**Java基础**

https://www.toutiao.com/i6723563574100230670/?tt\_from=weixin&utm\_campaign=client\_share&wxshare\_count=1&timestamp=1565621931&app=news\_article&utm\_source=weixin&utm\_medium=toutiao\_ios&req\_id=20190812225850010152017067601B19E&group\_id=6723563574100230670

1. 基本数据类型的大小及其包装类

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 基本类型 | boolean | byte | char | short | Int | long | float | double |
| 二进制位数 | 1 | 8 | 16 | 16 | 32 | 64 | 32 | 64 |
| 封装器类 | Boolean | Byte | Character | Short | Integer | Long | Float | Double |

1. Java数据类型之引用型数据类型：类(class)、接口(interface)、数组
2. 自动装箱
3. 装箱就是自动将基本数据类型转换为包装器类型；拆箱就是自动将包装器类型转换为基本数据类型。
4. Integer total = 99; 实际系统执行了：   
   Integer total = Integer.valueOf(99);

注：Integer.valueOf(int i) 如果i小于-128或者大于等于128，就创建一个Integer对象，否则直接返回cashe [i + 128]。cashe 是Integer中的静态Integer数组，保存了value为-127到128之间的Integer。

1. int totalprim = total; 实际系统执行了：   
   int totalprim = total.intValue();
2. switch 语句
3. switch中的变量类型可以是： byte、short、int 或者 char。从 Java SE 7 开始，switch 支持字符串 String 类型了。
4. case 语句中的值的数据类型必须与变量的数据类型相同，而且只能是常量或者字面常量。
5. equals与==的区别

参考：<https://www.cnblogs.com/zjc950516/p/7877511.html>

1. 如果没有重写equals 方法，equals与==是等效的；
2. String类中
3. == 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。
4. equals用来比较的是两个对象的内容是否相等
5. Java常量池

参考：<https://www.jianshu.com/p/c7f47de2ee80>

1. 用final修饰的成员变量表示常量，值一旦给定就无法改变；
2. final修饰的变量有三种：静态变量、实例变量和局部变量；
3. 静态常量池和运行时常量池；
4. 静态常量池：\*.class文件中的常量池，class文件中的常量池不仅仅包含字符串(数字)字面量，还包含类、方法的信息；
5. 运行时常量池：jvm虚拟机在完成类装载操作后，将class文件中的常量池载入到内存中，并保存在方法区中，我们常说的常量池，就是指方法区中的运行时常量池。
6. 常量池是为了避免频繁的创建和销毁对象而影响系统性能，其实现了对象的共享；
7. Object对象公用方法有：equals()、hashCode()、toString()、getClass()、notify()、notifyAll()和wait()；
8. Java四种引用：强引用、软引用、弱引用和虚引用；

参考：<https://blog.csdn.net/swebin/article/details/78571933>

<https://blog.csdn.net/junjunba2689/article/details/80601729>

<https://www.cnblogs.com/dudadi/p/8067671.html>

1. Hashcode的作用:

参考：<https://www.cnblogs.com/NathanYang/p/9427456.html>

1. 主要是为了查找的快捷性，用来在散列存储结构中确定对象的存储地址的；
2. 为了减少equals方法（equals比较两对象是否指向同一内存地址）的调用次数，从而提高程序效率；
3. equals()方法判断两个对象相等，则两个对象的hashCode()也应该相等；
4. 两个对象的hashCode()相等，equals()方法判断两个对象不一定相等。
5. 为什么重载hashcode，当类需要放在HashTable、HashMap、HashSet等等hash结构的集合时才会 重载hashCode；保证equals()相等的对象，在集合类中只存在一份；
6. 别Vector、ArrayList、LinkedList的区别

|  |  |  |  |
| --- | --- | --- | --- |
|  | Vector | ArrayList | LinkedList |
| 数据结构 | 数组 | 数组 | 双向链表 |
| 线程同步 | 线程安全 | 非线程安全 | 非线程安全 |
| 扩容 | 增加100%（默认为10） | 增加50%（默认为0， jdk1.5之前默认为10） | 无限制 |
| 增加、删除、访问 | 随机访问快  插入和删除慢，需要复制并移动插入、删除位置之后的元素 | 随机访问快  插入和删除慢，需要复制并移动插入、删除位置之后的元素 | 随机访问慢，每次都需要从头开始  插入和删除高效 |

