### 一．java基础

#### 1.继承：

父类称为基类、超类，子类称为派生类、衍生类。extends关键字，格式：[访问修饰符] class 子类 extends 父类 { ... }。继承的特点:1.子类自动拥有父类的非私有的属性和方法2.子类可重写(覆盖)父类的方法,按照自己的方式做3.子类可扩展父类的属性和行为4.子类的访问修饰符不能比父类的更严格

#### 2、方法重载和重写的区别：

Override 方法重写 | 方法覆盖 Overload 方法重载

1.方法名一致 方法名一致

2.参数一致 参数不同(类型 顺序 个数)

3.继承关系的类 本类

4.返回类型一致 与返回类型无关

方法原型 = 返回类型 方法名(参数列表)

#### 3.super关键字及三种用法：

super() 显式调用父类的构造方法且在子类构造方法的第一行

super.name 显式调用父类的属性和方法

super.run()

#### 4、super 与 this 的用法

super（参数）：调用基类中的某一个构造函数（应该为构造函数中的第一条语句）

this（参数）：调用本类中另一种形成的构造函数（应该为构造函数中的第一条语句）

super:　它引用当前对象的直接父类中的成员（用来访问直接父类中被隐藏的父类中成员数据或函数，基类与派生类中有相同成员定义时如：super.变量名 super.成员函数据名（实参）

this：它代表当前对象名（在程序中易产生二义性之处，应使用this来指明当前对象；如果函数的形参与类中的成员数据同名，这时需用this来指明成员变量名）

调用super()必须写在子类构造方法的第一行，否则编译不通过。每个子类构造方法的第一条语句，都是隐含地调用super()，如果父类没有这种形式的构造函数，那么在编译的时候就会报错。

super()和this()类似,区别是，super()从子类中调用父类的构造方法，this()在同一类内调用其它方法。

super()和this()均需放在构造方法内第一行。

尽管可以用this调用一个构造器，但却不能调用两个。

this和super不能同时出现在一个构造函数里面，因为this必然会调用其它的构造函数，其它的构造函数必然也会有super语句的存在，所以在同一个构造函数里面有相同的语句，就失去了语句的意义，编译器也不会通过。

this()和super()都指的是对象，所以，均不可以在static环境中使用。包括：static变量,static方法，static语句块。

从本质上讲，this是一个指向本对象的指针, 然而super是一个Java关键字。

#### 5.访问修饰符：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 访问权限 | 类 | 包 | 子类 | 其他 |
| Public | √ | √ | √ | √ |
| Protected | √ | √ | √ | × |
| Default | √ | √ | × | × |
| Private | √ | × | × | × |

#### 6.引用转型：

隐式转型 - 自动 - 小类型转大类型

显式转型 - 强制 - 大类型转小类型

#### 7、面向对象和面向过程的区别？

面向过程：优点：性能比面向对象高，因为类调用时需要实例化，开销比较大，比较消耗资源。比如，单片机、嵌入式开发、Linux/Unix 等一般采用面向过程开发，性能是最重要的因素。缺点：没有面向对象易维护、易复用、易扩展。

、面向对象:优点：易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护。缺点：性能比面向过程低。

#### **8、什么是值传递和引用传递？**

值传递，是对基本型变量而言的，传递的是该变量的一个副本，改变副本不影响原变量。引用传递，一般是对于对象型变量而言的，传递的是该对象地址的一个副本，并不是原对象本身。一般认为，Java 内的传递都是值传递，Java 中实例对象的传递是引用传递。

#### **9、接口和抽象类的区别**

|  |  |  |
| --- | --- | --- |
| 比较 | 抽象类（abstract class） | 接口（interface） |
| 默认方法 | 抽象类可以有默认的方法实现 | Java8之前，接口中不存在方法的实现 |
| 实现方式 | 子类使用extends关键字来继承抽象类.如果子类不是抽象类,子类需要提供抽象类中所声明方法的实现 | 子类使用implements来实现接口，需要提供接口中所有声明的实现 |
| 构造器 | 抽象类中可以有构造器, | 接口中不能 |
| 访问修饰符 | 抽象方法中可以有public，protected,和default等修饰 | 接口默认是public ，不能使用其他修饰符。接口中的方法默认是public abstract修饰 |
| 多继承 | 一个子类只能存在一个父类 | 一个子类可以存在多个接口 |
| 访问新方法 | 向抽象类中添加新方法，可以提供默认的实现，因此可以不修改子类现有的代码 | 子类必须实现接口的该新方法 |

