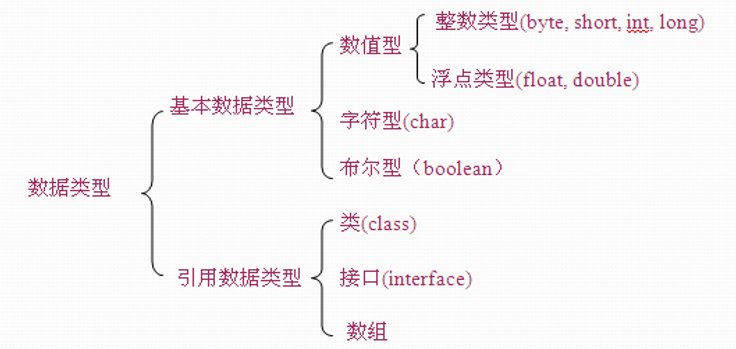
1. 数据类型

Java是一门强类型语言，声明变量时必须写数据类型



1. 局部变量

在方法或者语句块({})中声明的变量称为局部变量，先声明后赋值再使用，在作用域内声明的变量只能在作用域内被访问。作用域可以嵌套，此时内部作用域可以访问外部作用域的变量。

1. 常量

常量只能初始化一次，用final修饰。

final 数据类型 变量 = 初始值；

1. 运算符

赋值运算符（=），算数运算符（+, -, \*, /, %,++,--）,关系运算符（>,>=,<,<=,==,!=），逻辑运算符（&&，||，！），条件运算符(（5>4）? 1 : 0)，位运算符（按位与&，按位或|，按位异或^，取反~，左移<<,右移>>）。

1. 重载

一个类中方法名称相同，参数列表不同的方法构成方法重载

同一类中，方法名相同，参数列表不同，和返回值无关。

1. 递归

方法之间可以相互调用，在调用过程中，有一类特殊情况，就是方法自身调用自身。优点：简单的程序，缺点：占用大量的系统堆栈，内存耗用多。

1. 数组

数组是想同类型数据的有序集合。数组一旦被创建，长度不可改变。引用数据类型。

二维数组本质上是一个一维数组，每个元素是一个一维数组。

1. 内存结构

栈区（stack）：一般存储基本数据类型，栈区空间相对较小，栈区的变量使用完后系统自动回收。

堆区（heap）：一般存储引用数据类型，堆区空间相对较大，堆区的变量使用完后要回收，垃圾回收机制帮开发者回收。

1. static关键字

表示静态，可以修饰变量和方法。Static修饰变量称为静态变量，修饰方法称为静态方法。

静态变量：static修饰的变量分配在方法区，归类所有，该类的所有势力都可以访问该静态变量。

静态方法：用static修饰的方法称为静态方法，归类所有，不能访问实例变量、this、实例方法。实例方法可以访问静态变量或者静态方法。

1. 类和对象的概念

类：是一组具有相同特征和行为的对象的抽象描述；

对象：是类的一个具体实现，是客观存在的一个具体事物/实体，具有唯一性和独立性。

1. 封装

在面向对象程式设计中，封装是指将一种抽象性函式接口的实现细节部分包装、隐藏起来的方法，封装可以被认为是一个保护屏障，防止该类的代码和数据被外部类定义的代码随机访问。

1. 继承

子类继承父类，子类拥有父类非私有的属性和方法，使用extends关键字。

单根性：在Java中，类的继承是单继承，一个类只能有一个直接父类。

传递性：C继承于B，B继承于A，C也具有A的特性和行为。

1. super关键字

Super表示父类，它只是一个简单的关键字，用于引用父类，super没有内存地址。

1. 访问修饰符

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 访问修饰符 | 本类 | 同包子类 | 同包其他类 | 不同包子类 | 不同包其他类 |
| public | ✔ | ✔ | ✔ | ✔ | ✔ |
| protected | ✔ | ✔ | ✔ | ✔ | ✘ |
| 默认 | ✔ | ✔ | ✔ | ✘ | ✘ |
| private | ✔ | ✘ | ✘ | ✘ | ✘ |

