牛逼！这个设计模式一战封神！！！

转载

欢迎关注公众号：【码农突围】 于 2020-11-16 09:47:00 发布 82 收藏 1
文章标签： 编程语言 navicat nagios etag payment

最近很多小伙伴找我要一些 **设计模式** 基础资料，于是我翻箱倒柜，把这份字节跳动大牛总结的设计模式归纳笔记找出来，免费共享给大家！

据说有小伙伴靠这份笔记顺利进入 BAT 哦，所以一定要好好学习这份资料！

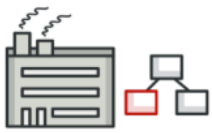
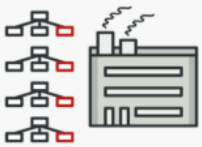
资料介绍

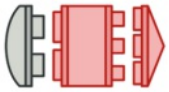
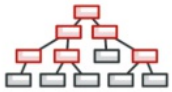
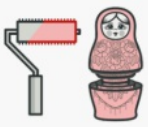
这份资料非常全面且详细，覆盖了 **设计模式** 基础学习的方方面面，非常适合初学者入门！

资料也按目录进行编排，每一章下面都有更具体的内容：


创建者模式(5节)
第 1 节：工厂方法模式
一、开发环境
二、工厂方法模式介绍
三、模拟发奖多种商品
四、用一坨坨代码实现
五、工厂模式优化代码
六、总结
第 2 节：抽象工厂模式
第 3 节：建造者模式
第 4 节：原型模式
第 5 节：单例模式

有趣的插图：

序号	类型	图稿	业务场景	实现要点
1	工厂方法		多种类型商品不同接口，统一发奖服务搭建场景	定义一个创建对象的接口，让其子类自己决定实例化哪一个工厂类，工厂模式使其创建过程延迟到子类进行。
2	抽象工厂		替换Redis双集群升级，代理类抽象场景	提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。
3	建造者		各项装修物料组合套餐选配场景	将一个复杂的构建与其表示相分离，使得同样的构建过程可以创建不同的表示。

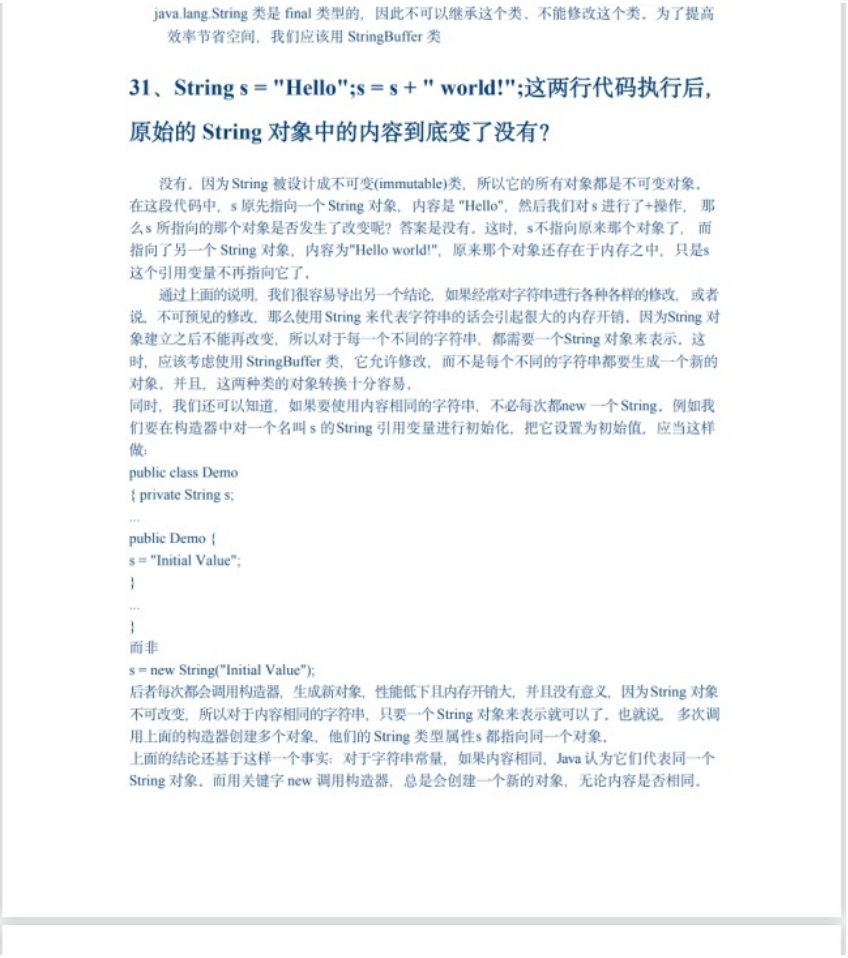
序号	类型	图稿	业务场景	实现要点
1	适配器		从多个MQ消息体中，抽取指定字段值场景	将一个类的接口转换成客户希望的另外一个接口。适配器模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。
2	桥接		多支付渠道(微信、支付宝)与多支付模式(刷脸、指纹)场景	将抽象部分与实现部分分离，使它们都可以独立的变化。
3	组合		营销差异化人群发券，决策树引擎搭建场景	将对象组合成树形结构以表示"部分-整体"的层次结构。组合模式使得用户对单个对象和组合对象的使用具有一致性。
4	装饰		SSO单点登录功能扩展，增加拦截用户访问方法范围场景	动态地给一个对象添加一些额外的职责。就增加功能来说，装饰器模式相比生成子类更为灵活。

