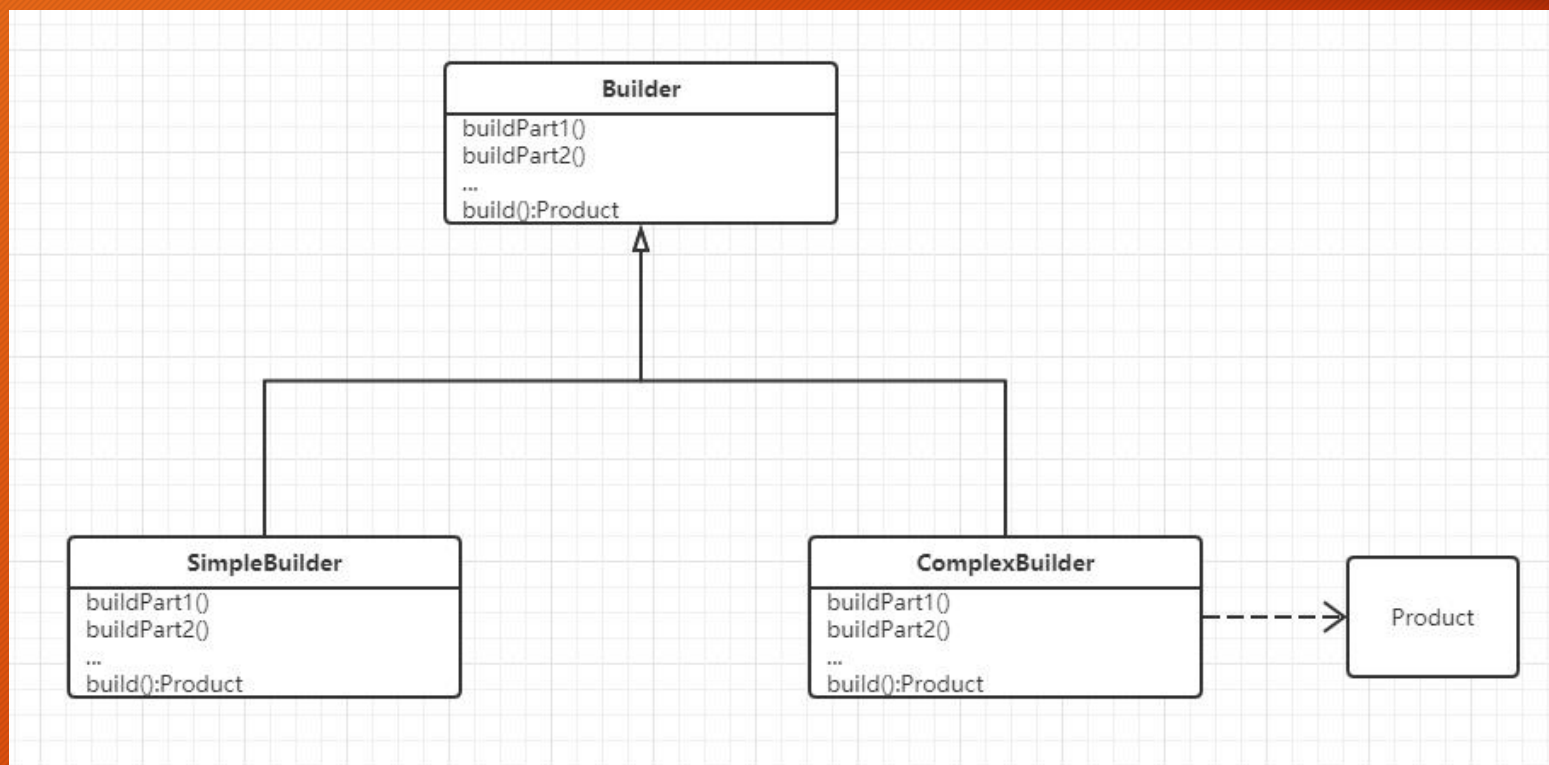
JVM虚拟机规范

- oracle网站查找

Builder

构建复杂对象

图解



