# 引言

## 参考资料

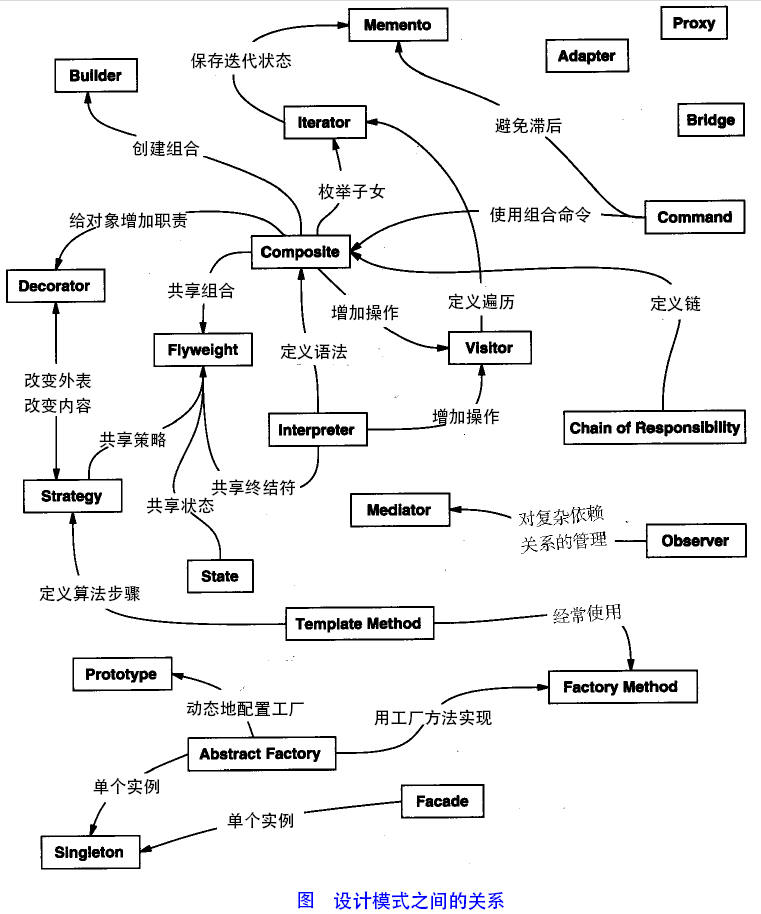
<https://www.cnblogs.com/susanws/p/5510229.html>

