面试经历整理

一、Java基础

设计模式

实践案例

1.工厂模式

简单工厂模式：

简单工厂模式核心：

简单工厂模式就是专门负责将大量有共同接口的类实例化，而且不必事先知道每次是要实例化哪个一个类的模式。它定义一个用与创建对象的接口，由子类决定实例化哪一个类，根据外界传递的信息来决定创建哪个具体类的对象。

简单工厂模式的优缺点：

优点：它的核心是工厂类，这个类负责产品的创建，而客户端可以免去产品创建的责任，这实现了责任的分割。

缺点：由于工厂类集中了所有产品创建逻辑，如果不能正常工作的话会对系统造成很大的影响。如果增加新产品必须修改工厂类的代码。

2.单例模式

- 介绍：
  - **意图：**保证一个类仅有一个实例，并提供一个访问它的全局访问点。
  - **主要解决：**一个全局使用的类频繁地创建与销毁。
  - **何时使用：**当您想控制实例数目，节省系统资源的时候。
  - **如何解决：**判断系统是否已经有这个单例，如果有则返回，如果没有则创建。
  - **关键代码：**构造函数是私有的。
  - **应用实例：**
    - 一个班级只有一个班主任。
    - Windows 是多进程多线程的，在操作一个文件的时候，就不可避免地出现多个进程或线程同时操作一个文件的现象，所以所有文件的处理必须通过唯一的实例来进行。
    - 一些设备管理器常常设计为单例模式，比如一个电脑有两台打印机，在输出的时候就要处理不能两台打印机打印同一个文件。
  - **优点：**

        - 在内存里只有一个实例，减少了内存的开销，尤其是频繁的创建和销毁实例（比如管理学院首页页面缓存）。
        - 避免对资源的多重占用（比如写文件操作）。
    - **缺点：**
        - 没有接口，不能继承，与单一职责原则冲突
        - 一个类应该只关心内部逻辑，而不关心外面怎么样来实例化。
    - **使用场景：**
        - 要求生产唯一序列号。
        - WEB 中的计数器，不用每次刷新都在数据库里加一次，用单例先缓存起来。
        - 创建的一个对象需要消耗的资源过多，比如 I/O 与数据库的连接等。
    - **注意事项：**getInstance() 方法中需要使用同步锁 synchronized (Singleton.class) 防止多线程同时进入造成 instance 被多次实例化
- 特点：
    - 单例模式一个类只能有一个实例
    - 单例模式必须自己创建自己的实例。
    - 单例模式必须给所有其它对象提供这一个实例
- 单例模式主要分为两种：
    - 懒汉式单例
    - 饿汉式单例
- 懒汉式和饿汉式的区别：
    - 懒汉式是在你真正用到的时候才去建这个对象
    - 饿汉式是在你没用到的时候就已经建好对象了
- 两种单例的要素：
    - 私有的构造方法
    - 指向自己实例的私有静态引用
    - 以自己实例为返回值的静态的公有方法
- 实现方法
    - 饿汉式：

```
public class EagerSingleton {
  private EagerSingleton(){} //私有化构造函数
  private static EagerSingleton eagerSingleton=new EagerSingleton() ;
  //获取对象
  public static EagerSingleton getEagerSingleton() {
    return eagersingleton;
  }
}
```

      这种写法就是所谓的单例的饿汉模式，该缺点也能一眼辨出，就是对象在没使用之前就已经在内存中实例化了，这就带来了一个潜在的性能问题，如果这个对象很大呢 => 懒汉式

    - 懒汉式：

```
package com.test.s1ngleton;
public class LazySingleton {
  private LazySingleton() {} //私有化构造函数
  private static LazySingleton lazySingleton=null;
  //获取对象
  public static LazySingleton getLazySingleton() {
    if(lazySingleton==nu11) {
      lazySingleton=new LazySingleton() ;
      return lazySingleton;
    }
    return lazySingleton;
  }
```

这就是所谓的懒汉模式，它沿用了一种设计思想-延迟加载(例如：Mybatis延迟加载)，也就是我没用到这个对象时是不会去实例化的。

在上面两种单例模式的写法当中它们都有一个相同的地方，那就是要私有化构造函数，私有构造函数就是为了防止外界去new该对象。

但是懒汉模式就没有了缺点吗？不，该模式是存在**线程不安全**的缺点的，可是该怎么写才能防止线程安全且性能没隐患的单例模式呢?