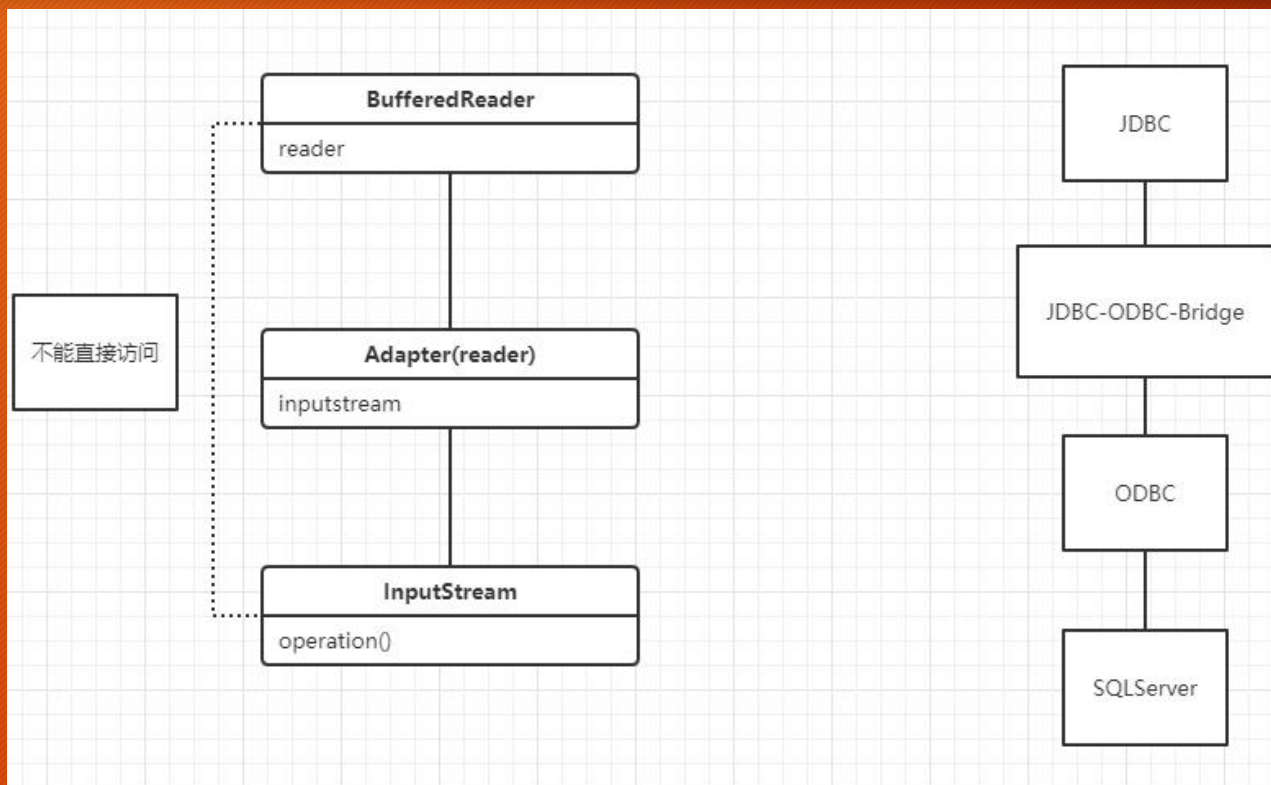
解析

- 分离复杂对象的构建和表示
- 同样的构建过程可以创建不同的表示
- 无需记忆，自然使用

Adapter(Wrapper)

