Java基础的介绍

https://www.cnblogs.com/rove888/p/9936109.html

Android 加密解密

<https://www.jianshu.com/p/a0cf938ce5ab>

Java基础

Wait和Sleep的区别

1. wait方法是在Object上的，而sleep方法是在Thread上，当调用Thread.sleep时，并不能改变对象的状态，因此也不会释放锁。wait方法释放了锁。

sleep不出让系统资源；wait是进入线程等待池等待，出让系统资源，其他线程可以占用CPU。一般wait不会加时间限制，因为如果wait线程的运行资源不够，再出来也没用，要等待其他线程调用notify/notifyAll唤醒等待池中的所有线程，才会进入就绪队列等待OS分配系统资源。Sleep(millseconds)可以用时间指定使它自动唤醒过来，如果时间不到只能调用interrupt()强行打断。

Thread.Sleep(0)的作用是“触发操作系统立刻重新进行一次CPU竞争”。

1. 使用范围：wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用

synchronized(x){

x.notify()

//或者wait()

}

3.sleep必须捕获异常，而wait，notify和notifyAll不需要捕获异常。

concatMap 和 flatMap 最大的区别是 concatMap 发射的数据集是有序的，flatMap 发射的数据集是无序的

不同点： 模块化（lib）主要解决问题是重用问题，不强调单独编译

组件化（lib）主要解决问题是功能拆分，强调单独编译

插件化（application）是所有组件都为apk的特殊组件化，特点可热更新

通讯方式的不同：模块化相互引入，需要引入需要的module

组件化通讯方式分为隐式和路由。

插件化本身是不同进程，因此是binder机制进程间通讯。

onSaveInstanceState()和ViewModel的不同点：

onSaveInstanceState()方法只能保存少量的、能支持序列化的数据，但是onSaveInstanceState()方法可以持久化页面的数据

ViewModel能支持页面中所有的数据，但是需要注意的是，ViewModel不支持数据的持久化，当页面被彻底销毁时，ViewModel及其持有的数据就不存在

ViewModel原理：

1.Activity实现了ViewModelStoreOwner接口，创建了ViewModelStore对象。

2.当Activity意外销毁时，onRetainNonConfigurationInstance函数被回调，在此函数中对ViewModelStore对象进行了保存。

3.当Activity重建时，onCreate方法中会先获取getLastNonConfigurationInstance，如果其中的ViewModelStore对象不为空，就直接引用，不再重新创建ViewModelStore对象了。