资料获取

1. 识别并关注下方公众号「**Github爱好者社区**」；
2. 在下面公众号后台回复关键字「**设计模式**」即可下载。

另外，之前还收藏过一套Java面试题，也附送给大家。

Java基础83题、算法相关12题、JavaWeb部分20题、数据库部分30题、流行框架与新技术20题、还有软件工程、设计模式、J2EE等部分知识。题目有200多道，共146页。


java.lang.String 类是 final 类型的，因此不可以继承这个类、不能修改这个类。为了提高效率节省空间，我们应该用 StringBuffer 类

31、String s = "Hello";s = s + " world!";这两行代码执行后，原始的 String 对象中的内容到底变了没有?

没有。因为 String 被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。在这段代码中，s 原先指向一个 String 对象，内容是 "Hello"，然后我们对 s 进行了+操作，那么 s 所指向的那个对象是否发生了改变呢？答案是没有。这时，s 不指向原来那个对象了，而指向了另一个 String 对象，内容为"Hello world!"，原来那个对象还存在于内存之中，只是 s 这个引用变量不再指向它了，

通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用 String 来代表字符串的话会引起很大的内存开销，因为 String 对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个 String 对象来表示。这时，应该考虑使用 StringBuffer 类，它允许修改，而不是每个不同的字符串都要生成一个新的对象，并且，这两种类的对象转换十分容易。

同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都 new 一个 String。例如我们要在构造器中对一个名叫 s 的 String 引用变量进行初始化，把它设置为初始值，应当这样做：

```
public class Demo
{ private String s;
...
public Demo {
s = "Initial Value";
}
...
}
```

而非

```
s = new String("Initial Value");
```

后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为 String 对象不可改变，所以对于内容相同的字符串，只要一个 String 对象来表示就可以了。也就说，多次调用上面的构造器创建多个对象，他们的 String 类型属性 s 都指向同一个对象。

上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java 认为它们代表同一个 String 对象。而用关键字 new 调用构造器，总是会创建一个新的对象，无论内容是否相同。

我把这份资料私底下发给过几个粉丝，他们通过这个复习之后，有的在工作上得到了很大的进步，有的拿到了更好的Offer。

扫描下方二维码"**程序猿进阶**"，关注后回复："**面试题**"，即可下载！