接口转换器

图解



- 电压转接头
- java.io
- jdbc-odbc bridge (不是桥接模式)
- ASM transformer

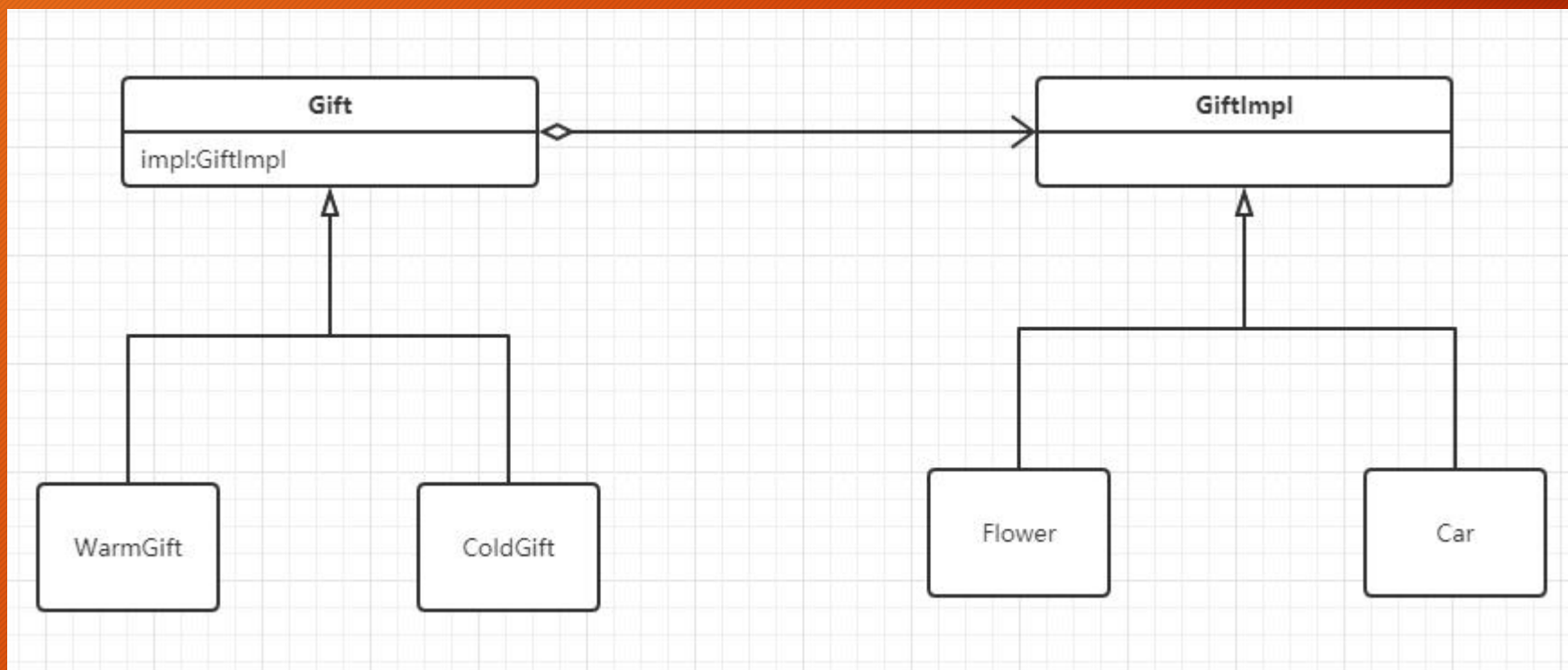
误区

- 常见的Adapter类反而不是Adapter
- WindowAdapter
- KeyAdapter

Bridge

双维度扩展

图解



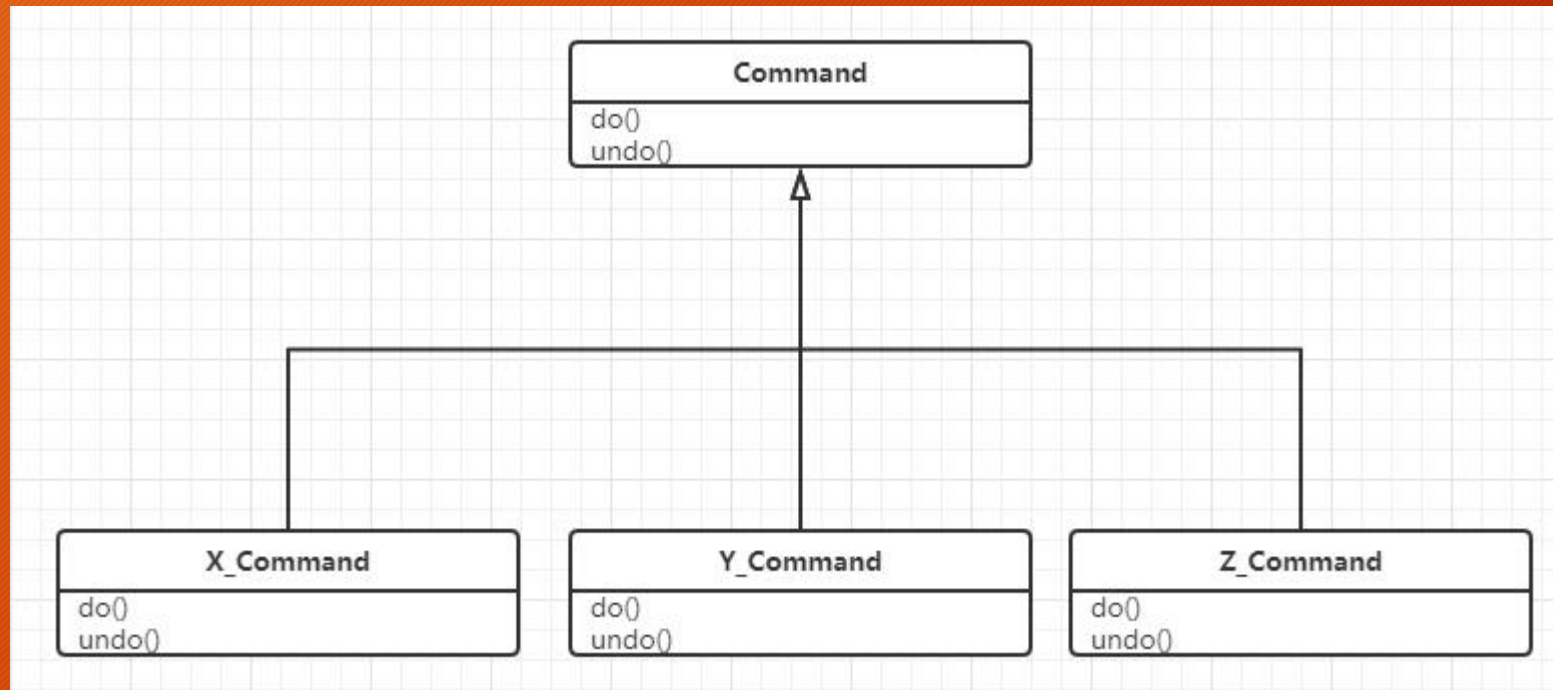
解析

