一.创建型模式(Creational):

简单工厂模式(simpleFactory)发音:['simpl]['fækt(ə)rɪ]

定义:

提供一个创建对象实例的功能,而无须关心其具体实现.被创建实例的类型可以是接口,抽象类,也可以是具体的类.

1.抽象工厂(AbstractFactory)发音:['æbstrækt]

定义:

提供一个创建一系列相关或相互依赖对象的接口,而无需指定他们具体的类.

使用场合:

1.如果希望一个系统独立于它的产品的创建,组合和表示的时候,换句话书,希望一个系统只是知道产品的接口,而不关心实现的时候.

2.如果一个系统要由多个产品系列中的一个来配置的时候.换句话说,就是可以,就是可以动态地切换产品簇的时候.

3.如果强调一系列相关产品的接口,以便联合使用他们的时候

.

2.建造者模式(Builder)发音: ['bɪldə]

定义:

将复杂对象的构建与它的表示分离,使得同样的构建过程可以创建不同的表示.

使用场合:

1.如果创建对象的算法,应该独立于该对象的组成部分以及它们的装配方式时

2.如果同一个构建过程有着不同的表示时

3.工厂方法模式(Factory Method)

定义:

为创建对象定义一个接口,让子类决定实例化哪个类.工厂方法让一个类的实例化延迟到了子类.

使用场景:

1.客户类不关心使用哪个具体类,只关心该接口所提供的功能.

2.创建过程比较复杂,例如需要初始化其他关联的资源类,读取配置文件等.

3.接口有很多具体实现或者抽象类有很多具体子类时,

4.不希望给客户程序暴露过多的此类的内部结构,隐藏这些细节可以降低耦合度.

5.优化性能,比如缓存大对象或者初始化比较耗时的对象.

4.原型模式(Prototype Method)发音: ['prəʊtətaɪp]

定义:

使用原形实例指定将要创建的对象类型,通过复制这个实例创建新的对象.

应用场合:

1.如果一个系统想要独立于它想要使用的对象时,可以使用原型模式,让系统只面向接口编程,在系统需要新的对象的时候,可以通过克隆原型来得到.

2.如果需要实例化的类是在运行时刻动态指定时,可以使用原型模式,通过克隆原型来得到需要的实例.

5.单例模式(Singleton) 发音:['sɪŋg(ə)lt(ə)n]

定义:

保证一个类仅有一个实例,并提供一个访问它的全局访问点.

使用场合:

当需要控制一个类的实例只能有一个,而且客户只能从一个全局访问点访问它时,可以使用单例模式,这些功能恰好是单例模式要解决的问题.

二.结构型模式(struct)发音:[strʌkt]

6.适配器模式(Adapter)发音:[ə'dæptə]

定义:

将一个类的接口转换成客户希望的另外一个接口.适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作.

使用场合;

1.如果先要使用一个已经存在的类,但是它的接口不符合你的需求,这种情况可以使用适配器模式,来把已有的实现转换成你需要的接口.

2.如果你想创建一个可以复用的类,这个类可能和一些不兼容的类一起工作,这中情况可以使用适配器模式,到时候需要什么就适配什么.

3.如果你想使用一些已经窜在的子类,是不坑对每一个子类都进行适配,这中情况可以使用适配器模式,直接适配这些子类的父类就可以了.

7.桥接模式(Bridge)发音: [brɪdʒ]

定义:

将抽象部分与它的实现部分分离,使他们可以独立变化.

使用场合:

1.如果你不希望在抽象部分和实现部分采用固定的绑定关系,可以采用桥接模式.

2.如果出现抽象部分和实现部分都能够扩展的情况,可以采用桥接模式,让抽象部分和实现部分独立地变化.

3.如果希望实现部分的修改不会对客户产生影响,可以采用桥接模式.

4.如果采用继承的实现方案,会导致产生很多子类,对于这种情况,可以考虑采用桥接模式.

8.组合模式(Composite)发音: ['kɒmpəzɪt]

定义:

将对象组合成属性结构以表示"部分-整体"的层次结构,组合模式使用的用户对单个对象和组合对象的使用具有一致性.

使用场合:

1.如果你想表示对象的部分-整体层次结构,可以使用..把整体和部分的操作统一起来,使得层次结构实现更简单,从外部来使用,这个层次结构也容易.

2.如果希望同意地使用组合结构中的所有对象,可以选用...,这正是组合模式提供的主要功能.

9.装饰器模式(Decorator Method)发音: ['dekəreɪtə]

定义:

动态的给一个对象增加一些额外的职责,就增加功能来说,装饰模式生成子类更为灵活.

使用场合:

1.如果需要爱不影响其他对象的情况下,以动态,透明的方式给对象添加职责,可以使用装饰模式.

2.如果不适合使用子类来进行扩展的时候,可以考虑使用装饰模式.

