云端观云

海纳百川，有容乃大；壁立千仞，无欲则刚；

十种常用的设计模式

转自 CSDN： https://blog.csdn.net/wymrdjm/article/details/78927139

1. 单例模式：

实现方式：

a） 将被实现的类的构造方法设计成private的。

b） 添加此类引用的静态成员变量，并为其实例化。

c） 在被实现的类中提供公共的CreateInstance函数，返回实例化的此类,就是b中的静态成员变量。

应用场景：

优点：
1.在单例模式中，活动的单例只有一个实例，对单例类的所有实例化得到的都是相同的一个实例。这样就 防止其它对象对自己的实例化，确保所有的对象都访问一个实例
2.单例模式具有一定的伸缩性，类自己来控制实例化进程，类就在改变实例化进程上有相应的伸缩性。
3.提供了对唯一实例的受控访问。
4.由于在系统内存中只存在一个对象，因此可以 节约系统资源，当 需要频繁创建和销毁的对象时单例模式无疑可以提高系统的性能。
5.允许可变数目的实例。
6.避免对共享资源的多重占用。
缺点：
1.不适用于变化的对象，如果同一类型的对象总是要在不同的用例场景发生变化，单例就会引起数据的错误，不能保存彼此的状态。
2.由于单利模式中没有抽象层，因此单例类的扩展有很大的困难。
3.单例类的职责过重，在一定程度上违背了“单一职责原则”。
4.滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致共享连接池对象的程序过多而出现连接池溢出；如果实例化的对象长时间不被利用，系统会认为是垃圾而被回收，这将导致对象状态的丢失。
使用注意事项：
1.使用时不能用反射模式创建单例，否则会实例化一个新的对象
2.使用懒单例模式时注意线程安全问题
3.单例模式和懒单例模式构造方法都是私有的，因而是不能被继承的，有些单例模式可以被继承（如登记式模式）
适用场景：
单例模式只允许创建一个对象，因此节省内存，加快对象访问速度，因此对象需要被公用的场合适合使用，如多个模块使用同一个数据源连接对象等等。如：
1.需要频繁实例化然后销毁的对象。
2.创建对象时耗时过多或者耗资源过多，但又经常用到的对象。
3.有状态的工具类对象。
4.频繁访问数据库或文件的对象。
以下都是单例模式的经典使用场景：
1.资源共享的情况下，避免由于资源操作时导致的性能或损耗等。如上述中的日志文件，应用配置。

2018年7月 (10)
2018年6月 (2)
2018年5月 (7)
2018年4月 (6)
2018年3月 (10)
2018年2月 (9)
2018年1月 (8)
2017年12月 (5)
2017年11月 (12)
2017年10月 (8)
2017年9月 (2)
2017年8月 (4)
2017年7月 (3)
2017年6月 (6)
2017年3月 (3)

阅读排行榜

1. 十种常用的设计模式(40203)
2. 五种js判断是否为整数（转）(10822)
3. 阿里云服务器配置SSL证书成功开启Https（记录趟过的各种坑）(7958)
4. css给div添加阴影效果(7300)
5. spring集成jwt验证方式，token验证(7239)

推荐排行榜

1. 在ssm框架中前后台数据交互均使用json格式(1)
2. Mybatis动态查询语句(1)
3. Springboot+MyBatis+JPA集成(1)
4. 一位不认识的同行的学习思想，触动很大(1)
5. 阿里云服务器Tomcat无法从外部访问(1)

2.控制资源的情况下，方便资源之间的互相通信。如线程池等。

应用场景举例：

1.外部资源：每台计算机有若干个打印机，但只能有一个PrinterSpooler，以避免两个打印作业同时输出到打印机。内部资源：大多数软件都有一个（或多个）属性文件存放系统配置，这样的系统应该有一个对象管理这些属性文件