- 分离抽象与具体
- 用聚合方式（桥）连接抽象与具体

Command

封装命令
结合cor实现undo

图解



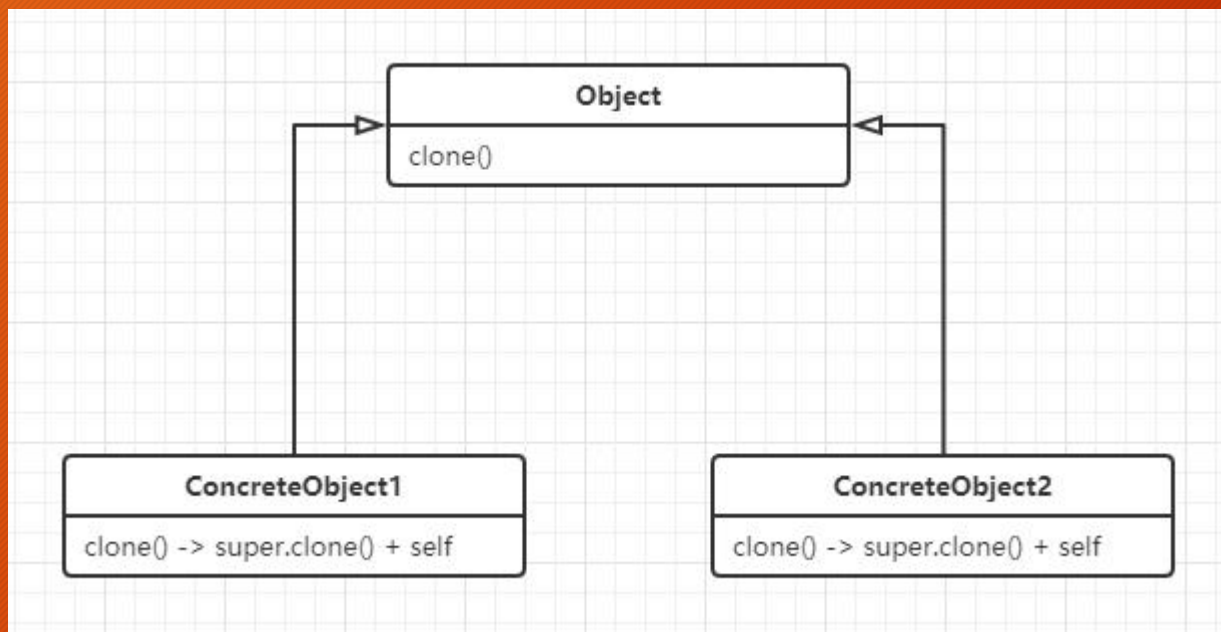
解析

- 别名: Action / Transaction
- 宏命令
 - command与?模式
- 多次undo
 - command与?模式
- trasaction回滚
 - command与?模式

Prototype原型模式

Object .clone()

