链滴

Java 进阶面试问题列表

作者：tianxin

原文链接：https://ld246.com/article/1488457563194

来源网站：链滴

许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

Java 进阶面试问题列表个人初步整理答案

面向对象编程理念与核心设计思想

多态性、封装性、内聚、耦合

1. 多态：把不同种类的东西当做相同的东西来处理 >

（举个例子：三个箱子，而多态的本质是什么：都是箱子，都可以打开箱子，但是打开箱子的具体动不同（方法）

2. 多态的优点：将各种数据统一的处理，根据对象的不同来选择最合适的方法

继承

1. 并不是说面向对象一定有继承，比如 javascript（但是非常重要的）
2. 多重继承（Python、C++）
3. 多重继承的缺点
4. Java中的继承：extends单继承 + implements 接口

知道哪些设计模式

1. 工厂模式 （Spring中的依赖注入中的IoC容器，BeanFactory与 ApplicationContext）
2. 单例模式 （Spring中的Bean的生命周期singleton 不同于我们设计模式中的单例模式，前者是容内单例，而后者是类加载器中单例）
3. 策略模式 （SQL在线审核系统中执行SQL语句时，导师提了一个需求，SQL具体执行的模块单独分出来，而且可以更换不同的执行策略）

    - 接口A定义策略
    - B、C 类实现该策略
    - Context 组合模式 组合接口A

4. 模板模式(Template，典型的是Spring中JDBCTemplate类，对JDBC进行封装)

    - JDBCTemplate 中已经封装好JDBC 的执行顺序，而具体执行的内容 （更新、查询）则交给子去做
    - 小技巧：使用Callback 回调函数形式可以避免再创建子类

5. 适配器模式（Tomcat中的适配器 CoyoAdapter： 在Nio）
6. 外观模式（Tomcat中的外观模式，RequestFacade）
7. 观察者模式 （Tomcat容器启动的过程）

    1. 抽象主题 （Subject）- 具体主题： 注册、删除、唤醒观察者
    2. 抽象观察者 （Observer）- 具体观察者

Tomcat中容器启动过程中

- 谁是观察者 （抽象观察者，LifecycleListener）
  - 具体观察者：ServerLifecycleListener
  - 抽象Subject：Lifecycle
  - 具体Subject： StandardServer

8. 装饰者模式 （JavaIO与装饰者）

9. 状态模式
10. 代理模式
11. 责任链模式
12. 原型模式
13. 迭代器模式
14. 组合模式

反模式

1. 过早优化（在Effective Java、Java并发编程模型中都多次提到过过早优化的种种缺点）

- 代码的性能瓶颈寻找问题（还没完成代码之前，如何谈性能）
- 对代码优化可能会提升代码的复杂度，提高Bug率

2. 不要讲大量时间花费在琐碎的问题上
3. 魔法数与字符串（而不是用一个变量去定义，在阿里巴巴编程规范中也有强烈指出该问题）
4. 避免那些无用的类（没有意义的类）
5. 不要用代码的行数来衡量项目的进度 （代码行数-进度； 重量 - 飞机进度）
6. 避免重新造轮子
7. 类的数量多，并不代表代码更复杂

常用重构的技巧

1. 重复代码提炼 （AOP）
2. 冗长方法的分割
3. 嵌套条件

思想：将不满足的条件放在前面，并及时跳出方法

```
class BadExample {

    public void someMethod(Object A,Object B){
        if (A != null) {
            if (B != null) {
                //code[1]
            }else {
                //code[3]
            }
        }else {
```

```
            //code[2]
        }
    }

}

/* ----------------------分割线---------------------- */

class GoodExample {

    public void someMethod(Object A,Object B){
        if (A == null) {
            //code[2]
            return;
        }
        if (B == null) {
            //code[3]
            return;
        }
        //code[1]
    }

}
```

分支的合并

```
class BadExample {

    public void someMethod(Object A,Object B){
        if (A != null) {
            if (B != null) {
                //code
            }
        }
    }

}

/* ----------------------分割线---------------------- */

class GoodExample {

    public void someMethod(Object A,Object B){
        if (A != null && B != null) {
            //code
        }
    }

}
```

复制代码

4. 一次性临时变量
5. 过长的参数列表 （Context上下文）
6. 常量提取

7. 把方法归类到类中，而不是再类之外单独实现
8. 冗长类的拆分
9. 将子类中重复的属性、方法提取到父类中（比如模板模式）

SOLID原则

1. 单一功能
2. 开闭原则
3. 里式替换
4. 接口隔离 ： 多个接口浩宇
5. 依赖反转 ： 一个方法应该“依赖于抽象而不是一个实例”

KISS、DRY、YAGNI原则

Java核心概念

1. 基础类型与封装类型的区别

- 内存分配与占据
- 基础类型的操作是直接由指令来操作

2. final关键字

- 修饰类： 类不可被继承
- 修饰方法：方法不可被重写
- 修饰字段： 字段的值初始化后不可再改变

3. static

- 修饰内部类：
- 修饰方法： 类方法，不用实例化就可以访问
- 修饰字段： 类成员变量（所有对象均共享，不用是实例化就可以访问）
- 修饰静态代码块： 类在第一次加载过程中就会执行该代码块，且只执行一次

4. java中的内部类

- 使用内部类来解决java不支持多重继承的缺点（implement只是解决了部分问题）
- 成员内部类： 不能存在static变量和方法；只有先创建外部类，才能创建内部类，在编译期间内部就存在一个对外部类的引用
- 静态内部类：不需要先创建外部类就可以直接访问静态内部类。不能使用外部类的任何非static修的方法与变量

5. StringBuilder 与 StringBuffer区别

- StringBuffer： 线程安全（方法上使用了 synchronized） JDK1.0
- StringBuilder : 线程不安全 JDK1.5

在JDK1.5之后，所有的字符串变量的连接操作（s1 + s2）都会采用StringBuilder来实现

6. Java中的不可变类

- String 与 Integer包装类
- 如何构建不可变类： final，private，getter

7. Java垃圾回收器简述原理

正向代理与反向代理区别

1. shadowsocks 与 nginx的应用区别
    - 正向代理是：客户