2. Windows的TaskManager（任务管理器）就是很典型的单例模式（这个很熟悉吧），想想看，是不是呢，你能打开两个windows task manager吗？ 不信你自己试试看哦~

3. windows的Recycle Bin（回收站）也是典型的单例应用。在整个系统运行过程中，回收站一直维护着仅有的一个实例。

4. 网站的计数器，一般也是采用单例模式实现，否则难以同步。

5. 应用程序的日志应用，一般都何用单例模式实现，这一般是由于共享的日志文件一直处于打开状态，因为只能有一个实例去操作，否则内容不好追加。

6. Web应用的配置对象的读取，一般也应用单例模式，这个是由于配置文件是共享的资源。

7. 数据库连接池的设计一般也是采用单例模式，因为数据库连接是一种数据库资源。数据库软件系统中使用数据库连接池，主要是节省打开或者关闭数据库连接所引起的效率损耗，这种效率上的损耗还是非常昂贵的，因为何用单例模式来维护，就可以大大降低这种损耗。

8. 多线程的线程池的设计一般也是采用单例模式，这是由于线程池要方便对池中的线程进行控制。

9. 操作系统的文件系统，也是大的单例模式实现的具体例子，一个操作系统只能有一个文件系统。

10. HttpApplication 也是单位例的典型应用。熟悉ASP.Net(IIS)的整个请求生命周期的人应该知道HttpApplication也是单例模式，所有的HttpModule都共享一个HttpApplication实例.

2. 策略模式：

实现方式：

a) 提供公共接口或抽象类，定义需要使用的策略方法。（策略抽象类）

b) 多个实现的策略抽象类的实现类。（策略实现类）

c) 环境类，对多个实现类的封装，提供接口类型的成员量，可以在客户端中切换。

d) 客户端 调用环境类 进行不同策略的切换。

注：Jdk中的TreeSet和 TreeMap的排序功能就是使用了策略模式。

策略模式的优点

（1）策略模式提供了管理相关的算法族的办法。策略类的等级结构定义了一个算法或行为族。恰当使用继承可以把公共的代码移到父类里面，从而避免代码重复。

（2）使用策略模式可以避免使用多重条件(if-else)语句。多重条件语句不易维护，它把采取哪一种算法或采取哪一种行为的逻辑与算法或行为的逻辑混合在一起，统统列在一个多重条件语句里面，比使用继承的办法还要原始和落后。

策略模式的缺点

（1）客户端必须知道所有的策略类，并自行决定使用哪一个策略类。这就意味着客户端必须理解这些算法的区别，以便适时选择恰当的算法类。换言之，策略模式只适用于客户端知道算法或行为的情况。

（2）由于策略模式把每个具体的策略实现都单独封装成为类，如果备选的策略很多的话，那么对象的数目就会很可观。

3. 代理模式：

一）静态代理

实现方式：

a） 为真实类和代理类提供的公共接口或抽象类。（租房）

b） 真实类，具体实现逻辑，实现或继承a。（房主向外租房）

c） 代理类，实现或继承a，有对b的引用，调用真实类的具体实现。（中介）

d） 客户端，调用代理类实现对真实类的调用。（租客租房）

二）动态代理

实现方式：

a） 公共的接口（必须是接口，因为Proxy类的newproxyinstance方法的第二参数必须是个接口类型的Class）

b） 多个真实类，具体实现的业务逻辑。

c） 代理类，实现InvocationHandler接口，提供Object成员变量，和Set方法，便于客户端切换。

d） 客户端，获得代理类的实例，为object实例赋值，调用Proxy.newproxyinstance方法在程序运行时生成继承公共接口的实例，调用相应方法，此时方法的执行由代理类实现的Invoke方法接管。

jdk动态代理使用的局限性：
通过反射类Proxy和InvocationHandler回调接口实现的jdk动态代理，要求委托类必须实现一个接口，但事实上并不是所有类都有接口，对于没有实现接口的类，便无法使用该方式实现动态代理。

4. 观察者模式：

观察者模式是对象的行为模式，又叫发布-订阅(Publish/Subscribe)模式、模型-视图(Model/View)模式、源-监听器(Source/Listener)模式或从属者(Dependents)模式。

实现方式：

a） 角色抽象类（提供对观察者的添加，删除和通知功能）。

b） 角色具体类，实现a，维护一个c的集合（对角色抽象类的实现）。

c） 观察者抽象类（被角色通知后实现的方法）。

d） 观察者实现类，实现c（多个）。

注：JDK提供了对观察者模式的支持，使用Observable类和Observer接口

两种模型（推模型和拉模型）：

■ 推模型是假定主题对象知道观察者需要的数据；而拉模型是主题对象不知道观察者具体需要什么数据，没有办法的情况下，干脆把自身传递给观察者，让观察者自己去按需要取值。

