咕泡教育
—gupaoedu—

Java架构师VIP第六期第1次直播答疑

趣谈常用的经典设计模式及重难点梳理

Tom

我们的愿景
推动每一次人才升级

我们的使命
让每个人的职业生涯不留遗憾

1 七大设计原则

七大设计原则

开闭原则(OCP)
Open-Close Principle

依赖倒置原则(DIP)
Dependence Inversion Principle

单一职责原则(SRP)
Simple Responsibility Principle

接口隔离原则(ISP)
Interface Segregation Principle

迪米特法则(LoD)
Law of Demeter

里氏替换原则(LSP)
Liskov Substitution Principle

合成复用原则(CARP)
Composite/Aggregate Reuse Principle

2 单例模式

单例模式的定义

官方原文：

```
Ensure a class has only one instance,and provide a global point of access to it.
```

大致意思是，确保一个类在任何情况下都绝对只有一个实例，并提供一个全局访问点。

哪些情况下的单例对象可能会破坏?

面试题：哪些情况下的单例对象可能会被破坏

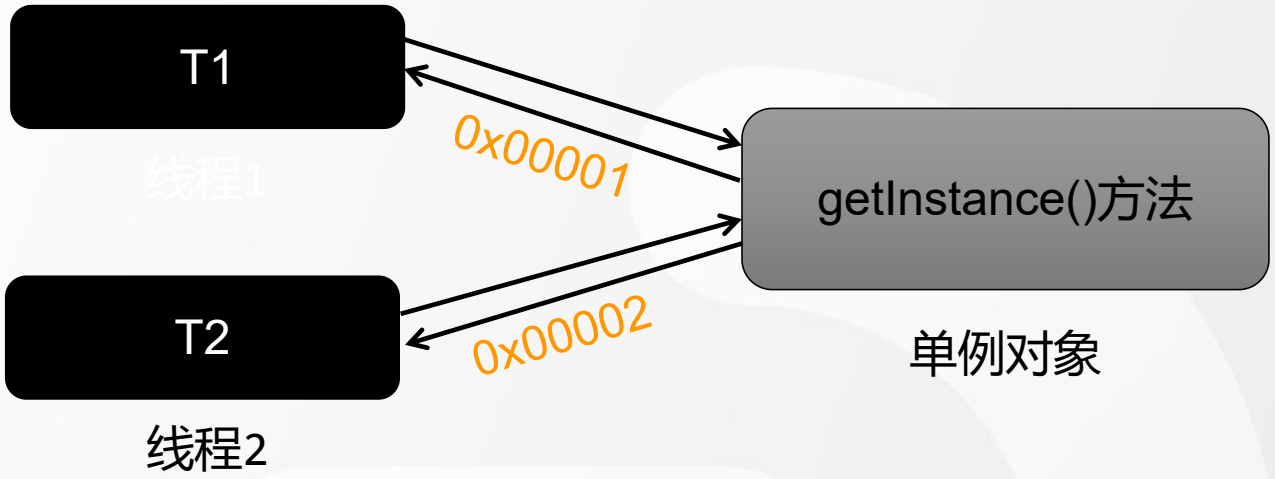
可能出现单例被破坏的情况：

1. 多线程破坏单例
2. 指令重排破坏单例
3. 克隆破坏单例
4. 反序列化破坏单例
5. 反射破坏单例

面试题：哪些情况下的单例对象可能会被破坏

1 多线程破坏单例

问题： 多个线程同时操作，导致同时创建多个对象



解决方案：

1. 改为DCL双重检查锁的写法
2. 使用静态内部类的写法，性能更高

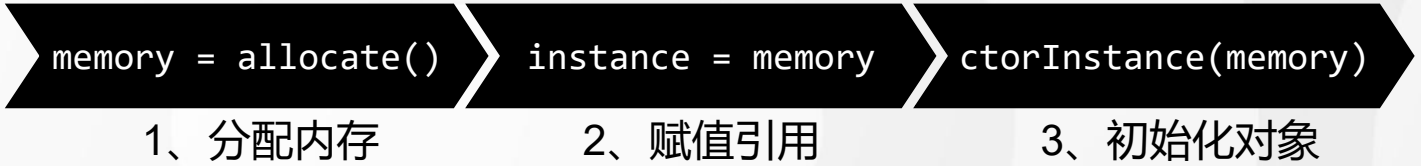
面试题：哪些情况下的单例对象可能会被破坏

问题： JVM指令重排可能导致懒汉式单例被破坏

```
instance = new Singleton()
```

执行指令：

指令重排：

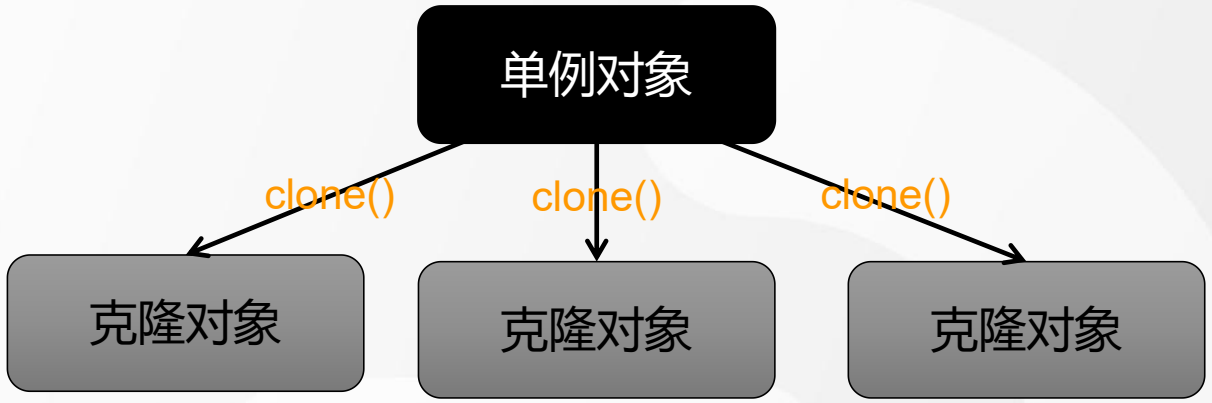