图解



java中的原型模式

- 自带
- 实现原型模式需要实现标记型接口Cloneable
- 一般会重写clone()方法
 - 如果只是重写clone()方法，而没实现接口，调用时会报异常
- 一般用于一个对象的属性已经确定，需要产生很多相同对象的时候
- 需要区分深克隆与浅克隆

Memento备忘录

记录状态
便于回滚

图解



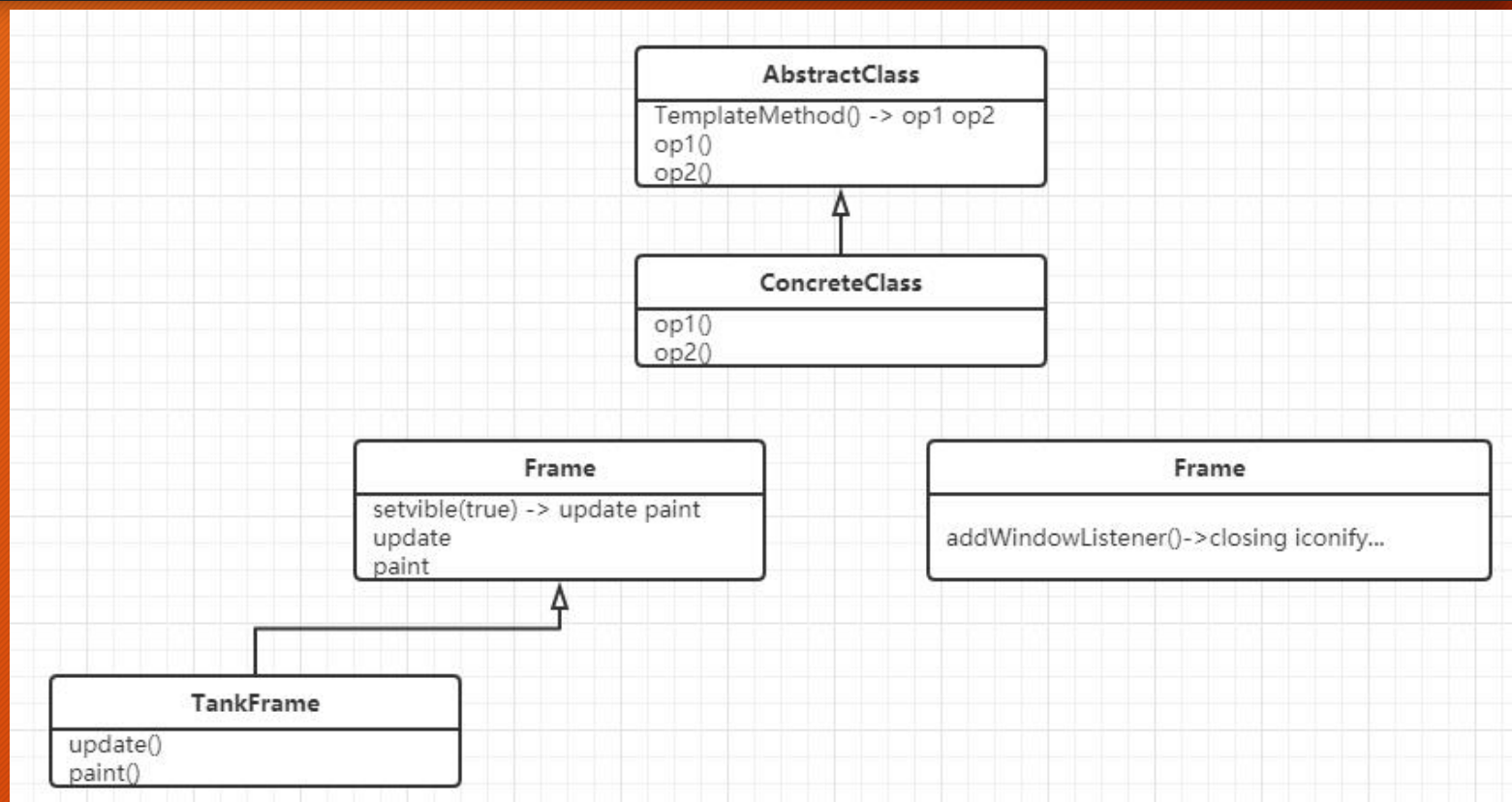