■ 推模型可能会使得观察者对象难以复用，因为观察者的update()方法是按需要定义的参数，可能无法兼顾没有考虑到的使用情况。这就意味着出现新情况的时候，就可能提供新的update()方法，或者是干脆重新实现观察者；而拉模型就不会造成这样的情况，因为拉模型下，update()方法的参数是主题对象本身，这基本上是主题对象能传递的最大数据集合了，基本上可以适应各种情况的需要。

5. 装饰模式:

实现方式：

a) 抽象的被装饰角色 （所有的角色都要直接或间接的实现本角色）

b) 具体的被装饰角色，实现或继承a （被功能扩展的角色）

c) 装饰角色，实现或继承a （本类有对a的引用，所有的具体装饰角色都需要继承这个角色）

d) 多个具体修饰角色 ，继承c（对被装饰角色的功能扩展，可以任意搭配使用）

意图：

动态地给一个对象添加一些额外的职责。就增加功能来说，Decorator模式相比生成子类更为灵活。该模式以对客 户端透明的方式扩展对象的功能。

适用环境:

（1）在不影响其他对象的情况下，以动态、透明的方式给单个对象添加职责。

（2）处理那些可以撤消的职责。

（3）当不能采用生成子类的方法进行扩充时。一种情况是，可能有大量独立的扩展，为支持每一种组合将产生大量的 子类，使得子类数目呈爆炸性增长。另一种情况可能是因为类定义被隐藏，或类定义不能用于生成子类。

6. 适配器模式:

适配器模式把一个类的接口变换成客户端所期待的另一种接口，从而使原本因接口不匹配而无法在一起工作的两个类能够在一起工作。

1. 类适配器（子类继承方式）

实现方式:

a) 目标抽象角色（定义客户要用的接口）

b) 适配器（实现a继承c，作为一个转换器被客户调用）

c) 待适配器（真正需要被调用的）

d) 客户端（借用a的实例调用c的方法）

2. 对象适配器（对象的组合方式）

实现方式:

a) 目标抽象角色（定义客户要用的接口）

b) 适配器（实现a，维护一个c的引用，作为一个转换器被d调用）

c) 待适配器（真正需要被调用的）

d) 客户端（此类，借用a类的实例调用c类的方法，类似静态代理，但是解决的问题不同）

3. 缺省的方式

实现方式:

a) 抽象接口

b) 实现a的适配器类（空实现）

c) 客户端，继承b，调用b中的方法，不必直接实现a（直接实现a需要实现a中的所有的方法）

适配器模式的优点:

1. 更好的复用性

系统需要使用现有的类，而此类的接口不符合系统的需要。那么通过适配器模式就可以让这些功能得到更好的复用。

2. 更好的扩展性

在实现适配器功能的时候，可以调用自己开发的功能，从而自然地扩展系统的功能。

适配器模式的缺点:

过多的使用适配器，会让系统非常零乱，不易整体进行把握。比如，明明看到调用的是A接口，其实内部被适配成了B接口的实现，一个系统如果太多出现这种情况，无异于一场灾难。因此如果不是很有必要，可以不使用适配器，而是直接对系统进行重构。

7. 命令模式

将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化；对请求排队或记录日志，以及支持可撤销的操作

将“发出请求的对象”和”接收与执行这些请求的对象”分隔开来。

实现方式：

a) 抽象的命令角色， 如：菜单（规定可以点哪些菜）

b) 具体的命令角色（实现a 维护一个对c的引用），如：订单（已点的菜）

c) 接收者（具体执行命令的角色），实际操作时，很常见使用"聪明"命令对象，也就是直接实现了请求，而不是将工作委托给c (弊端？) 如：厨师接收订单后做菜

d) 调用者（维护一个对a的引用），如：服务员负责点菜并把订单推给厨师

e) 客户端 调用d发出命令进而执行c的方法，如：顾客点餐

效果：
1)、command模式将调用操作的对象和实现该操作的对象解耦
2)、可以将多个命令装配成一个复合命令，复合命令是Composite模式的一个实例
3)、增加新的command很容易，无需改变已有的类
适用性：
1)、抽象出待执行的动作以参数化某对象
2)、在不同的时刻指定、排列和执行请求。如请求队列
3)、支持取消操作
4)、支持修改日志
5)、用构建在原语操作上的高层操作构造一个系统。支持事物