#### **10、静态变量和实例变量的区别?**

静态变量存储在方法区，属于类所有。实例变量存储在堆当中，其引用存在当前线程栈。

#### **11、讲讲类的实例化顺序？**

初始化顺序如下：

父类静态变量-父类静态代码块-子类静态变量-子类静态代码块-父类非静态变量（父类实例成员变量）-父类构造函数-子类非静态变量（子类实例成员变量）-子类构造函数

#### **12、java 创建对象的几种方式**

采用new

通过反射

采用clone

通过序列化机制

前2者都需要显式地调用构造方法。造成耦合性最高的恰好是第一种，因此你发现无论什么框架，只要涉及到解耦必先减少new的使用

#### 13、理解异常

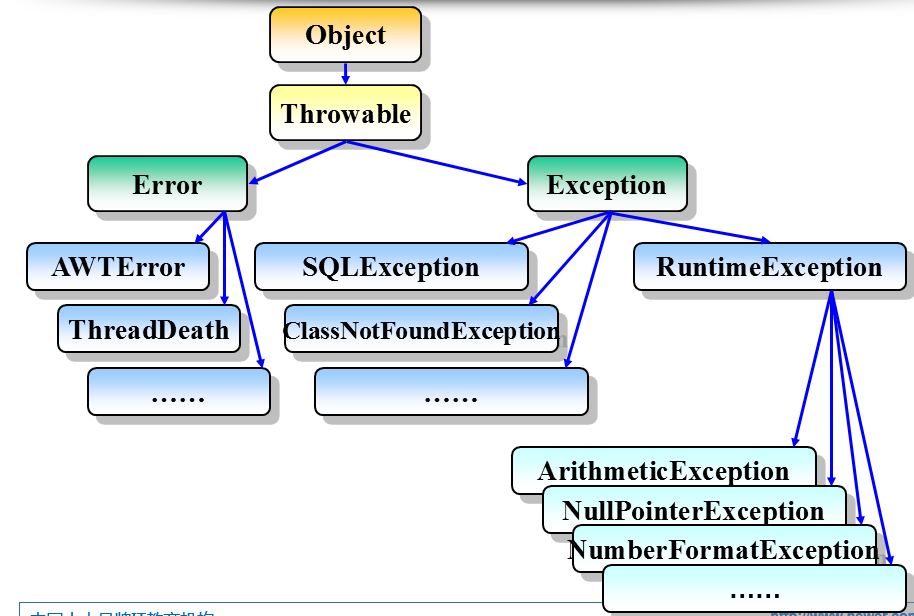
1.编译时异常(必须处理后才能运行代码)

2.运行时异常(执行代码时抛异常)

3、运用关键字throw和throws处理异常

throw语句用于手工抛出异常；

执行流程将在throw语句后立即停止，转而寻找与之类型相匹配的catch块。如果某个函数中的代码有可能引发异常，可以使用try/catch块进行处理，这种处理方式成为“内部处理”； 如果不方便在函数内部进行处理，也可以将异常往函数外部传递，这就要使用到关键字throws； throws用于将函数内部产生的异常抛给主调函数



#### 14、Object中有哪些公共方法?

equals()、clone()、getClass()、notify()、notifyAll()、wait()、toString

#### 15、a==b与a.equals(b)有什么区别

如果a 和b 都是对象，则 a==b 是比较两个对象的引用，只有当 a 和 b 指向的是堆中的同一个对象才会返回 true，而 a.equals(b) 是进行逻辑比较，所以通常需要重写该方法来提供逻辑一致性的比较。例如，String 类重写 equals() 方法，所以可以用于两个不同对象，但是包含的字母相同的比较。