解析

- 记录快照（瞬时状态）
- 存盘

TemplateMethod模板方法

钩子函数

图解



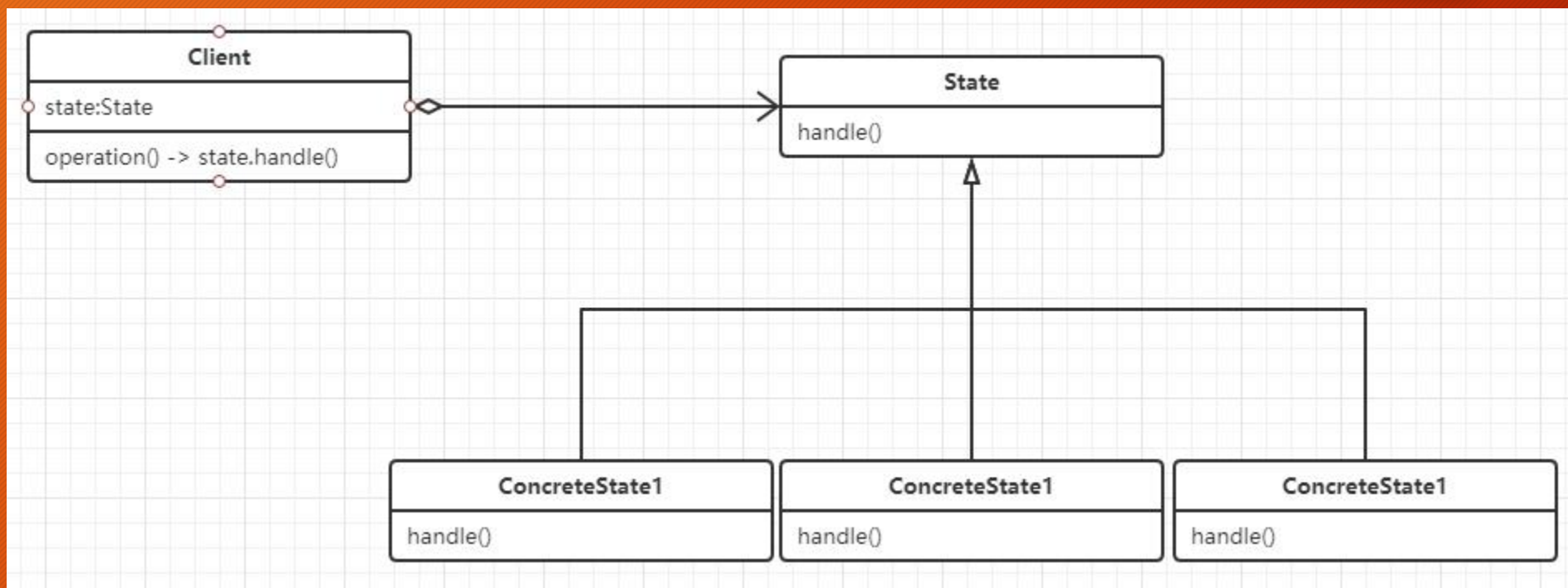
解析

- 其实，你一直在用
 - `paint(Graphics g)`
 - `WindowListener`
 - `windowClosing()`
 - `windowXXX()`
 - `ASM`
 - `ClassVisitor`