10.外观模式(Facade)发音: [fə'sɑ:d]

定义:

为子系统中的一组接口提供一个一致的界面,Facade模式定义了一个高层的接口,这个接口使得这一子系统更加同容易使用.

使用场景:

1.如果希望为一个复杂的子系统提供一个简单接口的时候,可以考虑使用外观模式.使用外观对象来实现大部分客户需要的功能,从而简化客户的使用.

2.如果想要让客户程序和抽象类的实现部分松散耦合,可以考虑使用外观模式,使用外观对象来将这个子系统与他的客户分离开来,从而提高子系统的独立性和可移植性.

3.如果构建多层节后的系统,可以考虑使用外观模式使用外观模式对象作为每层的入口,这样可以简化层间调用,也可以松散出层次之间的依赖关系.

11.享元模式(Flyweight)发音: ['flaɪweɪt]

定义:

运用共享技术有效地支持大量细粒度的对象.

使用场合:

1.如果一个应用程序使用了大量的细粒度对象,可以使用享元模式来减少对象的数量.

2.如果犹豫使用大量的对象,造成很大的存储开销,可以使用享元模式来减少对象数量,并节约内存.

3.如果对象的大多数状态都可以转变成外部状态,比如通过计算得到,或者从外部传入等,可以使用享元模式来实现内部状态和外部状态的分离.

4.如果不考虑对象的外部状态,可以用相对较少的共享对象取代很多组合对象,可以使用享元模式来共享对象.然后组合对象来使用这些共享对象.

12.代理模式(Proxy)发音: ['prɒksɪ]

定义:

为其他对象提供一种代理以控制对这个对象的访问.

使用场合:

1.需要为一个对象在不同的地址空间提供局部代表的时候,可以使用远程代理.

2.需要按照需要创建开销很大的对象的时候,可以使用虚代理.

3.需要控制对原始对象的访问的时候,可以使用保护代理.

4.需要在访问你对象执行一些附加操作的时候,可以使用智能指引代理.

三.行为型模式(behavioral)发音[bi'heivjərəl]

13.职责链模式(Chain Of Responsibility)发音: [tʃeɪn][rɪ,spɒnsɪ'bɪlɪtɪ]

定义:

使多个对象都有机会处理请求,,从而避免请求的发送者和接收者之间耦合关系.将这些对象连成一条链,并沿着这条链传递该请求,直到有一个对象处理它为止.

使用场合:

1.如果有多个对象可以处理同一个请求,但是具体由哪个对象来处理该请求,是运行时刻动态确定的.

2.如果你想在不明确指定接收者的情况下,向多个对象中的其中一个提交请求的话,可以使用职责链模式.

3.如果想要动态指定处理一个请求的对象结合,可以使用职责链模式.

14.命令模式(Command)发音: [kə'mɑ:nd]

定义:

将一个请求封装为一个对象,从而使你可用不同的请求对客户进行参数化,对请求排队或者记录请求日志,以及支持可撤销的操作.

15.解释器模式(Interpreter)发音: [ɪn'tɜ:prɪtə]

定义:

给定一个语言,定义它的文法的一种表示,并定义一个解释器,这个解释器使用该表示来解释语言中的句子.

使用场合:

16.迭代器模式(Iterator)

定义:提供一种方法顺序访问一个聚合对象中的各个元素,而又不需要暴露该对象的内部表示.

使用场合:

1.如果你希望提供访问一个聚合对象的内容,但是又不想暴露他的内部表示的时候,可以使用迭代器模式来提供迭代器接口,从而让客户端只是通过迭代器的接口来访问聚合对象,而无须关心聚合对象的内部实现.

2.如果你希望有多种遍历方式可以访问聚合对象,可以使用...

3.如果你希望为遍历不同的聚合对象提供一个统一的接口,可以使用....

17.中介模式(Mediator)发音:['mi:dieitə]

定义:

用一个中介对象类封装一系列对象的交互.中介者使得各对象不需要显式地相互引用,从而使其耦合松散,而且可以独立地改变他们之间的交互.

使用场合:

1.如果一组对象之间的通信方式比较复杂,导致相互依赖,结构混乱,可以采用中介模式,把这些对象相互的交互管理起来,各个对象都只需要和中介者交互,从而是的各个对象松散耦合,结构也更清晰易懂.

2.如果一个对象引用很多的对象,并直接跟这些对象交互,导致难以复用该对象,可以采用中介者模式,把这个对象跟其他对象的交互封装到中介者对象里面,这个对象只需要和中介者对象交互就可了.

18.备忘录模式(Memento)发音: [mɪ'mentəʊ]

在不破坏封装性的前提下,捕获一个对象的内部状态,并在该对象之外保存这个状态.这样以后就可将该对象恢复到原先保存的状态.

使用场合:

1.如果必须要保存一个对象在某一个时刻的全部或者部分状态,方便以后需要的时候,可以把该对象恢复到先前的状态,可以使用备忘录模式.