解决方案：

```
1    private static volatile Singleton instance = null;
```

面试题：哪些情况下的单例对象可能会被破坏

问题： 深clone()，每次都会重新创建新的实例

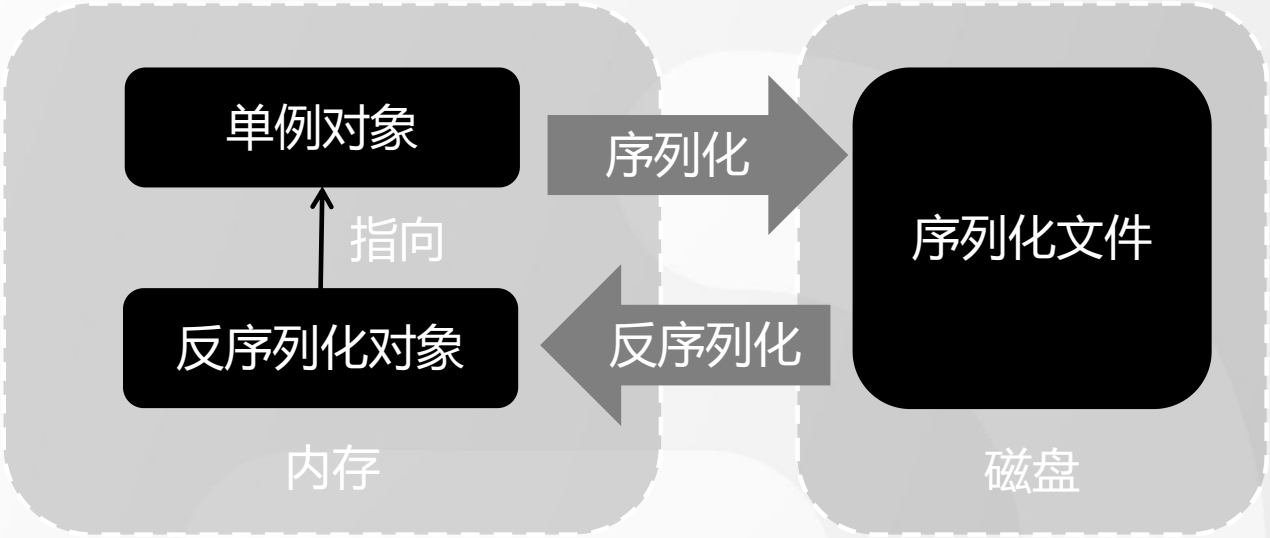


解决方案：

1 在单例对象中重写clone() 方法

面试题：哪些情况下的单例对象可能会被破坏

问题： 反序列化对象会重新分配内存，相当于重新创建对象

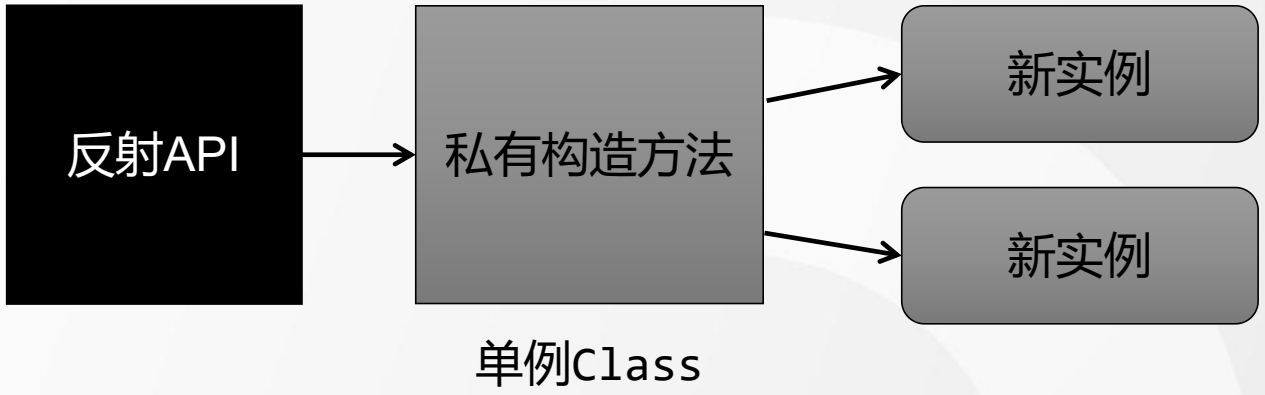


解决方案：

1. 需要重写readResolve()方法，将返回值设置为单例对象

面试题：哪些情况下的单例对象可能会被破坏

问题： 反射可以任意调用私有构造方法创建单例对象



解决方案：

1. 在构造方法中检查单例对象，如果已创建则抛出异常
2. 将单例的实现方式改为枚举式单例

面试题：哪些情况下的单例对象可能会被破坏

总结：

1. 如果程序不是太复杂，单例对象又不多，推荐使用饿汉式单例