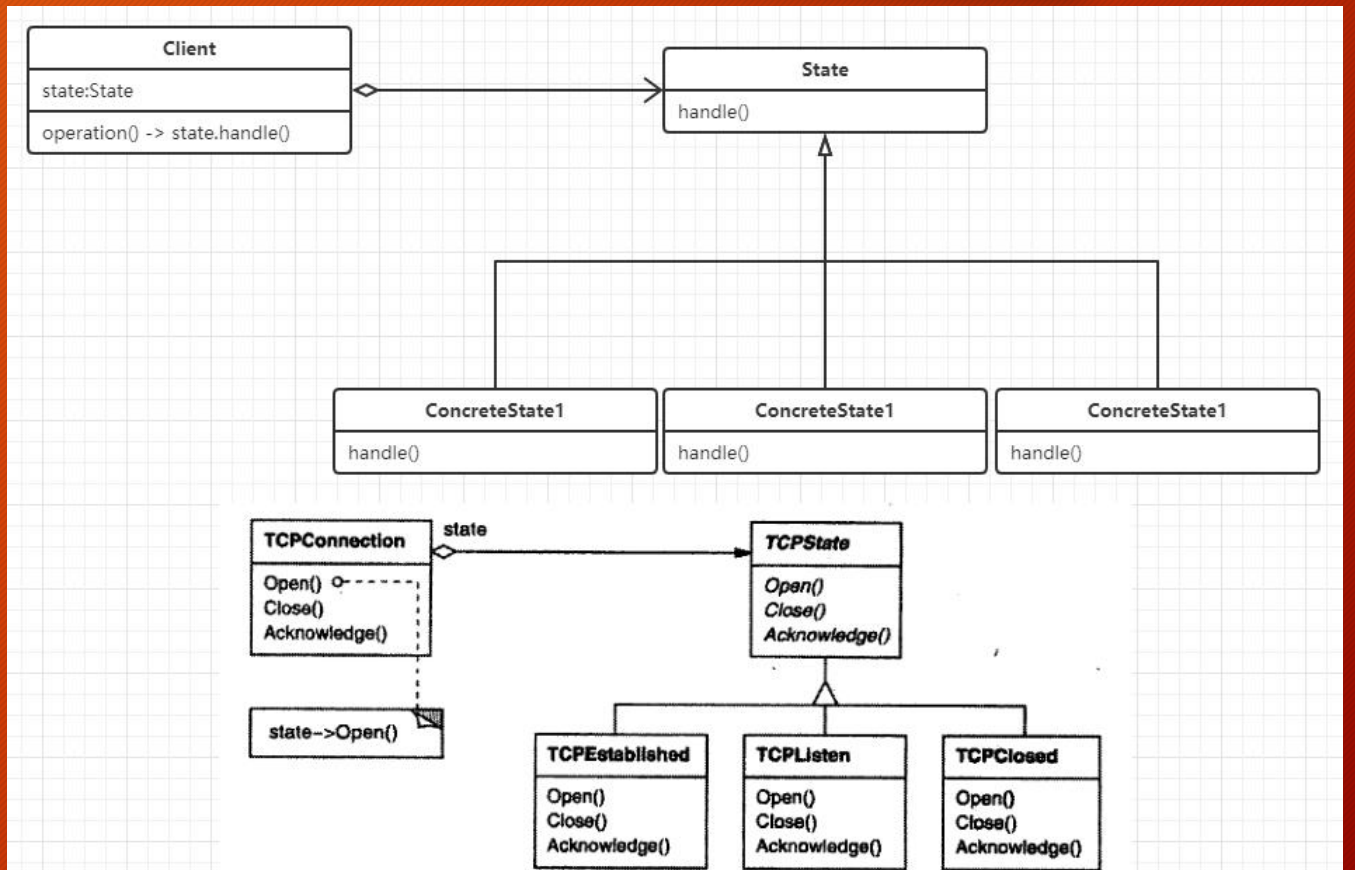
State状态模式

根据状态决定行为

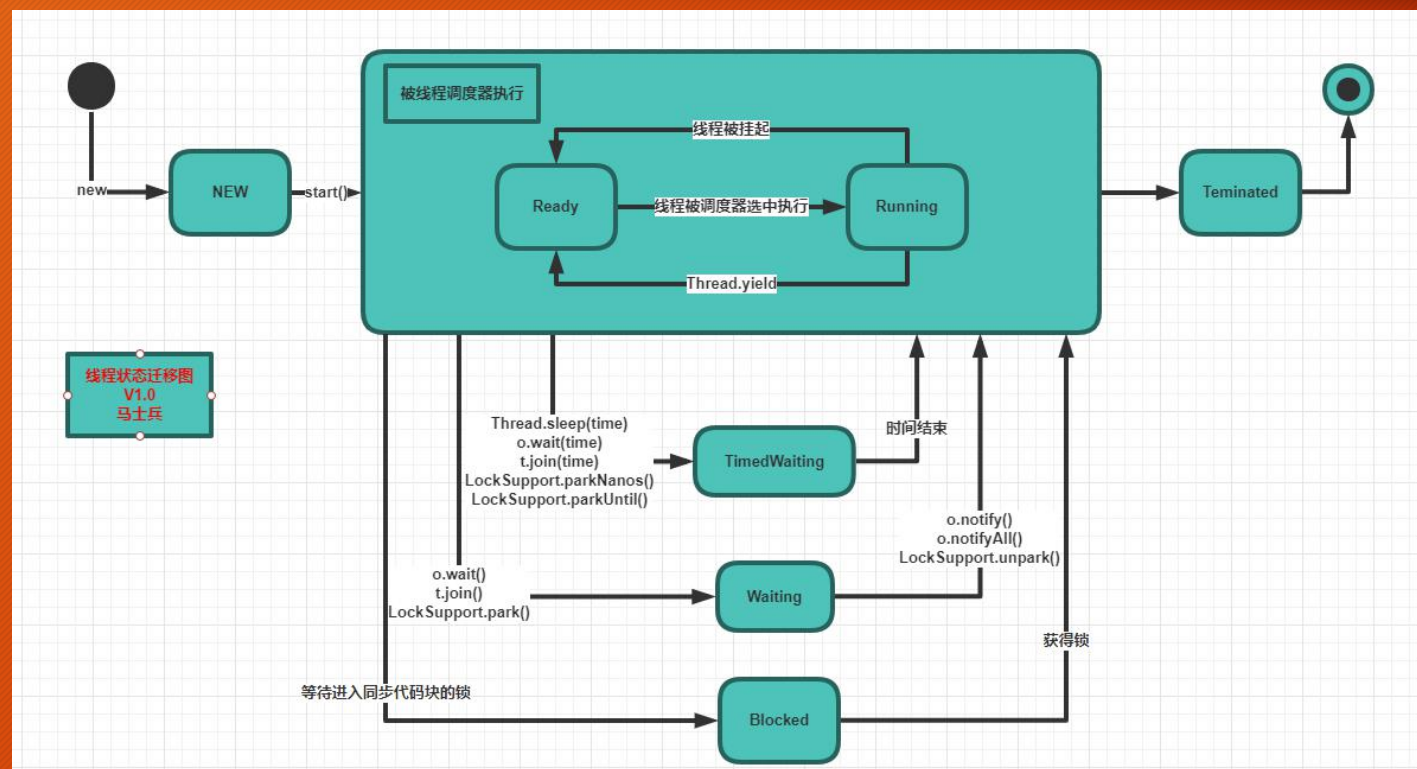
图解







GOF eg.



有限状态机(FSM)



作业：Car的State与Action

State/Action	open the door	close the door	run the car	stop the car
Open				
Closed				
Running				
Stopped				

Interpreter解释器

动态脚本解析

复习

设计模式列表

创建型模式

1. Abstract Factory..
2. Builder
3. Factory Method..
4. Prototype..
5. Singleton.

结构型模式

1. Adapter.
2. Bridge..
3. Composite..
4. Decorator.
5. Facade..
6. Flyweight.
7. Proxy..

行为型模式

1. Chain of Responsibility.
2. Command.
3. Interpreter.
4. Iterator
5. Mediator
6. Memento..
7. Observer
8. State..
9. Strategy.
10. Template Method..
11. Visitor.

面向对象六大原则

指导思想

- 可维护性Maintainability
 - 修改功能，需要改动的地方越少，可维护性就越好
- 可复用性Reusability
 - 代码可以被以后重复使用
 - 写出自己总结的类库
- 可扩展性Extensibility/Scalability