8. 组合模式

将对象组合成树形结构以表示“部分整体”的层次结构。组合模式使得用户对单个对象和复杂对象的使用具有一致性。

实现方式：

a) 抽象的构件接口 (规范执行的方法)，b及c都需实现此接口，如：Junit中的Test接口

b) 叶部件（实现a，最小的执行单位），如：Junit中我们所编写的测试用例

c) 组合类（实现a并维护一个a的集合[多个b的组合]），如：Junit中的 TestSuite

d) 客户端 可以随意的将b和c进行组合，进行调用

什么情况下使用组合模式：

当发现需求中是体现部分与整体层次结构时，以及你希望用户可以忽略组合对象与单个对象的不同，统一地使用组合结构中的所有对象时，就应该考虑组合模式了。

9. 简单工厂模式

就是建立一个工厂类，对实现了同一接口的一些类进行实例的创建。简单工厂模式的实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类（这些产品类继承自一个父类或接口）的实例。

实现方式:

a) 抽象产品类（也可以是接口）

b) 多个具体的产品类

c) 工厂类（包括创建a的实例的方法）

优点：

工厂类是整个模式的关键.包含了必要的逻辑判断,根据外界给定的信息,决定究竟应该创建哪个具体类的对象.通过使用工厂类,外界可以从直接创建具体产品对象的尴尬局面摆脱出来,仅仅需要负责“消费”对象就可以了。而不必管这些对象究竟如何创建及如何组织的．明确了各自的职责和权利，有利于整个软件体系结构的优化。

缺点：

由于工厂类集中了所有实例的创建逻辑，违反了高内聚责任分配原则，将全部创建逻辑集中到了一个工厂类中；它所能创建的类只能是事先考虑到的，如果需要添加新的类，则就需要改变工厂类了。当系统中的具体产品类不断增多时候，可能会出现要求工厂类根据不同条件创建不同实例的需求．这种对条件的判断和对具体产品类型的判断交错在一起，很难避免模块功能的蔓延，对系统的维护和扩展非常不利；

10. 模板方法模式

实现方式：

a) 父类模板类（规定要执行的方法和顺序，只关心方法的定义及顺序，不关心方法实现）

b) 子类实现类（实现a规定要执行的方法，只关心方法实现，不关心调用顺序）

优点：

1）封装不变部分，扩展可变部分：把认为不变部分的算法封装到父类实现，可变部分则可以通过继承来实现，很容易扩展。

2）提取公共部分代码，便于维护。

3）行为由父类控制，由子类实现。

缺点：

模板方法模式颠倒了我们平常的设计习惯：抽象类负责声明最抽象、最一般的事物属性和方法，实现类实现具体的事物属性和方法。在复杂的项目中可能会带来代码阅读的难度。

分类: java设计模式

云端观云
关注 - 9
粉丝 - 11
+加关注

« 上一篇：问答排序算法
» 下一篇：EurekaClient项目启动报错Invocation of destroy method failed on bean with name 'scopedTarget.eurekaClient': org.springframework.beans.factory.BeanCreationNotAllowedException: Error creating bean with name 'e

posted @ 2018-09-04 15:06 云端观云 阅读(40205) 评论(0) 编辑 收藏

相关博文：

· Java中的24种设计模式与7大原则
· Java中的23种设计模式与7大原则
· 设计模式总结
· [java笔记]常用的设计模式
· php设计模式总结2

最新新闻：

· 开启新一轮降价，AWS 加速在华布局
· 钉钉并入阿里云，双方要什么「乘数效应」？
· AI正从“感知智能”走向“认知智能”
· 苏宁易购收购家乐福中国80%股权，拟出资48亿元
· 造假AI又进化！只要一张照片，说话唱歌视频自动生成丨已开源

» 更多新闻...