2. 如果经常发生多线程并发情况，推荐使用静态内部类和枚举式单例

3 原型模式

原型模式的定义

官方原文：

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

大致意思是，指原型实例指定创建对象的种类，并且通过复制这些原型创建新的对象

谈谈你对深克隆和浅克隆的理解

面试题：谈谈你对深克隆和浅克隆的理解

深克隆和浅克隆的本质区别：

1. 数据拷贝后两者之间是否有关联
2. 改变一个值是否会影响到另一个数值变化

面试题：谈谈你对深克隆和浅克隆的理解

浅克隆（Shallow Clone）

面试题：谈谈你对深克隆和浅克隆的理解

浅克隆常用的API

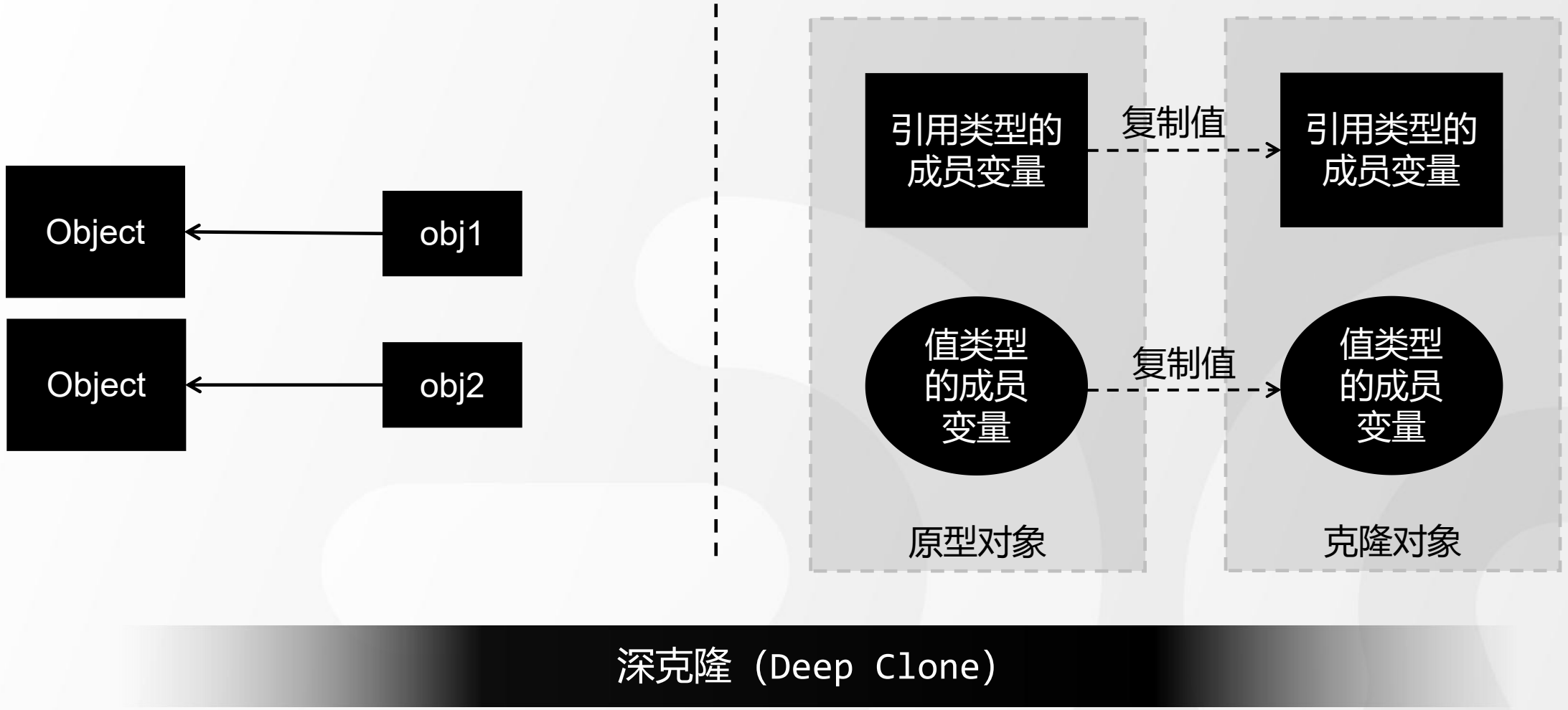
1. 工具类BeanUtils和PropertyUtils进行对象复制

	Spring的BeanUtils	commons的BeanUtils
方法	copyProperty和copyProperties	copyProperties
参数	src ， dest	dest，src