其他见：https://www.jianshu.com/p/2cebc62d7f28

3.代理模式

面向对象

1.equals和==的区别：

- == 比较的是变量(栈)内存中存放的对象的(堆)内存地址，用来判断两个对象的地址是否相同，即是否是指相同一个对象。比较的是真正意义上的指针操作。
- equals用来比较的是两个对象的内容是否相等，由于所有的类都是继承自java.lang.Object类的，所以适用于所有对象，如果没有对该方法进行覆盖的话，调用的仍然是Object类中的方法，而Object中的equals方法返回的却是==的判断

2.包装机制

https://blog.csdn.net/qq_43948583/article/details/90294422

- 不可变对象：普通类可以通过set()、get()对成员方法修改，不可变对象不可以被修改，即不可变对象
- 基本类型的包装类：int、double、... => Integer、Double、...
  - 基本类型的包装类是final类，不可修改

    - 好处：基本类型统一成了对象，可以将一些工具类放入包装器中
    - 泛型List nums = new ArrayList()，<>中不可以有基本数据类型，所以需要包装类
    - Nums.add() 可以直接传入一个整数类型int，这是因为Java中存在有自动包装机制

- 自动装箱与拆箱机制
- 其他详细见链接
- 包装机制与JVM：https://blog.csdn.net/weixin_34198583/article/details/91931829

多线程：

1.多线程实践案例：

电影院抢票

2.多线程的实现方法

new thread、implements runnable、threadlocal

3.多线程的线程池是如何工作的顶层代码和原理

4.本地线程thredocal的实现原理

File类、IO流

IO流实践案例及应用场景：

集合类：

1.HashSet实现什么接口

2.HashSet的去重原理(底层HashMap实现)、HashMap问的频繁

3.jdk1.7和1 8的hashmap有什么不同

HashMap的结构，JDK7与JDK8有哪些区别：

- 浅显理解：https://blog.csdn.net/changhangshi/article/details/82114727
- 深入理解(高级工程师)：https://blog.csdn.net/qq_36520235/article/details/82417949

在**JDK1.8**之前，哈希表底层采用数组+链表实现，即使用链表处理冲突，同一hash值的链表都存储在一个链表里。但是当位于一个桶中的元素较多，即hash值相等的元素较多时，通过key值依次查找的效率较低。