<https://www.cnblogs.com/ahangBlogs/p/7854098.html>

<http://www.cnblogs.com/Leo_wl/p/6067267.html>

<https://blog.csdn.net/jason0539/article/details/44956775>

## 设计模式关系图



## 类

### 数据

### 行为

## 类之间的关系

### IS-A（是一个）

### HAS-A（有一个）

### IMPLEMENTS（实现）

## 面向对象

### 封装

### 继承

### 多态

### 抽象

# 设计原则

## 分离封装变化原则

找出应用中可能需要变化之处，把它们独立出来，不要和那些不需要变化的代码混在一起。

把会变化的部分取出并封装起来，以便以后可以轻易地改动或扩充此部分，而不影响不需要变化的其他部分。

## 多用组合原则

多用组合，少用继承。

## 接口编程原则

针对接口编程，而不是针对实现。

接口可以是接口，也可以是抽象类，关键在于多态。

## 松耦合原则

为了交互对象之间的松耦合设计而努力。

## 开放-关闭原则

类应该对扩展开放，对修改关闭。

## 依赖倒置原则

要依赖抽象，不要依赖具体类。

## 最少知识原则

### 定义

只和你的密友谈话。减少对象之间的交互，只留下几个“密友”。

### 得墨忒耳法则(Law of Demeter)/迪米特法则

同一个意思。

### 如何做到

就任何对象而言，在该对象的方法内，我们只应该调用属于以下范围的方法：

#### 该对象本身

#### 被当做方法的参数而传递进来的对象

#### 此方法所创建或实例化的任何对象

#### 对象的任何组件

## 好莱坞原则

### 定义

别调用（打电话给）我们，我们会调用（打电话给）你。

### 详细解释

好莱坞原则可以给我们一种防止“依赖腐败”的方法。

允许低层组件将自己挂钩到系统上，但是高层组件会决定什么时候和怎样使用这些低层组件。

## 单一变化原则

一个类应该只只有一个引起变化的原因。

内聚。

# 创建型

## 工厂方法模式

### 定义

定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个，工厂方法让类把实例化推迟到子类。

### 前提

#### 产品类

#### 创建者类

### 接口

#### Creator

### 注意

## 抽象工厂模式

### 定义

提供一个接口，用于创建相关或依赖对象的家族，而不需要明确指定具体类。

### 前提

### 接口

#### AbstractFactory

### 注意

### 工厂方法模式和抽象工厂模式的区别

工厂方式模式提供一个抽象接口来创建“一个产品”。通过继承让每个子类来“决定”实例化哪个“具体类”。

抽象工厂模式定义的接口，包含一组方法来生产产品。使用的时候通过组合来实现。

## 单例模式

### 定义

确保一个类只有一个实例，并提供一个全局访问点。

### 前提

### 接口

### 注意

#### 构造函数私有

#### 线程同步问题

##### 双重检查加锁

#### 多个类加载器

自行指定类加载器，并指定同一个类加载器。

## 建造者模式

## 原型模式

# 结构型

## 适配器模式

### 定义

讲一个类的接口，转换成客户期望的另一个接口。适配器让原本接口不兼容的类可以合作无间。

### 前提

#### 目标接口

#### 被适配者的实例

#### 一个适配器一个被适配者

#### 双向适配器

### 接口

#### “对象”适配器

组合。

#### “类”适配器

多重继承。

### 注意

## 装饰器模式

### 定义

动态地将责任附加到对象上。若要扩展功能，装饰者提供了比继承更有弹性的替代方案。

### 前提

#### 装饰者

##### 装饰者抽象类。

##### 具体装饰者。

#### 被装饰对象

##### 被装饰对象抽象类

##### 具体被装饰对象。

### 接口

### 注意

#### 相同的超类型

装饰者和被装饰对象有相同的超类型。

#### 多次包装

可以用一个或多个装饰者包装一个对象。

#### 定制行为

装饰者可以在所委托被装饰者的行为之前/之后，加上自己的行为，以达到特定的目的。

### Jdk相关

Java.io中InputStream相关的类使用了装饰者模式。

## 代理模式

## 外观模式（Facade Pattern）

### 定义

提供了一个统一的接口，用来访问子系统中的一群接口。外观定义了一个高层接口，让子系统更容易使用。

### 前提

### 接口

### 注意

## 桥接模式

## 组合模式（Composite Pattern）

### 定义

允许你将对象组合成树形结构来表现“整体/部分”层次结构。组合能让客户以一致的方式处理个别对象以及对象组合。

### 前提

### 接口

### 注意

## 享元模式

# 行为型

## 策略模式

### 定义

定义了算法族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化独立于使用算法的客户。

### 前提

### 接口

### 注意

## 模板方法模式

### 定义

在一个方法中定义一个算法的骨架，而将一些步骤延迟到子类中。模板方法使得子类可以在不改变算法结构的情况下，重新定义算法中的某些步骤。

### 前提

### 接口

Public abstract class AbstractClass {

Void templateMethod() {

Operation1();

Operation2();

......

Hook();

}

Abstract void opertion1();

Void hook() {

/\* 一个什么都不做的方法。 \*/

}

}

### 注意

## 观察者模式

### 定义

定义了对象之间的一对多依赖，这样一来，当一个对象改变状态时，它的所有依赖者都会收到通知并自动更新。

### 前提

#### 一对多关系

一个主题对象 --- 多个观察者

#### 主题对象

具有状态，自动更新/通知许多观察者

#### 依赖者对象

众多的观察者

### 接口

#### 主题对象

public interface Subject

{

Void registerObserver(Observer o);

Void removeObserver(Observer o);

Void notifyObservers();

}

#### 观察者

public interface Observer

{

Void update(double data);

}

### 注意

#### 获取数据方式

##### 推送(push)

由主题对象在调用update方法时传递个观察者。

##### 拉取(pull)

由观察者在update时从主题对象中获取。

### Jdk相关

java.util.Observer和java.util.Observerable。

## 迭代器模式

### 定义

提供一种方法顺序访问一个聚合对象中的各个元素，而又不暴露其内部的表示。

### 前提

### 接口

Public interface Iterator {

Boolean hasNext();

Object next();

}

### 注意

## 责任链模式

## 命令模式

### 定义

将“请求”封装成对象，以便使用不同的请求、队列或者日志来参数化其他对象。命令模式也支持可撤销的操作。

### 前提

#### 动作的请求者

#### 动作的执行者

### 接口

Public interface Command {

Void execute();

Void undo();

}

### 注意

## 备忘录模式

## 状态模式

### 定义

允许对象在内部状态改变时改变它的行为，对象看起来好像修改了它的类。

### 前提

### 接口

#### State

状态在Context中使用。

#### Context/State Machine

Context对外使用。

### 注意

## 访问者模式

## 中介者模式

## 解释器模式

# 并发型

# 线程池

# MVC(Model-View-Controller)