基本类型比较用==，比较的是他们的值。默认下，对象用==比较时，比较的是内存地址，如果需要比较对象内容，需要重写equal方法。

#### 16、final有哪些用法

1.被final修饰的类不可以被继承

2.被final修饰的方法不可以被重写

3.被final修饰的变量不可以被改变。如果修饰引用，那么表示引用不可变，引用指向的内容可变。

4.被final修饰的方法，JVM会尝试将其内联，以提高运行效率

5.被final修饰的常量，在编译阶段会存入常量池中。

回答出编译器对final域要遵守的两个重排序规则更好：

1.在构造函数内对一个final域的写入，与随后把这个被构造对象的引用赋值给一个引用变量,这两个操作之间不能重排序。

2.初次读一个包含final域的对象的引用，与随后初次读这个final域,这两个操作之间不能重排序。

#### 17、String, StringBuffer和StringBuilder区别

String是字符串常量，final修饰：StringBuffer字符串变量(线程安全)；StringBuilder 字符串变量(线程不安全)。

String和StringBuffer主要区别是性能：String是不可变对象，每次对String类型进行操作都等同于产生了一个新的String对象，然后指向新的String对象。所以尽量不在对String进行大量的拼接操作，否则会产生很多临时对象，导致GC开始工作，影响系统性能。

StringBuffer是对对象本身操作，而不是产生新的对象，因此在有大量拼接的情况下，我们建议使用StringBuffer。

但是需要注意现在JVM会对String拼接做一定的优化：

String s=“This is only ”+”simple”+”test”会被虚拟机直接优化成String s=“This is only simple test”，此时就不存在拼接过程。

StringBuffer和StringBuilder：

StringBuffer是线程安全的可变字符串，其内部实现是可变数组。StringBuilder是jdk 1.5新增的，其功能和StringBuffer类似，但是非线程安全。因此，在没有多线程问题的前提下，使用StringBuilder会取得更好的性能。

#### 18、final、finalize 和 finally 的不同之处？

final 是一个修饰符，可以修饰变量、方法和类。如果 final 修饰变量，意味着该变量的值在初始化后不能被改变。Java 技术允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在确定这个对象没有被引用时对这个对象调用的，但是什么时候调用 finalize 没有保证。finally 是一个关键字，与 try 和 catch 一起用于异常的处理。finally 块一定会被执行，无论在 try 块中是否有发生异常。

#### 19、常见集合接口问题：

##### 1.List，Set,Map三者的区别及总结

List：对付顺序的好帮手

List接口存储一组不唯一（可以有多个元素引用相同的对象），有序的对象

* Arraylist：数组（查询快,增删慢 线程不安全,效率高 ）
* Vector：数组（查询快,增删慢 线程安全,效率低 ）
* LinkedList：链表（查询慢,增删快 线程不安全,效率高 ）

Set:注重独一无二的性质

不允许重复的集合。不会有多个元素引用相同的对象。

* HashSet（无序，唯一）:哈希表或者叫散列集(hash table)
* LinkedHashSet：链表和哈希表组成 。 由链表保证元素的排序 ， 由哈希表证元素的唯一性
* TreeSet（有序，唯一）：红黑树(自平衡的排序二叉树。)

Map:用Key来搜索的专家

使用键值对存储。Map会维护与Key有关联的值。两个Key可以引用相同的对象，但Key不能重复，典型的Key是String类型，但也可以是任何对象。

* HashMap：基于哈希表的Map接口实现（哈希表对键进行散列，Map结构即映射表存放键值对）
* LinkedHashMap:HashMap 的基础上加上了链表数据结构
* HashTable:哈希表
* TreeMap:红黑树（自平衡的排序二叉树）

##### 2.Arraylist 与 LinkedList 区别

Arraylist底层使用的是数组（存读数据效率高，插入删除特定位置效率低），LinkedList 底层使用的是双向链表数据结构（插入，删除效率特别高）（JDK1.6之前为循环链表，JDK1.7取消了循环。注意双向链表和双向循环链表的区别：）； 详细可阅读JDK1.7-LinkedList循环链表优化。学过数据结构这门课后我们就知道采用链表存储，插入，删除元素时间复杂度不受元素位置的影响，都是近似O（1）而数组为近似O（n），因此当数据特别多，而且经常需要插入删除元素时建议选用LinkedList.一般程序只用Arraylist就够用了，因为一般数据量都不会蛮大，Arraylist是使用最多的集合类。

##### 3.ArrayList 与 Vector 区别

Vector类的所有方法都是同步的。可以由两个线程安全地访问一个Vector对象、但是一个线程访问Vector ，代码要在同步操作上耗费大量的时间。Arraylist不是同步的，所以在不需要同步时建议使用Arraylist。

