设计模式

单例模式

> 解决的是如何在整个项目中创建唯一对象实例的问题，工厂模式解决的是如何不通过**new**建立实例对象的方法。

1.单例模式

- $_instance必须声明为静态的私有变量
- 构造函数和析构函数必须声明为私有,防止外部程序new 类从而失去单例模式的意义
- getInstance()方法必须设置为公有的,必须调用此方法 以返回实例的一个引用
- ::操作符只能访问静态变量和静态函数
- new对象都会消耗内存
- 使用场景:最常用的地方是数据库连接。
- 使用单例模式生成一个对象后， 该对象可以被其它众多对象所使用。
- 私有的__clone()方法防止克隆对象

单例模式，使某个类的对象仅允许创建一个。构造函数private修饰，申明一个static getInstance方法，在该方法里创建该对象的实例。如果该实例已经存在，则不创建。比如只需要创建一个数据库连接。

2.工厂模式

工厂模式，工厂方法或者类生成对象，而不是在代码中直接new。 使用工厂模式，可以避免当改变某个类的名字或者方法之后，在调用这个类的所有的代码中都修改它的名字或者参数。

3. 注册模式

注册模式，解决全局共享和交换对象。已经创建好的对象，挂在到某个全局可以使用的数组上，在需要使用的时候，直接从该数组上获取即可。将对象注册到全局的树上。任何地方直接去访问。

4. 适配器模式

将各种截然不同的函数接口封装成统一的API。 PHP中的数据库操作有MySQL,MySQLi,PDO三种，可以用适配器模式统一成一致，使不同的数据库操作，统一成一样的API。类似的场景还有cache适配器，可以将memcache,redis,file,apc等不同的缓存函数，统一成一致。 首先定义一个接口(有几个方法，以及相应的参数)。然后，有几种不同的情况，就写几个类实现该接口。将完成相似功能的函数，统一成一致的方法。

5.策略模式

策略模式，将一组特定的行为和算法封装成类，以适应某些特定的上下文环境。 （如电商系统男女不同的用户展示不同的信息）

6. 观察者模式

1：观察者模式(Observer)，当一个对象状态发生变化时，依赖它的对象全部会收到通知，并自动更新。 2：场景:一个事件发生后，要执行一连串更新操作。传统的编程方式，就是在事件的代码之后直接加入处理的逻辑。当更新的逻辑增多之后，代码会变得难以维护。这种方式是耦合的，侵入式的，增加新的逻辑需要修改事件的主体代码。 3：观察者模式实现了低耦合，非侵入式的通知与更新机制。

7. 装饰器模式

1：装饰器模式，可以动态的添加修改类的功能 2：一个类提供了一项功能，如果要在修改并添加额外的功能，传统的编程模式，需要写一个子类继承它，并重写实现类的方法 3：使用装饰器模式，仅需要在运行时添加一个装饰器对象即可实现，可以实现最大额灵活性。

8. 原型模式

原型模式（对象克隆以避免创建对象时的消耗） 1：与工厂模式类似，都是用来创建对象。 2：与工厂模式的实现不同，原型模式是先创建好一个原型对象，然后通过clone原型对象来创建新的对象。这样就免去了类创建时重复的初始化操作。 3：原型模式适用于大对象的创建，创建一个大对象需要很大的开销，如果每次new就会消耗很大，原型模式仅需要内存拷贝即可。