而JDK1.8中，哈希表存储采用数组+链表+红黑树实现，当链表长度超过阈值（8）时，将链表转换为红黑树，这样大大减少了查找时间。

1.重要概念：

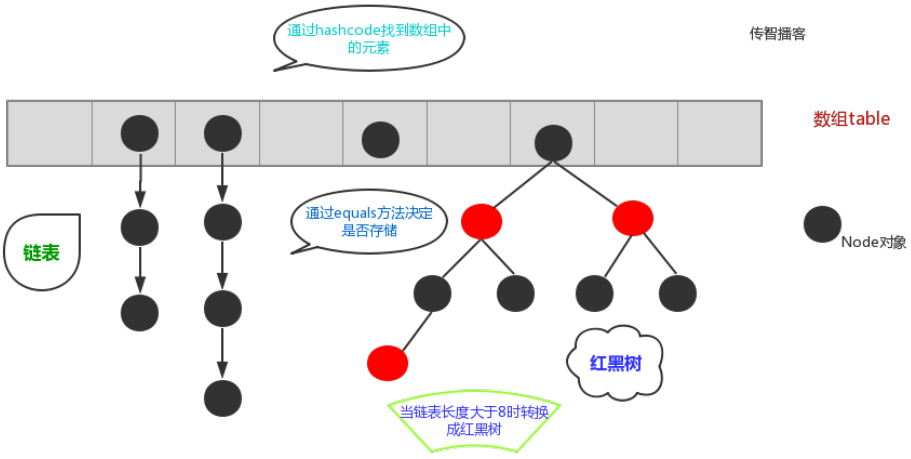
哈希表：查询速度快

哈希值 hashcode：逻辑地址，系统随机给

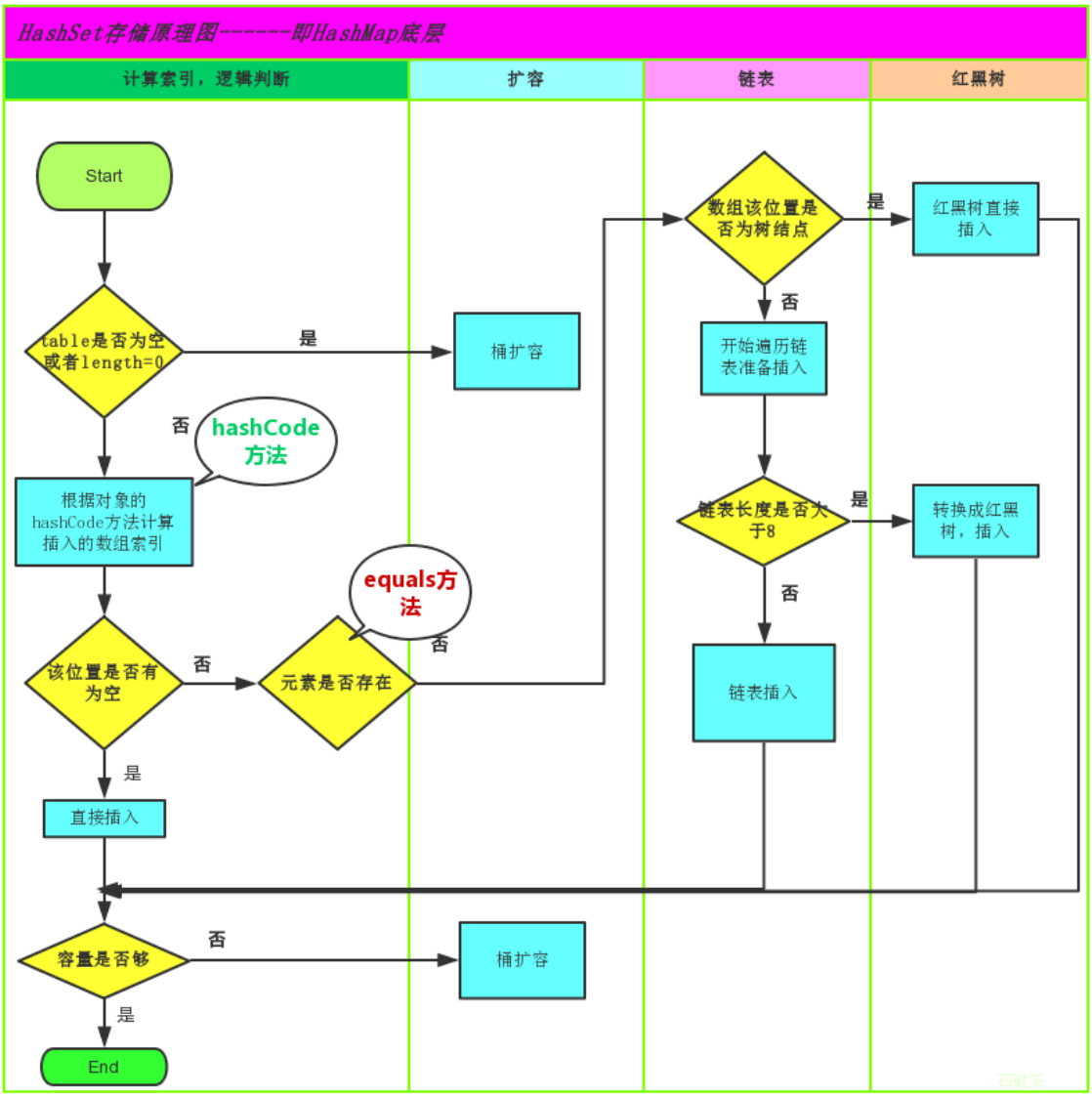
2.原理剖析：

- Java中的HashSet去重操作
  - 添加值add()通过hashcode、equals()字符串比较，确定存储对象的桶位置(hash表位置)→**先比较hashcode->冲突再比较字符串equals()**，若字符串同则认定重复、
- HashMap实现原理：
- 不同对象相同hashcode处理方法：由于不同对象可能有相同hash值(数据结构中的hash冲突)、把相同hashcode的对象通过拉链法拉到一起，(1.8之前是链表+数组、1.8之后是链表+数组+红黑树(拉链长度超过8，则用红黑树))->从而确定对象的桶的唯一位置。
- 为什么使用1.8使用红黑树，效率提升在哪?
- jdk1.5之前的hashmap => hashtable
- hashmap为什么不是线程安全？乐观锁(cas)、悲观锁(syschronized)：百度
- 数据结构剖析：

  简单的来说，哈希表是由数组+链表+红黑树（JDK1.8增加了红黑树部分）实现的，如下图所示。



结合一个存储流程图来说明一下：



JDK1.8引入红黑树大程度优化了HashMap的性能，那么对于我们来讲保证HashSet集合元素的唯一，其实就是根据对象的hashCode和equals方法来决定的。如果我们往集合中存放自定义的对象，那么保证其唯一，就必须复写hashCode和equals方法建立属于当前对象的比较方式。

4.HashMap里的 hashcode 方法和 equal 方法什么时候需要重写？如果不重写会有什么后果?

