Android四大组件

- Service
  - 启动方法：startService，bindService
    - onCreate（第一次启动）->onStartCommand()->onStart()(这两个方法每次都会调用)->onDestroy();
  - bindService
    - onCreate（）->onBind()->onUnbind()->onDestroy()(均只调用一次)
- Activity
  - Standard
  - SingleTask
  - SingleInstance
  - SingleTop
- ContentProvider
  - 数据共享
  - 增删改查
- BroadcastReceiver
  - 两种注册方式
    - 静态注册，AndroidManifest中注册
    - 动态注册，Context.registerReceiver()，需要反注册

NDK

- 原生开发工具包是一组可以让您在Android应用中利用C和c++代码的工具,可用以从您自己的源代码构建,或者利用现有的预构建库.

java权限修饰符

- private，在同一个类中才能直接访问。
- public，都可以访问
- 不写关键字，（default）在同一个包中能访问
- protected，不同包的子类中可以访问
- volatile关键字的作用很简单，就是一个线程在对主内存的某一份数据进行更改时，改完之后会立刻刷新到主内存。并且会强制让缓存了该变量的线程中的数据清空，必须从主内存重新读取最新数据。这样一来就保证了可见性

深拷贝与浅拷贝

浅拷贝只是对指针的拷贝，拷贝后两个指针指向同一个内存空间，深拷贝不但对指针进行拷贝，而且对指针指向的内容进行拷贝，经深拷贝后的指针是指向两个不同地址的指针。

- 1.当函数的参数为对象时，实参传递给形参的实际上是实参的一个拷贝对象，系统自动通过拷贝构造函数实现；
- 2.当函数的返回值为一个对象时，该对象实际上是函数内对象的一个拷贝，用于返回函数调用处。
- 3.浅拷贝带来问题的本质在于析构函数释放多次堆内存，使用std::shared_ptr，可以完美解决这个问题。

String StringBuilder StringBuffer

- String的值是不可变的，这就导致每次对String的操作都会生成新的String对象。
- StringBuffer 和 StringBuilder 类的对象能够被多次的修改，并且不产生新的未使用对象。

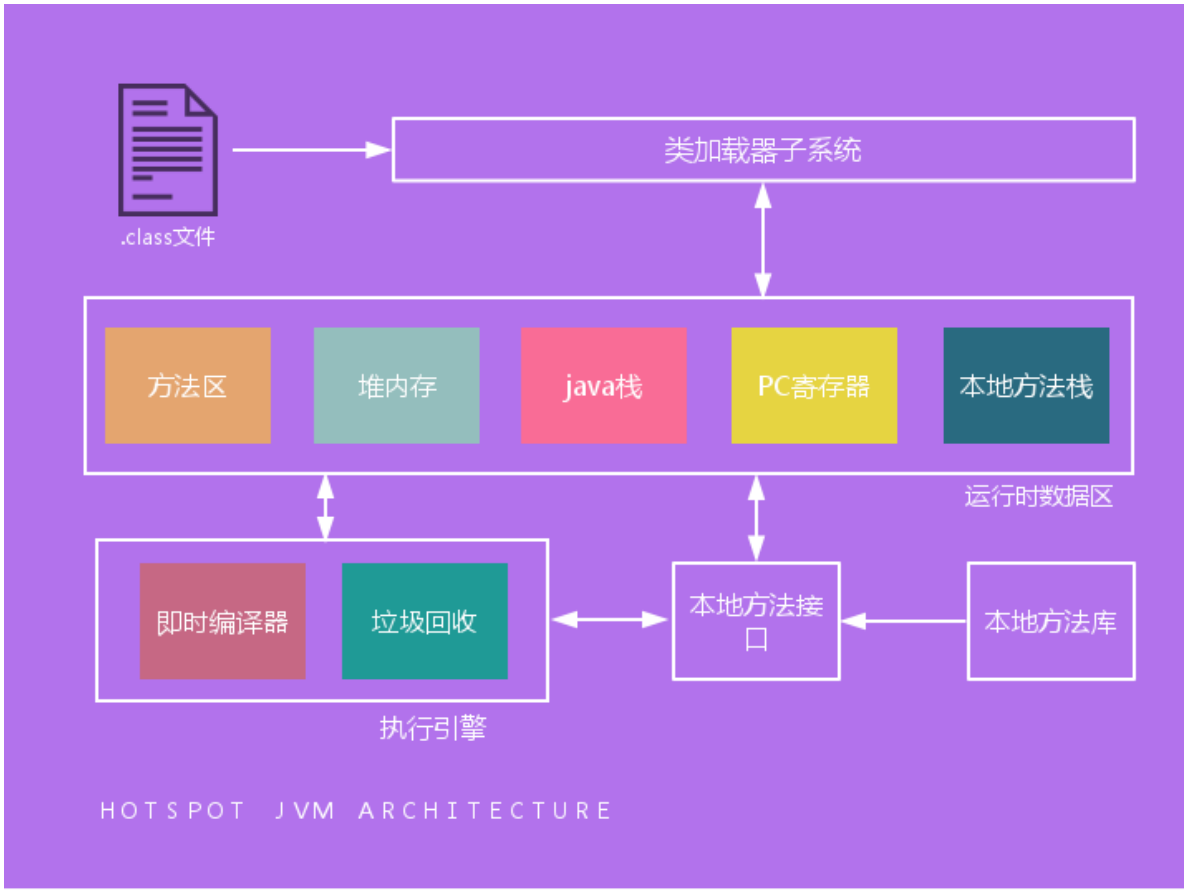
- StringBuilder：可变字符序列，效率高，线程不安全。
- StringBuffer：可变字符序列，效率低，线程安全。
- StringBuffer线程安全是因为这个类中有大量的synchronized关键字修饰。
- String在修改多次以后会在内存中留下大量副本内存，极大的影响性能。