##### 4.HashMap 和 Hashtable 的区别

HashMap是非线程安全的，HashTable是线程安全的；HashTable内部的方法基本都经过synchronized修饰。

因为线程安全的问题，HashMap要比HashTable效率高一点，HashTable基本被淘汰。

HashMap允许有null值的存在，而在HashTable中put进的键值只要有一个null，直接抛出NullPointerException。

Hashtable和HashMap有几个主要的不同：线程安全以及速度。仅在你需要完全的线程安全的时候使用Hashtable，而如果你使用Java5或以上的话，请使用ConcurrentHashMap吧

HashSet 和 HashMap 区别



##### 5.HashMap 和 ConcurrentHashMap 的区别

ConcurrentHashMap对整个桶数组进行了分割分段(Segment)，然后在每一个分段上都用lock锁进行保护，相对于HashTable的synchronized锁的粒度更精细了一些，并发性能更好，而HashMap没有锁机制，不是线程安全的。（JDK1.8之后ConcurrentHashMap启用了一种全新的方式实现,利用CAS算法。）

HashMap的键值对允许有null，但是ConCurrentHashMap都不允许。

#### 20、comparable 和 comparator的区别

comparable接口实际上是出自java.lang包 它有一个 compareTo(Object obj)方法用来排序

comparator接口实际上是出自 java.util 包它有一个compare(Object obj1, Object obj2)方法用来排序

一般我们需要对一个集合使用自定义排序时，我们就要重写compareTo方法或compare方法，当我们需要对某一个集合实现两种排序方式，比如一个song对象中的歌名和歌手名分别采用一种排序方法的话，我们可以重写compareTo方法和使用自制的Comparator方法或者以两个Comparator来实现歌名排序和歌星名排序，第二种代表我们只能使用两个参数版的Collections.sort().

#### 21、如何求ArrayList集合的交集 并集 差集 去重复并集

需要用到List接口中定义的几个方法：

addAll(Collection<? extends E> c) :按指定集合的Iterator返回的顺序将指定集合中的所有元素追加到此列表的末尾 实例代码：

retainAll(Collection<?> c): 仅保留此列表中包含在指定集合中的元素。

removeAll(Collection<?> c) :从此列表中删除指定集合中包含的所有元素。

#### 22、IO

事务ASID 原子性·、一致性、

《Java并发编程的艺术》。

单例模式

volatile和synchronized关键字

synchronized关键字最主要的三种使用方式的总结：

修饰实例方法，作用于当前对象实例加锁，进入同步代码前要获得当前对象实例的锁

修饰静态方法，作用于当前类对象加锁，进入同步代码前要获得当前类对象的锁 。也就是给当前类加锁，会作用于类的所有对象实例，因为静态成员不属于任何一个实例对象，是类成员（ static 表明这是该类的一个静态资源，不管new了多少个对象，只有一份，所以对该类的所有对象都加了锁）。所以如果一个线程A调用一个实例对象的非静态 synchronized 方法，而线程B需要调用这个实例对象所属类的静态 synchronized 方法，是允许的，不会发生互斥现象，因为访问静态 synchronized 方法占用的锁是当前类的锁，而访问非静态 synchronized 方法占用的锁是当前实例对象锁。

修饰代码块，指定加锁对象，对给定对象加锁，进入同步代码库前要获得给定对象的锁。 和 synchronized 方法一样，synchronized(this)代码块也是锁定当前对象的。synchronized 关键字加到 static 静态方法和 synchronized(class)代码块上都是是给 Class 类上锁。这里再提一下：synchronized关键字加到非 static 静态方法上是给对象实例上锁。另外需要注意的是：尽量不要使用 synchronized(String a) 因为JVM中，字符串常量池具有缓冲功能！

**双重校验锁实现对象单例（线程安全）**

public class Singleton {

private volatile static Singleton uniqueInstance;

private Singleton() {

}

public static Singleton getUniqueInstance() {

//先判断对象是否已经实例过，没有实例化过才进入加锁代码

if (uniqueInstance == null) {

//类对象加锁

synchronized (Singleton.class) {

if (uniqueInstance == null) {

uniqueInstance = new Singleton();

}

}

}

return uniqueInstance;

}

}

地址：

<https://github.com/zhuChaoc/JavaGuide/blob/master/Java/synchronized.md>

1. 集合

3.ssm框架基本介绍

4.spring IOC AOP bean工厂new实例过程

5.常见的设计模式

6.数据库优化

7.多线程过程

8.分布式-集群-异步

9.