回答：

- 当实例化的两个对象具有相同的属性时 => 需要重写hashcode() & equal()，先比较hashcode若不同则不需要调用equal()，直接认定不同
- 若不重写则不会去重，直接把相同属性的两个对象都添加到HashSet集合中

原理解析：

HashSet内部是通过HashMap实现。只有使用排序的时候才使用TreeMap => 否则使用HashMap。

```
HashSet set = new HashSet()
set.put(new Student(1,"aa") );
set.put(new Student(1,"aa") );
set.put(new Student(2,"aa") );
//结果set内的元素为3个，没有去除重复的new Student(1,"aa") ？为什么呢?
```

这里由于两个new Student(1,"aa")是不一样的Student对象而默认的Student类的hashcode是根据对象的引用算的，所以直接认为是两个不一样的对象，直接put进去了，所以需要重写hashcode方法，如果hashcode不一样则直接认为是不同对象，如下：

```
/*重写实体类的hashcode*/
class Student {
        private int code;
        private Stirng name;
        public int hashcode ()  {
            return code*name.hashcode();
        }
}
```

发现还是不对，还是put进去了呢?
这里重写的hashcode是一样的，所以还是put进去了。所以还需要重写equals方法。其实是有这样一个规定，**如果hahscode一样时，则还需要继续调用equals方式看看对象是否相等**。如下即可实现：

```
public boolean equals (Object o)  {
     Student s = (Student ) o;
     if (name.equals(s.getName) && code == o.getCode()) {
          return true;
     }
     return false;
}
```

可以看到如果hashcode不一样就直接认为是不一样的对象，不需要再去equal比较，更加节省时间。
如果`new Student(1,"aa")、new Student(1,"bb")`。通过code和name算出的hashcode就可以算是不一样的对象，就不需要再去equals比较。
往往HashSet中存放的对象是否相等的逻辑都需要自己定义，而并不会直接用默认的引用来算，即一般都会重新hashcode和equals方法，而且同时需要重写。以后要注意哦。

HashMap的put和get也类似

HashMap是底层实现时数组加链表。

A.当put元素时:

- 首先根据put元素的key获取hashcode,然后根据hashcode算出数组的下标位置,如果下标位置没有元素，直接放入元素即可
- 如果该下标位置有元素（即根据put元素的key算出的hashcode一样即重复了），则需要已有元素和put元素的key对象比较equals方法，如果equals不一样，则说明可以放入进map中。这里由于hashcode一样，所以得出的数组下标位置相同。所以会在该数组位置创建一个链表，后put进入

的元素到放链表头，原来的元素向后移动。

B.当get元素时:

根据元素的key获取hashcode，然后根据hashcode获取数组下标位置，如果只有一个元素则直接取出。如果该位置一个链表，则需要调用equals方法遍历链表中的所有元素与当前的元素比较,得到真正想要的对象。

可以看出如果根据hashcdoe算出的数组位置尽量的均匀分布，则可以避免遍历链表的情况，以提高性能。
所以要求重写hashmap时，也要重写equals方法。以保证他们是相同的比较逻辑

5.LinkedMap、HashMap、TreeMap

共同点：

- HashMap,LinkedHashMap,TreeMap都属于Map；
- Map 主要用于存储键(key)值(value)对，根据键得到值，因此键不允许键重复,但允许值重复。

不同点:

- HashMap里面存入的键值对在取出的时候是随机的,也是我们最常用的一个Map.它根据键的HashCode值存储数据,根据键可以直接获取它的值，具有**很快的访问速度。在Map 中插入、删除和定位元素**，HashMap 是最好的选择。
- TreeMap取出来的是排序后的键值对。但如果您要按**自然顺序**（字典顺序），那么TreeMap会更好。
- LinkedHashMap 是HashMap的一个子类，如果需要输出的**顺序和输入的相同**,那么用LinkedHashMap可以实现。