1. String、StringBuffer与StringBuilder的区别

<https://blog.csdn.net/dfdsggdgg/article/details/51347748>

|  |  |  |
| --- | --- | --- |
| String | StringBuffer | StringBuilder |
|  | 线程安全 | 非线程安全 |
| 1. String是final修饰的，不可变的； 2. 每次修改操作都是在内存中重新new一个对象出来。因为存储String内容的value[]是final类型； 3. 对于String str1=“test”; JVM编译期会检查常量池是否存在该字符串，若存在则不创建直接返回对象的引用；否则先在常量池中创建该字符串实例再返回实例的引用; 4. 对于String str1=new String(“test”); JVM编译期会检查常量池是否存在该字符串，若存在则不再在字符串常量池创建该字符串对象，而直接堆中复制该对象的副本，然后返回堆中对象的地址；若字符串不存在常量池中，就会实例化该字符串并且将其放到常量池中，再在堆中复制该对象的副本，然后返回堆中对象的地址； | 1. 修改值时不需要重新new对象； 2. 内容超过默认数组长度时，则扩容为原来字节数组的长度\*2+2。 3. 适当考虑下初始化大小，较少扩容的次数，提高代码的高效性 | |

1. try?catch?finally

<https://blog.csdn.net/qq_35632702/article/details/79572120>

1. Excption与Error包结构，OOM你遇到过哪些情况，SOF你遇到过哪些情况

<https://blog.csdn.net/GS_008/article/details/50951617>

<http://www.mamicode.com/info-detail-1091344.html>

1. Java(OOP)面向对象的三个特征与含义
2. 封装：封装就是要把属于同一类事物的共性(包括属性与行为)归到一个类中,把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏
3. 继承：继承父类的属性和方法，父子之间有共性,但还存在区别
4. 多态：相同方法在不同情形有不同表现形式；同一消息可以根据发送对象的不同而采用多种不同的行为方式。分为重载和覆盖。
5. Override和Overload的含义去区别

<https://blog.csdn.net/u011860731/article/details/48730617>

Override（覆盖）：

1. 父类与子类之间的垂直关系；
2. 覆盖的方法必须要和被覆盖的方法相同的的函数名、参数和返回值；
3. 覆盖的方法必须和被覆盖的方法所抛出的异常一致，或是其子类；
4. 方法被定义为final或访问权限是private不能被重写。

Overload（重载）：

1. 同一类中不同方法之间的水平关系；
2. 方法间通过不同参数样式（不同参数类型，不同参数个数，不同参数顺序）重载；
3. 不能通过访问权限、返回类型、抛出的异常进行重载；
4. Abstract与Interface类的区别

<https://blog.csdn.net/aptentity/article/details/68942916>

|  |  |
| --- | --- |
| **Abstract** | **Interface** |
| 单继承  类只能继承一个基类，但可以实现多个接口 | 多继承  接口可以继承多个接口 |
| 可以有构造方法 | 不能有构造方法 |
| 可以有抽象方法、普通方法和静态方法  抽象方法的访问类型可以是public，protected和默认类型 | 只能有抽象方法，访问类型为public abstract |
| 可以有任意变量 | 只能是public static final |

1. Static class与non static class的区别

<https://www.jb51.net/article/74838.htm>

1. 静态内部类不需要有指向外部类的引用。非静态内部类需持有对外部类的引用，不能脱离外部类实体被创建。
2. 非静态内部类能够访问外部类的所有静态和非静态成员（包括私有的）。静态类不能访问外部类的非静态成员，只能访问外部类的静态成员（包括私有的）。
3. For和foreach效率对比

<https://blog.csdn.net/qq_40285302/article/details/79742409>

<https://blog.csdn.net/mlc1218559742/article/details/52712408>

**结论**：

1. 对于数组结构的数据，for循环比foreach效率稍高；
2. 对于链表结构的数据，foreach循环比for效率高。

**原因：**

1. ArrayList：是采用数组的形式保存对象的，将对象放在连续的内存块中，插入和删除比较麻烦，查询比较方便。
2. LinkList：采用链表结构，将对象放在独立的空间中，且每个空间还保存下一空间的索引，插入和删除比较方便，但是查找很麻烦，要从第一个开始遍历。
3. for循环通过get(i)获取元素，而foreach是通过iterator实现的遍历；对于链表结构，get(i)每次都要从第一个元素开始遍历，foreach则是依次遍历，整个过程只需遍历一次，获取第i个元素可在上一元素的基础得到，不需要要重头遍历；
4. Java IO与NIO的区别