Public：公共的，访问权限最大。

Protected：受保护的，子类可见，同包可见

默认：同包可见

Private：私有的，本类可见

1. 重写

当子类继承父类的方法不能满足自身需要的时候，子类可以重写父类的同名方法。

继承关系，方法名相同，参数列表相同，返回值要么和父类相同，要么是其子类。

1. 抽象类

当一个类定义一个方法根据自身特性无法实现时，可以把该方法声明为抽象方法(abstract method),只有声明，没有方法体。该抽象方法所在的类称为抽象类(abstract class)。抽象类不能实例化。

1. 多态

一个接口，多种实现，同一事物表现出的多种形态。

继承中，子类重写了或实现了父类的A方法，父类对象引用子类类型，在运行是执行的是子类的A方法。增强了系统的可拓展能力，从而提高了可维护性。

1. 接口概念

接口是一种引用数据类型，使用interface声明接口

1. 接口的特性

接口中可以声明属性，定义的所有变量都是static final类型的

接口中的方法都是公共抽象方法

接口不能拥有构造方法，不能用于创建对象

接口可以多继承，一个接口可以继承多个其它接口

一个类可以实现一个或者多个接口，该类称为接口的实现类(implement class)，实现类必须实现接口中的所有(抽象)方法

一个类只能继承一个父类，同时实现多个接口，继承在前，实现在后

1. 接口总结

实现类实现接口，必须实现接口中定义的抽象方法。

方法即行为，表示一种功能

接口定义了一套功能，实现类必须实现这些功能->实现类能力得到拓展

1. 接口应用

接口中定义的一系列方法表示的是一种种能力，接口让实现类实现这些能力，实现类的能力得到拓展和升级。实现类根据自身特性实现接口中定义的方法。

1. 接口实现多态

接口类型引用实现类对象。接口定义的方法被实现类实现，通过接口引用实现类，调用接口中的方法时，执行的是实现类实现的方法。

1. 面向接口编程

接口表示一种约定(协议)，约定了实现类应该具备(has a)的能力。

实现类必须实现接口中所有的方法，所以接口规范了实现类的行为

面向接口编程时，接口定义方只关心实现类是否具备接口所定义的能力，而实现类如何被该能力接口定义则不关心。

1. 抽象类和接口的异同比较

·抽象类和接口都是引用数据类型，他们都不能创建对象。

·都可以定义抽象方法，都可以实现多态，但是抽象类可以定义非抽象方法，而接口中定义的都是抽象方法

·抽象类和接口都具有传递性，抽象类是单继承，接口是多继承

·在概念上，都可以重写抽象方法，子类重写抽象类，实现类实现接口

·抽象类和子类解决的是模块内的问题（代码重用,重写,多态），而接口解决的是模块间问题=>高内聚，低耦合。接口可以用解耦模块。

1. Object

Object类是所有类的根类，如果一个类没有显示继承另一个类，那么继承Object。

toString()返回对象的字符串表示形式

equals(Object obj)用于判断两个对象内容属性是否相等。==是内存地址比较。

1. 成员内部类

一个内部类作为一个外部类的成员而存在，此时该内部类称为外部类的成员内部类。

1. 静态内部类

静态内部类也可以作为外部类的静态成员而存在，可以访问外部类的静态成员。

1. 方法内部类

如果一个类出现在方法中，称为方法内部类。可以直接访问方法的局部变量，不能在方法内部类中修改外部类方法的局部变量。

1. 匿名内部类

如果一个类未来只会使用一次，可以定义成匿名类。匿名类和内部类结合使用形成匿名内部类，匿名内部类一定存在：实现(接口)。

1. 异常

异常是指在程序的运行过程中所发生的不正常的事件，它会中断正在运行的程序。Java编程语言使用异常处理机制为程序提供异常处理的能力，处理完继续执行。

1. try-catch-finally

try{}代码用于执行可能存在异常的代码，catch{}代码块用于捕获并处理异常。getMessage()获取异常的描述。toString()返回异常的类型和异常描述。