> 补充：如何选择合适的Map

- HashMap可实现快速存储和检索，但其缺点是其包含的元素是**无序的**，这导致它在存在大量**迭代的情况下表现不佳**。
- LinkedHashMap保留了HashMap的优势，且其包含的元素是**有序的**。它在有大量**迭代的情况下表现更好**。
- TreeMap能便捷的实现对其内部元素的**各种排序**，但其一般**性能比前两种map差**。
- **LinkedHashMap映射减少了HashMap排序中的混乱，且不会导致TreeMap的性能损失**

6. HashSet、LinkedHashSet、TreeSet比较

- HashSet底层是HashMap
- TreeSet底层是HashSet

HashSet

- 不能保证元素的排列顺序，顺序有可能发生变化
- 不是同步的
- 集合元素可以是null,但只能放入一个null
- 说明：

当向HashSet结合中存入一个元素时，HashSet会调用该对象的hashCode()方法来得到该对象的hashCode值，然后根据 hashCode值来决定该对象在HashSet中存储位置。

简单的说，HashSet集合判断两个元素相等的标准是两个对象通过equals方法比较相等，并且两个对象的hashCode()方法返回值相等

> 注意，如果要把一个对象放入HashSet中，重写该对象对应类的equals方法，也应该重写其hashCode()方法。其规则是如果两个对象通过equals方法比较返回true时，其hashCode也应该相同。另外，对象中用作equals比较标准的属性，都应该用来计算 hashCode的值。

LinkedHashSet

- LinkedHashSet集合同样是根据元素的hashCode值来决定元素的存储位置，但是它同时使用链表维护元素的次序。
- 这样使得元素看起来像是以插入顺序保存的，也就是说，当遍历该集合时候，LinkedHashSet将会以元素的添加顺序访问集合的元素。
- LinkedHashSet在**迭代访问**Set中的全部元素时，性能比HashSet好，但是**插入时性能稍微逊色**于HashSet。

TreeSet类

- TreeSet是SortedSet接口的唯一实现类，TreeSet可以确保集合元素处于排序状态。
- TreeSet支持两种排序方式，自然排序 和定制排序，其中自然排序为默认的排序方式。
- 向TreeSet中加入的应该是同一个类的对象。
- TreeSet判断两个对象不相等的方式是两个对象通过equals方法返回false，或者通过CompareTo方法比较没有返回0

网络编程

网络编程实践案例及应用场景：

模拟BS服务器、模拟文件上传

反射

垃圾回收机制

jvm的三种回收方式有什么不同比如CMS和G1有什么区别

二、Web类

三、前端类

vue框架是否使用过是否会用

四、Java框架类

springmvc 与 springboot 区别

springboot：通过spring提供的@ImportResource来加载xml配置

springcloud、单体，docker部署

maven原理，是啥

Model2、ssm、springboot，整合优缺点及心得

Nginx是怎么用的

Redis是怎么用的

springCloud微服务和插件的问题很多

Mybatis面试题：

https://blog.csdn.net/a745233700/article/details/80977133

mybatis

动态sql

https://www.cnblogs.com/dongying/p/4092662.html

三、数据库

MySQL & Oracle

数据库优化时候索引有哪几种如何选择索引

Redis

四、分布式、微服务、高并发

如何处理高并发 => 就是请求数量大于本地线程数量的时候不出错

多人同时抢购一个库存的商品该怎么处理(分布式事务)

其他：

你的优势在哪

如何解决bug?

首先你得确认，在以前的开发中有过大Bug吗？如果没有的话就编吧，编你熟悉的领域

Bug是业务型的还是技术型的，当然技术型的Bug更能体现你解决问题的能力，总体分为以下几步

1. 功能详细，说你要做的这个功能是什么
2. 如何引发了bug，偶然的还是必现的
3. 出现了什么样的后果，带来了什么损失
4. 你的第一反应以及团队的第一反应
5. 如何根据出现的错误定位到了这个bug，如何排查的
6. 排查到bug所花费的时间，以及推理步骤
7. 尝试了几种解决方案

大概就这么多吧，把整个bug看成一个事件，慢慢来分析，注意这个问题并不是说上来就行了，面试官会从中问你不少的问题，所以还是要说自己的熟悉的领域，如果发生过那就更那办了