<https://blog.csdn.net/u013956074/article/details/80974190>

|  |  |  |  |
| --- | --- | --- | --- |
| IO | 面向流 | 阻塞IO |  |
| NIO | 面向缓冲区 | 非阻塞IO | Selector |

1. Java IO面向流意味着每次从流中读一个或多个字节，直至读取所有字节，没有被缓存，因此不能前后移动流中的数据。如果需要前后移动从流中读取的数据，需要先将它缓存到一个缓冲区。 Java NIO的缓冲导向方法略有不同。数据读取到缓冲区，需要时可在缓冲区中前后移动。增加了处理过程中的灵活性。但需要检查是否该缓冲区中包含所有您需要处理的数据。且需确保当更多的数据读入缓冲区时，不要覆盖缓冲区里尚未处理的数据。
2. Java IO的各种流是阻塞的，当一个线程调用read() 或 write()时，该线程被阻塞，直到有数据被读取或数据完全写入。该线程在此期间不能再干任何事情了。 Java NIO的非阻塞模式，使一个线程从某通道发送请求读取数据，它仅能得到目前可用的数据，如果目前没有数据可用时，就什么都不会获取。而不是保持线程阻塞，所以直至数据变的可以读取之前，该线程可以继续做其他的事情。 非阻塞写也是如此。一个线程请求写入一些数据到某通道，但不需要等待它完全写入，这个线程同时可以去做别的事情。 线程通常将非阻塞IO的空闲时间用于在其它通道上执行IO操作，所以一个单独的线程现在可以管理多个输入和输出通道（channel）。
3. Java NIO的选择器允许一个单独的线程来监视多个输入通道，可注册多个通道使用一个选择器，然后使用一个单独的线程来“选择”通道：这些通道里已经有可以处理的输入，或者选择已准备写入的通道。这种选择机制，使得一个单独的线程很容易来管理多个通道。
4. java反射的作用的原理

定义：在运行状态中，对于任意一个类，通过反射都能知道该类的所有属性和方法；对于任意一个对象，都能调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java的反射机制。class是一切反射的根源。

作用：只要给定类的名字，就可以通过反射机制来获取类的所有信息（属性，方法，构造方法），可以动态的创建对象和编译。

注：Class类的实例在java应用中主要体现jdk自带的和我们自定义类的实例及接口。枚举是一种特别的类（反编译后可知枚举是java的语法糖效），注解是一种特殊的接口。java的原生类型及数组、void关键字都是Class类的实例。

Class类没有构造函数，它的实例化主要由java虚拟机负责。

public static void main(String[] args) {

System.*out*.println(int.class.isPrimitive());// true

System.*out*.println(Collection.class.isInterface());// true

System.*out*.println(Object[].class.isArray());// true

System.*out*.println(int[].class.getComponentType());// int

System.*out*.println(IInterface.class.isInterface());// true

System.*out*.println(IInterface.class.isAnnotation());//true

System.*out*.println(void.class.isPrimitive());// true  
}  
public static @interface IInterface {  
 String value() default "";  
}



1. 泛型的特点
2. 泛型是JDK1.5引入的 ，主要应用于容器类 ；
3. 减少类型转换，提供编译期的类型安全；
4. 不能用基本数据类型做为类型参数 ；
5. 擦除：编译器在编译时擦除了所有类型相关的信息，所以在运行时不存在任何类型相关的信息。
6. DOM、SAX解析XML的原理与特点：<https://blog.csdn.net/cangchen/article/details/44034799>