finally{}代码块用于回收资源(关闭文件、关闭数据库、关闭管道)的代码块，finally代码块不管是否出现异常，都必须执行，除非jvm正常退出。

1. 异常分类

·Throwable类是Java语言中所有错误或异常的父类，Throwable有两个子类Error和Exception。

·Error：仅靠程序本身无法恢复的严重错误，例如jvm崩溃、内存耗尽。

·Exception：靠程序本身通过异常处理机制能处理的非正常情况。

·异常分为两类：

运行时异常：不要求程序必须做处理，RuntimeException是所有运行时异常的父类。

检测时异常：程序必须处理该类异常，

1. 常见异常

·运行时异常：

InputMismatchException:控制台输入不匹配异常

ArithmeticException:算术计算异常。例如:除数为0

ArrayIndexOutOfBoundsException:数组下标越界

NullPointerException:空指针异常

IllegalArgumentException:方法接收到非法参数

ClassCastException:强制类型转换异常

NumberFormatException:数组格式化异常。如把"abc"转换成数字

NoSuchElementException: 表明枚举中没有更多的元素

ConcurrentModificationException:并发操作异常

·检测时异常：

ClassNotFoundException:无法找到类异常

SQLException:数据库操作异常。

FileNotFoundException:文件未被发现

IOException:IO异常

ParseException:解析异常

TimeoutException:超时异常

InterruptedException:中断异常。

1. 声明异常

开发者在定义方法时，事先知道方法在调用时会出现异常，但不知道该如何处理，此时可以在改方法上声明异常，表示该方法在调用时会出现异常。在java中使用throws声明异常，可以声明多个。

1. 抛出异常

当系统异常满足不了开发需要时，开发者可以自行根据需要自行抛出异常，throw用于手动抛出异常。

1. 自定义异常

当JDK中的异常类型不能满足程序的需要时，可以自定义异常类。

确定异常类型，继承Exception或者RuntimeException

编写自定义异常类，并实现构造方法

在方法需要的地方手动声明并抛出异常。

1. 包装类

包装类就是把基本数据类型封装到一个类中，并提供相应的属性和方法更方便的去操作基本数据类型。

1. 自动拆箱和自动装箱

Jvm编译时会把基本数据类型自动转化成对应的包装类，称为自动装箱(auto-boxing)

Jvm编译时会把包装类自动转化成对应的基本数据类型，称为自动拆箱(auto-unboxing)

1. 为什么使用集合

数组不能扩容，数组在操作元素时需要大量的移动元素，逻辑需要开发者代码实现。实际开发中，需要自动扩容的容器。

1. Collection简介

Collection表示集合的顶层接口，它表示一种容器，容器中可以存储一组对象，存储的对象成为集合或者容器的元素。List可重复的，有序的数据。Set唯一的，无序的数据。

1. ArrayList/Vector

·ArrayList底层数据结构是数组，是List接口的容量大小可变的数组。

扩容规则：默认10个，oldCapacity + oldCapacity / 2.

线程不安全，不保证同步，效率高。

·Vector底层数据结构是数组，是List接口大小可变数组的实现。

扩容规则：默认10个，有增长因子按因子增加，没有按oldCapacity 增加。

线程安全，保证同步，效率低。

1. LinkList

LinkList底层数据结构是链表，是List接口的链表实现。

1. 泛型

泛型是程序设计语言的一种特性。允许程序员在强类型程序设计语言中编写代码时定义一些可变部分，那些部分在使用前必须作出指明。

将类型参数化以达到代码复用提高软件开发工作效率的一种数据类型。泛型类是引用类型，是堆对象，主要是引入了类型参数这个概念。

泛型只工作在编译期，运行时感知不到泛型的存在。

1. Iterator和ListIterator

Iterator和ListIterator都可以用于迭代。不同的是

·ListIterator继承于Iterator

·Iterator只能正向遍历，ListIterator可以正向和逆向

·Iterator不支持并发修改(add)，ListIterator支持并发修改(add)

ConcurrentModificationException 并发修改异常。表示在遍历集合是不能对集合进行添加元素的操作

1. Set接口和其实现类

·Set接口约定存储唯一、无序的元素。

·HashSet 底层数据结构是哈希表,它是Set接口的实现类。

·LinkedHashSet 底层数据接口是哈希表+链表，链表用于维持**添加**次序。继承于HashSet。

·TreeSet的底层数据结构是二叉树，它是Set接口的实现类。

46、Map接口

·Map 接口表示映射，是键值对(key-value)的容器.