synchronized关键字与lock

- 底层原理属于jvm层面、
- synchronized关键字解决的是多个线程之间访问资源的同步性，synchronized关键字可以保证被它修饰的方法或者代码块在任意时刻只能有一个线程执行。
- JVM 是通过进入、退出 对象监视器(Monitor) 来实现对方法、同步块的同步的，而对象监视器的本质依赖于底层操作系统的 互斥锁(Mutex Lock) 实现。

JAVA底层实现

图如下：



- Java内存中的可见性，原子性和有序性
  - 可见性：是指线程之间的可见性，一个线程修改的状态对另一个线程是可见的。
  - 原子性：原子是世界上的最小单位，具有不可分割性。在 Java 中 synchronized 和在 lock、unlock 中操作保证原子性。
  - Java 语言提供了 volatile 和 synchronized 两个关键字来保证线程之间操作的有序性，volatile 是因为其本身包含“禁止指令重排序”的语义，synchronized 是由“一个变量在同一个时刻只允许一条线程对其进行 lock 操作”这条规则获得的，此规则决定了持有同一个对象锁的两个同步块只能串行执行。
  - 当把变量声明为volatile类型后，编译器与运行时都会注意到这个变量是共享的，因此不会将该变量上的操作与其他内存操作一起重排序。volatile变量不会被缓存在寄存器或者对其他处理器不可见的地方，因此在读取volatile类型的变量时总会返回最新写入的值。
    - 当对非 volatile 变量进行读写的时候，每个线程先从内存拷贝变量到CPU缓存中。如果计算机有多个CPU，每个线程可能在不同的CPU上被处理，这意味着每个线程可以拷贝

到不同的 CPU cache 中。而声明变量是 volatile 的，JVM 保证了每次读变量都从内存中读，跳过 CPU cache 这一步。

内部类与静态内部类

- 静态内部类创建对象的时候，独立于外部类及其对象，就好像它是一个独立的类，可以和外部类一样使用。
- 内部类创建对象的时候，不能独立于外部类，必须要先创建外部类的对象，然后再用这个对象来new出内部类的对象。
- 内部类不可以有非常量的静态成员

抽象类与接口

- 在面向对象的概念中，所有的对象都是通过类来描绘的，但是反过来，并不是所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就是抽象类。
- 不能实例化对象，必须被继承才能使用
- 一个类只能继承一个抽象类
- 在JAVA编程语言中接口是一个抽象类型，是抽象方法的集合，接口通常以interface来声明。一个类通过继承接口的方式，从而来继承接口的抽象方法。
- 除非实现接口的类是抽象类，否则该类要定义接口中的所有方法。
- 静态方法：静态方法不能访问非静态的数据和方法，因为这两项都依赖于具体的实例，而静态方法在对象实例化之前就已经被JVM装载，而类中的实例变量和实例对象必须在对象开辟堆内存之后才能使用。