|  |  |  |  |
| --- | --- | --- | --- |
| 区 别 | DOM解析 | SAX解析 | Pull解析 |
| 操作 | 1. Dom文档对象模型 2. 将所有文件读取到内存中形成DOM树，占用内存大，若文件量过大，则无法使用 | 1. sax是基于xml事件驱动的“推”模型； 2. 顺序读入所需要的文件内容，不会一次性全部读取，占用内存小，不受文件大小的限制 3. 自动将事件推入事件处理器处理，一旦开始解析，就必须解析完成，无法中途主动结束 | 1. pull是基于xml事件驱动的“推”模型； 2. 顺序读入所需要的文件内容，不会一次性全部读取，占用内存小，不受文件大小的限制 3. 主动从解析器中获取事件，可以在程序中控制解析结束的位置 |
| 读写限制 | DOM树在内存中形成，可以随意存放或读取文件树的任何部分，没有次数限制，可以任意修改文件树 | 由于采用部分读取，只能对文件按顺序从头到尾读取XML文件内容，但不能修改 | 1. 由于采用部分读取，只能对文件按顺序从头到尾读取XML文件内容，但不能修改 2. 只能读取XML文件内容，但不能修改 |
| 复杂度 | 易于理解，易于开发 | 开发上比较复杂，需要用户自定义事件处理器 | 开发上比较复杂，需要用户自定义事件处理器 |
| 对象模型 | 系统为使用者自动建立DOM树，XML对象模型由系统提供 | 对开发人员更加灵活，可以用SAX建立自己的XML对象模型 | 对开发人员更加灵活，可以用SAX建立自己的XML对象模型 |

1. Java1.5与1.6, 1.7,1.8,1.9 新特性

<https://blog.csdn.net/fzy198926/article/details/78134100>

<https://blog.csdn.net/qq_33204709/article/details/78948650>

1. 设计模式：单例模式、工厂模式、适配器模式、责任链模式和观察者模式
2. JNI的使用

<https://www.jianshu.com/p/87ce6f565d37>

1. 线程和进程的区别
2. 多线程的作用
3. 线程创建的方式

<https://blog.csdn.net/m0_37840000/article/details/79756932>

1. 继承Thread类创建线程
2. 实现Runnable接口创建线程
3. 使用Callable和Future创建线程
4. 使用线程池例如用Executor框架
5. start()方法和run()方法的区别

<https://blog.csdn.net/kwame211/article/details/79030255>

<https://blog.csdn.net/dada360778512/article/details/6965790>

1. start()启动线程，真正实现了多线程运行，这时无需等待run方法体代码执行完毕而直接继续执行下面的代码。通过调用Thread类的 start()方法来启动一个线程，这时此线程处于就绪（可运行）状态，并没有运行，一旦得到cpu时间片，就开始执行run()方法，这里方法 run()称为线程体，它包含了要执行的这个线程的内容，Run方法运行结束，此线程随即终止。
2. run()方法只是类的一个普通方法而已，若直接调用Run方法，程序中依然只有主线程这一个线程，程序执行路径只有一条，还是要顺序执行，且要等待run方法体执行完毕后才可继续执行下面的代码，这样就没有达到多线程的目的。
3. sleep和wait的区别
4. Runnable接口和Callable接口的区
5. Callable可以返回一个类型V，而Runnable不可以；
6. Callable能够抛出checked exception,而Runnable不可以；
7. Runnable是自从java1.1就有了，而Callable是1.5之后才加上去的；
8. Callable和Runnable都可以应用于executors。而Thread类只支持Runnable；
9. Callable与executors联合在一起，在任务完成时可立刻获得一个更新了的Future。而Runable却要自己处理。
10. CyclicBarrier和CountDownLatch的区别

<https://blog.csdn.net/tolcf/article/details/50925145>

|  |  |
| --- | --- |
| CountDownLatch | CyclicBarrier |
| 减计数方式 | 加计数方式 |
| 计算为0时释放所有等待的线程 | 计数达到指定值时释放所有等待线程 |
| 计数为0时，无法重置 | 计数达到指定值时，计数置为0重新开始 |
| 调用countDown()方法计数减一，调用await()方法只进行阻塞，对计数没任何影响 | 调用await()方法计数加1，若加1后的值不等于构造方法的值，则线程阻塞 |
| 不可重复利用 | 可重复利用 |

1. volatile关键字的作用：依靠内存屏障保持内存可见性和防止指令重排序。

<https://www.cnblogs.com/monkeysayhi/p/7654460.html>

1. 线程安全：在拥有共享数据的多条线程并行执行的程序中，线程安全的代码会通过同步机制保证各个线程都可以正常且正确的执行，不会出现数据污染等意外情况。
2. 一个线程运行时发生异常会怎样：
3. 未处理的RuntimeException，发生异常后程序停止运行；
4. RuntimeException异常被捕获或抛出，则程序继续运行。
5. ThreadLocal

<https://www.cnblogs.com/xzwblog/p/7227509.html>

<https://www.jianshu.com/p/3c5d7f09dfbd>