2.如果需要保存一个对象的内部状态,但是如果用接口来让其他对象直接得到这些需要保存的状态,将会暴露对象的实现希捷并破坏对象的封装性,这是可以使用备忘录.

19.观察者模式(Observer)发音: [əb'zɜ:və]

定义:

定义对象间的一种一对多的依赖关系.当一个对象的状态发生改变时,所有依赖于它的对象都得到通知并被自动更新.

使用场合;

1.当一个抽象模型有两个方面,其中一个方面的操作依赖于另一个方面的状态变化,那么就可以选用观察者模式,将这两者封装成观察者和目标对象,当目标对象变化的时候,依赖于它的观察者对象也会发生相应的变化.这样就把抽象模型的这两个方面分离了使得,它们可以独立地改变和复用.

2.如果在更改一个对象的时候,需要同时连带改变其他对象,而且不知道究竟应该有多少对象需要被连带改变,这种情况可以选用观察者模式,被改的那一个对象很明显就相当于是目标对象,而需要连带修改的对歌其

他对象,就作为多个观察着对象了.

3.当一个对象必须通知其他的对象,但是你又希望这个对象和其他被它通知的对象是松散耦合的,也就是说这个对象其实不详知道具体被通知的对象.这种情况可以选用观察者模式,这个对象就相当于是目标对象,而被它通知的对象就是观察者对象了.

20.状态模式(State)发音: [steɪt]

允许一个对象在其内部状态改变是改变它的行为.对象看起来似乎修改了他的类.

使用场合:

1.如果一个对象的行为取决于它的状态,而且它必须在运行时刻根据状态来改变它的行为,可以使用...来包状态和行为分离开.虽然分离了,但是状态和行为是有对应关系的,可以在运行期间,通过改变状态,就能够调用到该状态对应的状态处理对象上去从而改变对象的行为.

2.如果一个操作中含有庞大的多分枝语句,而且这些分支依赖于该对象的状态,可以使用....把各个分支的处理分散包装到单独的对象处理类中,这样,这些分支对应的对象就可以不依赖于其他对象而独立变化了.

21.策略模式(Strategy)发音: ['strætɪdʒɪ]

定义:

定义一系列的算法,把它们一个个封装起来,并且使他们可以相互替换.本模式使得算法可独立于使用它的客户而变化.

使用场合;

1.出现有许多相关的类,仅仅是行为有差别的情况下,可以使用策略模式来使用多个行为中的一个来配置一个类的方法,实现算法动态切换

2.出现同一算法,有很多不同实现的情况下,可以使用策略模式来把这些"不同的实现"实现成为一个算法的类层次.

3.需要封装算法中,有与算法相关数据的情况下,可以使用策略模式来避免暴露这些跟算法相关的数据结构.

4.出现抽象一个定义了很多行为的类,并且是通过多个if-else语句来选择这些行为的情况下,可以使用策略模式来替换这些条件语句.

22.模版方法模式(Template Method)发音:['templeɪt;-plɪt]

定义:

定义在一个操作中的算法框架,把一些步骤推迟到子类去实现.模版方法模式让子类不需要改变算法的结构而重新定义特定的算法步骤

功能:

1.能够解决代码的冗余问题

2.把某些算法步骤延迟到子类

3.易于扩展

4.父类提供了算法框架,控制了算法的执行流程,而子类不能改变算法的流程,子类的方法的调用由父类的模版方法决定.

5.父类可以把那些重要的,不允许改变的方法屏蔽掉,不让子类去复写他们.

1.需要固定定义算法骨架,实现一个算法的不变的部分,并把可变的行为留给子类来实现的情况.

2.各个子类中具有公共行为,应该抽取出来,集中在一个公共类中去实现,从而避免复杂的代码重复

3.需要控制子类扩展的情况.模版方法模式会在特定的点来调用子类的方法,这样只允许在这些点进行扩展.

知识:

回调:表示一段可执行逻辑的引用(或者指针),我们把该引用(或者指针)传递到另外一段逻辑(或者方法)里供这段逻辑适时调用

(网站:redhat.iteye.com)

23.访问者模式(Visitor)发音:['vɪzɪtə]

定义:

表示一个作用于某对象结构中的各个元素的操作.它使你可以在不改变各元素的类的前提下定义作用于这些元素的新操作.

使用场合:

1.如果想对一个对象结构实施一些依赖于对象结构中具体类的操作,可以使用访问者模式.

2.如果想对一个对象结构中的各个元素进行很多不同的而且不相关的操作,为了避免这些操作使类变得杂乱,可以使用访问者模式.

3.如果对象结构很少变动,但是需要经常给对象结构中的元素定义新的操作,可以使用访问者模式.

3.如果对象结构很少变动,但是需要经常给对象结构中的元素定义新的操作,可以使用访问者模式.