·Map中的一个元素(Entry)包含两部分，一部分是key，一部分是value。开发过程中，都是通过key来存取value值，key不能重复。

·HashMap key的底层数据结构是哈希表。通过Key存储value时，一定要实现key的hashCode()和equals()方法。

·LinkedHashMap key的底层数据结构是哈希表+链表，链表维持添加次序。LinkedHashMap 元素中的key一定要实现hashCode()和equals()方法

·TreeMap key的底层数据结构是二叉树。向TreeMap容器中添加元素时，一定要提供key的内部比较器和外部比较器。

1. IO流

**概念：**流是一连串流动的数据单元(字符、字节)，以先进先出的方式发送的信息的通道中。

**按流向**：输入流(OutputStream和Writer作为基类)和输出流(InputStream和Reader作为基类)

**按数据单元**：字节流(二进制文件)和字符流

**按功能**：节点流(可以直接从数据源或目的地读写数据)和处理流(不直接连接到数据源或目的地，是其他流进行封装。目的主要是简化操作和提高性能)

1. 进程和线程

**进程：**程序的一次执行产生进程。进程是动态向前的。

**线程：**线程的出现为了解决实时性问题。

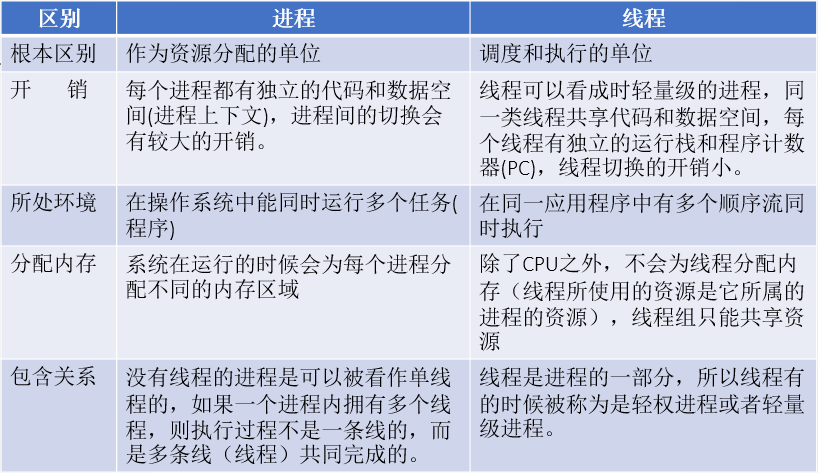
线程是进程的细分，通常，在实时性操作系统中，进程会被划分为多个可以**独立运行**的子任务，这些子任务被称为线程，多个线程配合完成一个进程的任务。

**总结**：[1] 线程再一次提高CPU利用率

[2] 线程包含在进程中，是对进程任务的细分。线程共享进程资源(内存资源等)

[3] 线程称为CPU调度的基本单位。进程称为操作系统资源分配的基本单位。

48、线程和进程的区别



1. 实现多线程的方式

**继承Thread：**java实现多线程可以继承Thread类并重写run方法，调用start开始执行线程。**实现Runable接口**：Runnable让一个类具有在线程中执行的能力。

**总结：**

[1] 继承Thread的线程类不能再继承其他类。实现Runnable接口的线程类还可以继承其他类。

[2] 实现Runnable接口方便多个线程共享资源。多个线程可以访问同一份资源。

49、线程生命周期(线程状态)

**·新生状态**：用new关键字建立一个线程后，该线程对象就处于新生状态。

处于新生状态的线程有自己的内存空间，通过调用start()方法进入就绪状态。

**·就绪状态**：处于就绪状态线程具备了运行条件，但还没分配到CPU，处于线程就绪队列，等待系统为其分配CPU。当系统选定一个等待执行的线程后，它就会从就绪状态进入执行状态，该动作称为“CPU调度”。

**·运行状态**：在运行状态的线程执行自己的run方法中代码,直到等待某资源而阻塞或完成任务而死亡。如果在给定的时间片内没有执行结束，就会被系统给换下来回到等待执行状态。

**·阻塞状态：**处于运行状态的线程在某些情况下，如执行了sleep(睡眠)方法，或等待I/O设备等资源，将让出CPU并暂时停止自己运行，进入阻塞状态。在阻塞状态的线程不能进入就绪队列。只有当引起阻塞的原因消除时，如睡眠时间已到，或等待的I/O设备空闲下来，线程便转入就绪状态，重新到就绪队列中排队等待，被系统选中后从原来停止的位置开始继续执行。

**·死亡状态**：死亡状态是线程生命周期中的最后一个阶段。线程死亡的原因有三个，

一个是正常运行的线程完成了它的全部工作；另一个是线程被强制性地终止，如通过stop方法来终止一个线程【不推荐使用】；三是线程抛出未捕获的异常。