JAVA多态

- 多态是同一个行为具有多个不同表现形式或形态的能力。多态就是同一个接口，使用不同的实例而执行不同操作，如图所示
- 多态的实现方式：重写，接口，抽象类和抽象方法。
- 重写：
  - 重写是子类对父类的允许访问的方法的实现过程进行重新编写, 返回值和形参都不能改变。即外壳不变，核心重写！
  - 当需要在子类中调用父类的被重写方法时，要使用 super 关键字。
  - 重载(overloading) 是在一个类里面，方法名字相同，而参数不同。返回类型可以相同也可以不同。
- 注：java里没有虚函数，所有函数都具有虚函数的性质，除非加了final关键字编程非虚函数。

  (1)方法重载是一个类中定义了多个方法名相同,而他们的参数的数量不同或数量相同而类型和次序不同,则称为方法的重载(Overloading)。

  (2)方法重写是在子类存在方法与父类的方法的名字相同,而且参数的个数与类型一样,返回值也一样的方法,就称为重写(Overriding)。

  (3)方法重载是一个类的多态性表现,而方法重写是子类与父类的一种多态性表现。

malloc的底层实现

- 关于malloc获得的内存空间是否连续：

  对于虚拟空间：答案是连续的，malloc()分配后返回空间可以被看做是一块连续的地址

  对于物理空间：答案是不连续的，一块连续的内存空间是由若干不连续的物理页面拼凑而成的
- malloc分配的内存位于堆 上， malloc通过底层的系统调用brk()（brk()用来标识堆的起始地址）和mmp()实现。
- malloc()并不是直接在操作系统申请内存，而是在进程内存空间中申请内存，如果够用，则直接在进程内存空间申请，如果不够用，则会通过系统调用向操作系统申请内存。

iostream

- cout语句一般格式：

  cout<<expression1<<expression2<<expression3...<<expressionn;

- cin语句一般格式：

  cin>>变量1>>变量2>>变量3>>.....>>变量n;

- 常用函数：

  cin.get();cin.getline();cin.getloc();

C++ map和unordered_map

- map是有序的，二unordered_map是无序的，unordered_map查找更省时。

重写和重载

- java
  - 同名的方法如果有不同的参数列表（**参数类型不同、参数个数不同甚至是参数顺序不同**）则视为重载。同时，重载对返回类型没有要求，可以相同也可以不同，但**不能通过返回类型是否相同来判断重载**。
  - 其实就是**在子类中把父类本身有的方法**重新写一遍。子类继承了父类原有的方法，但有时子类并不想原封不动的继承父类中的某个方法，所以**在方法名，参数列表，返回类型(除过子类中方法的返回值是父类中方法返回值的子类时)都相同的情况下，** 对方法体进行修改或重写，这就是重写。但要注意**子类函数的访问修饰权限不能少于父类的**。
- C++
  - 重载是指同一可访问区内被声明的几个具有不同参数列（参数的类型，个数，顺序不同）的同名函数，根据参数列表确定调用哪个函数，重载不关心函数返回类型。
  - 是指派生类中存在重新定义的函数。其函数名，参数列表，返回值类型，所有都必须同基类中被重写的函数一致。只有函数体不同（花括号内），派生类调用时会调用派生类的重写函数，不会调用被重写函数。重写的基类中被重写的函数必须有virtual修饰。

C++内存管理

- 在C++中，虚拟内存分为代码段、数据段、BSS段、堆区、文件映射区以及栈区六部分。
- 代码段:包括只读存储区和文本区，其中只读存储区存储字符串常量，文本区存储程序的机器代码。
- 数据段：存储程序中已初始化的全局变量和静态变量
- bss 段：存储未初始化的全局变量和静态变量（局部+全局），以及所有被初始化为0的全局变量和静态变量。
- 堆区：调用new/malloc函数时在堆区动态分配内存，同时需要调用delete/free来手动释放申请的内存。
- 映射区:存储动态链接库以及调用mmap函数进行的文件映射
- 栈：使用栈空间存储函数的返回地址、参数、局部变量、返回值

Android细节

-