链滴

02Java 语言特性与设计模式

作者：dulinanaaa
原文链接：https://ld246.com/article/1556070592595
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)

常用的设计模式

有单例模式,工厂模式,代理模式,构造者模式,责任链模式,适配器模式,观察者模式等.

单例模式

线程安全实现的常见三种方法:

1. 静态初始化(饿汉).不管是否使用都会创建
2. 双重检查(懒汉).单例变量必须要用volatile修饰.（注意内存可见性引起的并发问题）
3. 单例注册表.spring中bean的单例模式就是用该方法实现.

```
// 双重检查锁的单例模式
public class Singleton {

    private volatile static Singleton uniqueSingleton;
    private Singleton() {
    }
    public Singleton getInstance() {
        if (null == uniqueSingleton) {
            synchronized (Singleton.class) {
                if (null == uniqueSingleton) {
                    uniqueSingleton = new Singleton();
                }
            }
        }
        return uniqueSingleton;
    }
}
```

工厂模式

这个是创建不同类型实例常用的方式。

例如Spring中的各种Bean是由不同的工厂类进行创建的

代理模式

- 主要用在不适合或不能直接引用另一个对象的场景。
- 可以用代理模式对被代理的队象进行访问行为的控制。
- Java的代理模式分为静态代理和动态代理
- 静态代理是指在编译时就创建好的代理类。例如我们在源代码中编写的类
- 动态代理指在JVM运行过程中动态创建的代理类

例如，在Mybatis中getMapper时会通过MapperProxyFactory及配置文件动态生成的Mapper代理象,代理对象会拦截Mapper接口的方法调用,创建对应方法的MapperMethod类并执行execute方法,后返回结果.

（使服务看上去就像是在使用本地的方法）

责任链模式

有点像工厂的流水线，链上的某一个节点完成对对象的某一种处理

Netty框架在处理消息时，使用的Pipeline就是一种责任链模式

适配器模式

类似于转接头，把两种不适配的对象进行适配，也可以起到对两个不同的对象进行解耦的作用。

日志处理框架SLF4J，可以跟Log4j、logback这种具体的日志实现框架进行解藕，通过不同适配器将SF4J与Log4j等不同框架进行适配，完成日志功能的使用。

观察者模式

也可称为发布订阅模式,适用于一个对象的某个行为需要触发一系列事件的场景。

GRPC中stream流式请求的处理就是通过观察者模式来实现的

构造者模式

适用于一个对象有很多复杂的属性，需要根据不同情况来创建不同具体的对象

创建Protocol Buffer对象时,需要用到Builder方式

Java语言特性

动态代理与反射

- 是java语言的特色,需要掌握动态代理与反射的使用场景
- ORM框架中会大量使用代理类
- PRC调用时使用反射机制调用实现类的方法

Java常见数据类型

面试的常见问题：

- 每种数据类型占用多大空间

(1.byte boolean 2.char short 4.int float 8.double)

- 数据类型的自动转换与强制转换
- 基础数据类型与Wrapper数据类型的自动装箱与拆箱等

Java对对象的引用

分为强引用、软引用、弱引用、虚引用4种。

这些引用在GC时的处理策略不同?

- 强引用不会被GC回收
- 软引用在内存空间不足时会被GC回收
- 弱引用在每次GC时都会被GC回收
- 虚引用必须和引用队列联合使用，主要用于跟踪一个对象被垃圾回收的过程

Java的异常处理机制

就是try/catch/finally机制

需要知道在异常时在try、catch中的处理流程

需要了解Error和Exception的区别

HashMap与ConcurrentHashMap

Map的实现这个题目，能够考查到数据结构、Java基础实现以及对并发问题的处理思路的掌握程度。

HashMap

- HashMap的实现

简单说，Java的HashMap就是数组+链表实现的

数组中的每一项都是一个链表

通过计算存入对象的HashCode，来计算对象在数组中要存入的位置，用链表来解决散列冲突，链表的节点存储的是键值对。

- 填充因子的作用
- Map扩容的rehash机制
- 需要知道它的容量是二的幂次方

是为了可以通过按位与操作来计算余数，比求模要快

- HashMap是非线程安全的

在多线程put的情况下，有可能容量在超过填充因子时进行rehash。因为HashMap为了避免尾部遍，在链表的插入时使用的是头插法，多线程的场景下，可能会产生死循环。