 - 添加功能无需修改原来代码
- 灵活性flexibility / mobility / adaptability
 - 代码接口可以灵活调用

单一职责原则

- Single Responsibility Principle
- 一个类别太大，别太累，负责单一的职责
 - Person
 - PersonManager
- 高内聚，低耦合

开闭原则

- Open-Closed Principle
- 对扩展开放，对修改关闭
 - 尽量不修改原来代码的情况下进行扩展
- 抽象化，多态是开闭原则的关键

里氏替换原则

- Liscov Substitution Principle
- 所有使用父类的地方，必须能够透明的使用子类对象

依赖倒置原则

- Dependency Inversion Principle
- 依赖倒置原则
 - 依赖抽象，而不是依赖具体
 - 面向抽象编程

接口隔离原则

- Interface Segregation Principle
- 每一个接口应该承担独立的角色，不干不该自己干的事儿
 - Flyable Runnable 不该合二为一
 - 避免子类实现不需要实现的方法
 - 需要对客户提供接口的时候，只需要暴露最小的接口

迪米特法则

- Law of Demeter
- 尽量不要和陌生人说话
- 在迪米特法则中，对于一个对象，非陌生人包括以下几类：
 - 当前对象本身(this);
 - 以参数形式传入到当前对象方法中的对象;
 - 当前对象的成员对象;
 - 如果当前对象的成员对象是一个集合，那么集合中的元素也都是朋友;
 - 当前对象所创建的对象。
- 和其他类的耦合度变低

总结

- OCP：总纲，对扩展开放，对修改关闭
- SRP：类的职责要单一
- LSP：子类可以透明替换父类
- DIP：面向接口编程
- ISP：接口的职责要单一
- LoD：降低耦合