2. 实现Clonenable接口

3. Arrays.copyOf()，但在ArrayList中实现了深克隆的效果

面试题：谈谈你对深克隆和浅克隆的理解



深克隆（Deep Clone）

面试题：谈谈你对深克隆和浅克隆的理解

深克隆常用的API

1. 每个对象都要实现Clonenable接口并重写Object类中的clone()方法
2. 序列化，必须实现 Serializable 接口
3. Apache Commons工具包SerializationUtils.clone(T object);
4. 通过 JSON 工具类实现深克隆
5. 通过构造方法实现深克隆（手动new对象）

4 建造者模式

建造者模式的定义

官方原文：

Separate the construction of a complex object from its representation so that the same construction process can create different representations.

大致意思是，将一个复杂对象的构建过程与它的表示分离，使得同样的构建过程可以创建不同的表示。

链式编程
是建造者模式的标配吗?

实现零件无序装配

5 代理模式

代理模式的定义

官方原文：

```
Provide a surrogate or placeholder for another object to control access to it.
```

大致意思是，指为其他对象提供一种代理，以控制对这个对象的访问

你如何理解静态代理和动态代理

面试题：你如何理解静态代理和动态代理

什么是代理

代理对象在客服端和目标对象之间起到中介作用

保护目标对象

增强目标对象

面试题：你如何理解静态代理和动态代理

静态代理和动态代理

1. 一个代理只能服务于一种类型的对象，当有n个业务时，需要n个静态代理，不利于业务的扩展。
2. 一个代理类可以服务于所有的业务对象。

面试题：你如何理解静态代理和动态代理

动态代理的基本实现

1. 拿到被代理类的引用，并且获取它的所有的接口（反射获取）
2. JDK Proxy类重新生成一个新的类，实现了被代理类所有接口的方法
3. 动态生成Java代码，把增强逻辑加入到新生成代码中
4. 编译生成新的Java代码的class文件
5. 加载并重新运行新的class，得到类就是全新类

面试题：你如何理解静态代理和动态代理

CGLib和JDK动态代理对比

1. JDK动态代理是实现了被代理对象的接口，CGLib是继承了被代理对象
2. JDK和CGLib都是在运行期生成字节码
3. JDK调用代理方法，是通过反射机制调用，CGLib是通过FastClass机制

CGLib无法代理final修饰的方法

6 策略模式

策略模式的定义

官方原文：

```
Define a family of algorithms, encapsulate each one, and make them interchangeable.
```

大致意思是，将定义的算法家族分别封装起来，让它们之间可以互相替换，从而让算法的变化不会影响到使用算法的用户

什么场景下应该用策略模式
什么场景下不该用?

面试题：什么场景下应该用策略模式，什么场景下不该用？

适合使用的场景

1 需要经常自由切换执行逻辑和规则的场景

面试题：什么场景下应该用策略模式，什么场景下不该用？

不适合使用的场景

1 如果两种逻辑之间关联性本来就比较大，而且变化也比较快

7 责任链模式

责任链模式的定义

官方原文：

Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

将链中每一个节点都看作一个对象，每个节点处理的请求均不同，且内部自动维护下一个节点对象。当一个请求从链式的首端发出时，会沿着责任链预设的路径依次传递到每一个节点对象，直至被链中的某个对象处理为止。