ConcurrentHashMap

会从非线程安全的HashMap就会自然的跳转到线程安全的ConcurrentHashMap

- 采用分段锁的思想来降低并发场景下的锁定发生频率
- 在jdk1.7和jdk1.8中的实现差异非常大
    - 1.7中采用segment分段加锁,降低并发锁定程度
    - 1.8中采用CAS自旋锁(一种乐观锁实现模式)提高性能.但在并发度较高时,性能一般.
    - 1.8ConcurrentHashMap引入红黑树，用来解决hash冲突时的链表顺序查找问题。
    - 红黑树的启用条件与链表的长度和map的总容量有关。默认是链表大于8，且容量大于64时转为黑树方式。建立阅读源码来进行学习

Java版本

jdk1.8和jdk1.11是长期支持版本

- jdk1.8
  - Lambda表达式
  - StreamAPI
  - 方法引用
  - 接口默认方法
  - （对方法区进行了调整）Metaspace替换PermGen

(Metaspace与PermGen的最大区别在于：Metaspace并不在虚拟机中，而是使用本地内存
替换的目地：1.提升对源数据的处理、提升GC效率2.方便后续Hotpot与gearotic合并)

- jdk1.9-1.10
  - 模块系统
  - 默认G1回收器
  - 接口私有方法
  - 局部变量判断
  - Graal编译器
- jdk1.11
  - ZGC(最激动人心的，新的gc回收器)
    - 为大内存堆设计，能够实现10ms以下的gc停顿
  - 字符串API增强（提供了字符复制等功能）
  - 内建HTTP Client

面试考察点

1. 基本概念和基本原理

要求:正确清晰

- 网络协议4/7层模型的概念
- TCP协议流量控制的实现原理
- 等

2. 实现方法和使用方式

- HashMap在JDK1.8中的实现方式
- 单例模式有哪几种实现方式,什么场景该使用静态方法实现,什么场景该使用双检锁实现
- 等

3. 经常用到的知识点

- 常用的Linux命令有哪些,用来解决什么样的问题
  - 等

4. 实际应用中容易犯错的点
   - ==与equals区别是什么
   - 对象强引用使用不当会导致内存泄露,考察不同引用方式和作用的理解
   - 等

5. 与面试方向相关的知识点
   - 中间件:存储,网络相关的考察
   - 等

加分项

1. 知识点与典型的业务场景关联.

如,谈到设计模型时,可以讲XX框架在解决XX问题时使用了那种设计模式.

2. 以反例来描述实际场景中误用的危害.

如,大量使用反射会影响性能.

3. 与知识点相关的优化点.

如,讲到tcp建连和断连时,如遇到洪水攻击或大量TIME_WAIT时,可以调整系统参数预防.

4. 与知识点相关的最新技术趋势.

如,讲到ConcurrentHashMap,可以介绍1.8的改进细节.

或,讲到HTTP时,能说出HTTP2和QUIC的特点和实现.

5. 在了解的前提下,尽量增加回答内容深度.

如,讲到tcp的滑动窗口时,能讲到流量与拥塞控制,进一步能指出解决拥塞的不同算法.

真题汇总-1

1. 进程和线程的区别和联系

从资源占用,切换效率,通信方式等方面解答

2. 简单介绍一下进程的切换过程

线程上下文的切换代价,要回答,切换会保存寄存器,栈等线程相关的现场,需要由用户态切换到内核态,可用vmstat命令查看线程上下文的切换状况

3. 你经常使用哪些Linux命令,主要用来解决哪些问题?

参考之前操作系统汇总中提到的命令

4. 为什么TCP建连需要3次握手而断连需要4次?

参考之前内容

5. 为什么TCP关闭链接时需要TIME_WAIT状态,为什么要等2MSL?

参考之前内容

6. 一次完整的HTTP请求过程是怎样的?

DNS解析,TCP建连,HTTP请求,HTTP响应等.

实际回答时可以画一个简单的交互图

7. HTTP2和HTTP的区别有哪些?
8. 在你的项目中你使用过哪些设计模式？主要用来解决哪些问题?
9. Object中的equals和hashcode的作用分别是什么?
10. final,finally,finalize的区别与使用场景
11. 简单表述一下Java的异常机制
12. 线上上使用的那个版本jdk,为什么使用这个版本(有什么特色)?