（设计模式，集成测试工具，文档模版）

我们可能有用的设计模式：
工厂模式，单例模式，接口的适配器模式，代理模式，外观模式，命令模式，状态模式

设计模式及其使用场景	
工厂模式	出现大量的产品需要创建，并且具有共同的接口时，通过工厂方法模式进行创建
单例模式	只能有一个实例化的对象存在
建造者模式	将各种产品集中起来进行管理，用来创建复合对象
原型模式	将一个对象作为原型，对其进行复制、克隆，产生一个和原对象类似的新对象
适配器模式	类的适配器模式：当希望将一个类转换成满足另一个新接口的类时，可以使用类的适配器模式，创建一个新类，继承原有的类，实现新的接口即可。 对象的适配器模式：当希望将一个对象转换成满足另一个新接口的对象时，可以创建一个 Wrapper 类，持有原类的一个实例，在 Wrapper 类的方法中，调用实例的方法就行。 接口的适配器模式：当不希望实现一个接口中所有的方法时，可以创建一个抽象类 Wrapper，实现所有方法，我们写别的类的时候，继承抽象类即可。
装饰器模式	装饰器模式的应用场景： 1、需要扩展一个类的功能。 2、动态的为一个对象增加功能，而且还能动态撤销。（继承不能做到这一点，继承的功能是静态的，不能动态增删） 缺点：产生过多相似的对象，不易排错！
代理模式	如果已有的方法在使用的时候需要对原有的方法进行改进，此时有两种办法： 1、修改原有的方法来适应。这样违反了“对扩展开放，对修改关闭”的原则。 2、采用一个代理类调用原有的方法，且对产生的结果进行控制。即代理模式。
外观模式	降低类类之间的耦合度 CPU、Memory、Disk 他们之间将会相互持有实例，产生关系，这样会造成严重的依赖，修改一个类，可能会带来其他类的修改，这不是我们想要看到的，有了 Computer 类，他们之间的关系被放在了 Computer 类里，这样就起到了解耦的作用，这，就是外观模式！
桥接模式	类似 JDBC 桥 DriverManager
组合模式	将多个对象组合在一起进行操作，常用于表示树形结构中，例如二叉树等
享元模式	实现对象的共享，即共享池，当系统中对象多的时候可以减少内存的开销，通常与工厂模式一起使用。
策略模式	用在算法决策系统中，外部用户只需要决定用哪个算法即可

设计模式及其使用场景	
观察者模式	当一个对象变化时，其它依赖该对象的对象都会收到通知，并且随着变化
迭代子模式	一是需要遍历的对象，即聚集对象； 二是迭代器对象，用于对聚集对象进行遍历访问
责任链模式	有多个对象，每个对象持有对下一个对象的引用，这样就会形成一条链，请求在这条链上传递，直到某一对象决定处理该请求。但是发出者并不清楚到底最终那个对象会处理该请求。责任链模式可以实现，在隐瞒客户端的情况下，对系统进行动态的调整。
命令模式	达到命令的发出者和执行者之间解耦，实现请求和执行分开
备忘录模式	保存一个对象的某个状态，以便在适当的时候恢复对
状态模式	当对象的状态改变时，同时改变其行为。 1、可以通过改变状态来获得不同的行为。 2、你的好友能同时看到你的变化。
访问者模式	适用于数据结构相对稳定算法又易变化的系统。因为访问者模式使得算法操作增加变得容易。若系统数据结构对象易于变化，经常有新的数据对象增加进来，则不适合使用访问者模式。
中介者模式	用来降低类类之间的耦合的，因为如果类类之间有依赖关系的话，不利于功能的拓展和维护，因为只要修改一个对象，其它关联的对象都得进行修改。如果使用中介者模式，只需关心和 Mediator 类的关系，具体类类之间的关系及调度交给 Mediator 就行
解释器模式	应用在 OOP 开发中的编译器的开发中