将处理不同逻辑的对象连接成一个链表结构，每个对象都保存下一个节点的引用

责任链模式的实现原理

面试题：责任链模式的实现原理

工作生活中的审批流程

过五关、斩六将

被戏称为“踢皮球”模式

面试题：责任链模式的实现原理

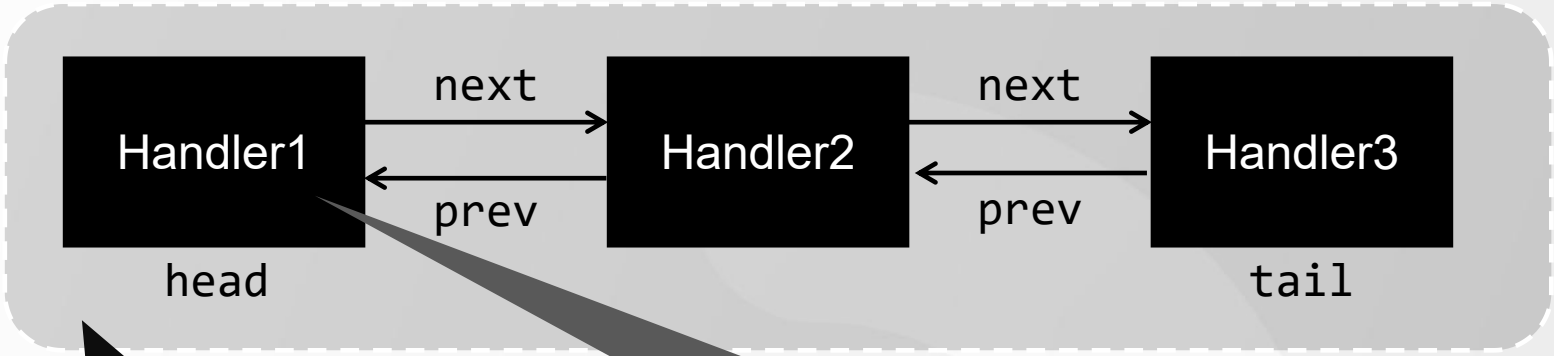
单向链表上下文


```
public class Context {
    Handler head;
    Handler tail;
}
```

```
public abstract class Handler {
    protected Handler next;
}
```

面试题：责任链模式的实现原理

双向链表上下文




```
public class Context {
    Handler head;
    Handler tail;
}
```

```
public abstract class Handler {
    protected Handler prev;
    protected Handler next;
}
```

面试题：责任链模式的实现原理

优点：

1. 将请求与处理解耦
2. 请求处理者将不是自己职责范围内的请求，转发给下一个节点。
3. 请求发送者不需要关系链路结构，只需等待请求处理结果即可。
4. 链路结构灵活，易于扩展新的请求处理节点

面试题：责任链模式的实现原理

缺点：

1. 责任链太长或者处理时间过长，会影响整体性能。
2. 如果节点对象存在循环引用，则会造成死循环，导致程序崩溃。

8 各设计模式的对比

GoF设计模式分类

创建型
Creational

对类的实例化进行了抽象，能够使软件模块做到与对象的创建和代码组织无关。

类的创建

结构型
Structural

描述类和对象之间如何进行有效的组织，以形成良好的软件体系结构，主要的方式是使用继承关系来组织各个类。

组合代替、类与类之间的关系

行为型
Behavioral

描述 类和对象之间如何交互以及如何分配职责。

关注对象与行为的分离、就是要把行为分离到类里面

创建型(Creational)设计模式使用频率总结

工厂方法模式
Factory Method

抽象工厂模式
Abstract Factory

建造者模式
Builder

单例模式
Singleton

原型模式
Prototype

结构型(Creational)设计模式使用频率总结

适配器模式
Adapter

装饰器模式
Decorator

代理模式
Proxy

门面模式
Facade

组合模式
Composite

享元模式
Flyweight

桥接模式
Bridge

咕泡教育
—gupaoedu—

行为型(Behavioral)设计模式使用频率总结

策略模式
Strategy

模板方法模式Template
Method

访问者模式
Visitor

观察者模式
Observer

迭代器模式
Iterator

备忘录模式
Memento

责任链模式
Chain of Responsibility

中介者模式
Mediator

状态模式
State

解释器模式
Interpreter

命令模式
Command

一句话归纳创建型设计模式

设计模式	一句话归纳	目的	生活案例	框架源码举例
工厂模式（Factory）	产品标准化，生产更高效	封装创建细节	实体工厂	LoggerFactory、BeanFactory
单例模式（Singleton）	世上只有一个Tom	保证独一无二	CEO	Calender、Runtime
原型模式（Prototype）	拔一根猴毛，吹出千万个	高效创建对象	克隆	ArrayList、PrototypeBean
建造者模式（Builder）	高配中配与低配，想选哪配就哪配	开放个性配置步骤	选配	StringBuilder、BeanDefinitionBuilder

一句话归纳结构型设计模式

设计模式	一句话归纳	目的	生活案例	框架源码举例
代理模式（Proxy）	没有资源没时间，得找媒婆来帮忙	增强职责	媒婆	ProxyFactoryBean、JdkDynamicAopProxy、CglibAopProxy
门面模式(Facade)	打开一扇门，走向全世界	统一访问入口	前台	JdbcUtils、RequestFacade
装饰器模式（Decorator）	他大舅他二舅，都是他舅	灵活扩展、同宗同源	煎饼	BufferedReader、InputStream
享元模式(Flyweight)	优化资源配置，减少重复浪费	共享资源池	全国社保联网	String、Integer、ObjectPool
组合模式(Composite)	人在一起叫团伙，心在一起叫团队	统一整体和个体	组织架构树	HashMap、SqlNode
适配器模式（Adapter）	适合自己的，才是最好的	兼容转换(求同存异)	电源适配	AdvisorAdapter、HandlerAdapter
桥接模式(Bridge)	约定优于配置	不允许用继承	桥	DriverManager

一句话归纳行为型设计模式

咕泡教育 gupaoedu

设计模式	一句话归纳	目的	生活案例	框架源码举例
委派模式（Delegate）	这个需求很简单，怎么实现我不管	只对结果负责	授权委托书	ClassLoader、BeanDefinitionParserDelegate
模板模式（Template）	流程全部标准化，需要微调请覆盖	逻辑复用	把大象装进冰箱的步骤	JdbcTemplate、HttpServlet
策略模式（Strategy）	条条大道通北京，具体哪条你来定	把选择权交给用户	选择支付方式	Comparator、InstantiationStrategy
责任链模式(Chain of Responsibility)	各人自扫门前雪，莫管他人瓦上霜	解耦处理逻辑	踢皮球	FilterChain、Pipeline
迭代器模式(Iterator)	流水线上坐一天，每个包裹扫一遍	统一对集合的访问方式	统一刷脸进站	Iterator
命令模式(Command)	运筹帷幄之中 决胜千里之外	解耦请求和处理	遥控器	Runnable、TestCase
状态模式(State)	状态驱动行为，行为决定状态	绑定状态和行为	订单状态跟踪	Lifecycle
备忘录(Memento)	给我一剂“后悔药”	备份	草稿箱	StateManageableMessageContext
中介者(Mediator)	联系方式我给你，怎么搞定我不管	统一管理网状资源	朋友圈	Timer
解释器模式（Interpreter）	我想说“方言” 一切解释权归我所有	实现特定语法解析	摩斯密码	Pattern、ExpressionParser
观察者模式（Observer）	到点就通知我	解耦观察者与被观察者	闹钟	ContextLoaderListener
访问者模式（Visitor）	横看成岭侧成峰，远近高低各不同	解耦数据结构和数据操作	KPI考核	FileVisitor、BeanDefinitionVisitor

咕泡教育
—gupaoedu—

谢谢

GUPAO EDU

我们的愿景

推动每一次人才升级

我们的使命

让每个人的职业生涯不留遗憾

请在此处输入文字

请在此处插入二维码