ViewModelStore就是个HashMap，通过key来获取ViewModel对象。

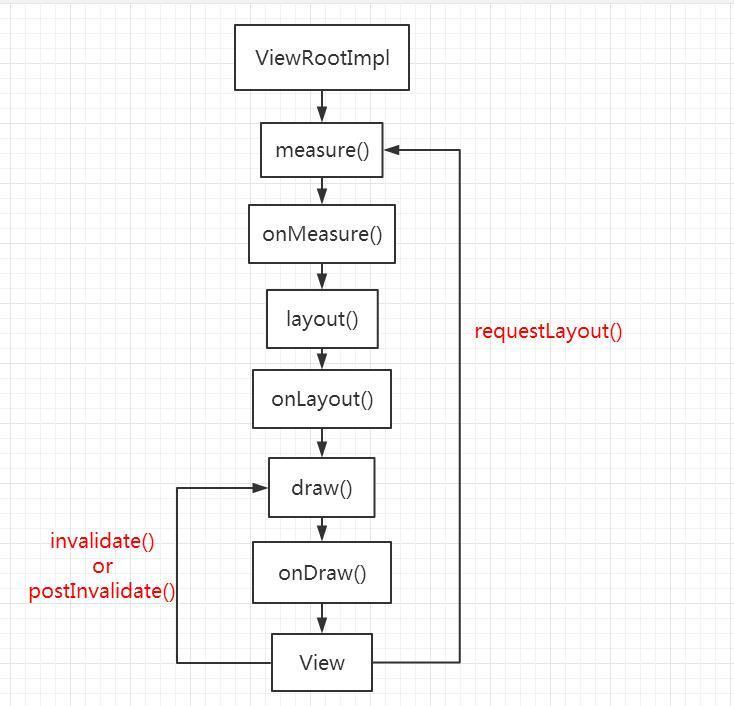
ViewModleProvider的get方法实现也比较简单，总结一下就是：

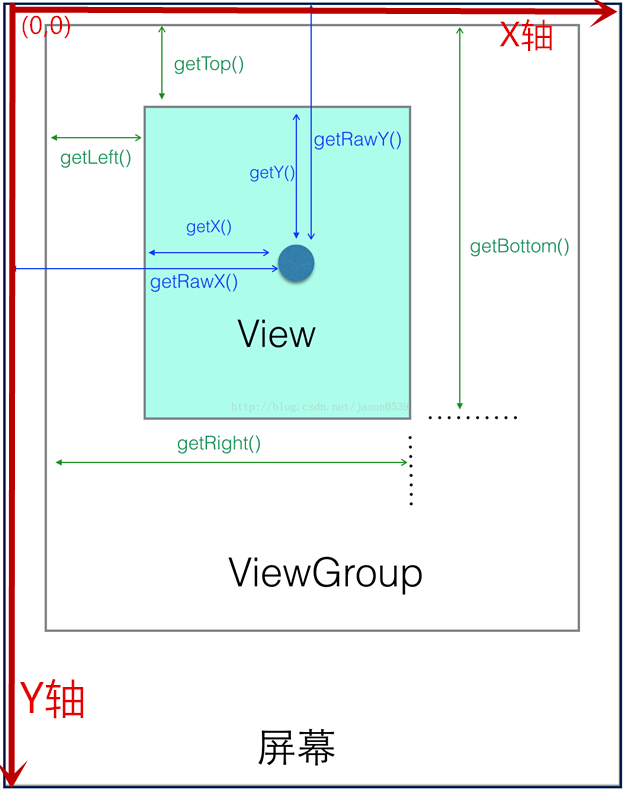
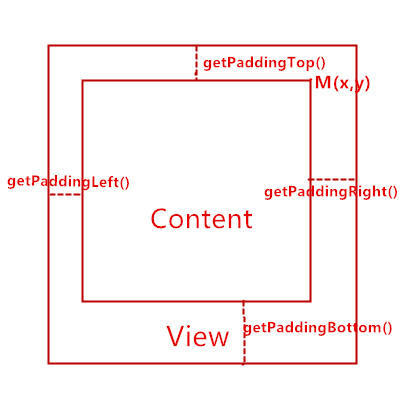
1. 使用ViewModel Class的canonicalName作为ViewModel在ViewModelStore中的唯一标识。
2. 通过唯一标识，先查询一下ViewModelStore中是否有该ViewModel对象，如果有则直接返回。
3. 如果ViewModelStore中没有该ViewModel对象，则通过Factory工厂类反射创建出ViewModel对象，存入ViewModelStore中，并返回给调用者。

invalidate 在UI线程中使用（主线程）。

postInvalidate 在非UI线程中通知重绘。

requestLayout 会标记当前View及父容器，同时逐层向上提交，直到ViewRootImpl处理该事件，ViewRootImpl会调用三大流程，从measure开始，对于每一个含有标记位的view及其子View都会进行测量、布局、绘制。





### 安卓原生与H5混合开发

1.在安卓原生代码中声明

private class JsInterface {  
        // 安卓原生与h5互调方法定义  
        @JavascriptInterface //js接口声明  
        public void takePhoto() {  
          //安卓原生处理逻辑  
        }  
    }

再通过webView控件调用

myWebView.addJavascriptInterface(new JsInterface(), "android");

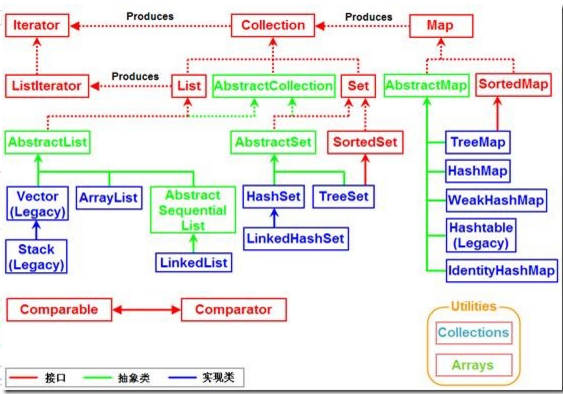
在h5的html中使用window.android.takePhoto();就可以调用上述声明的方法。

1. 安卓原生调用H5方法是

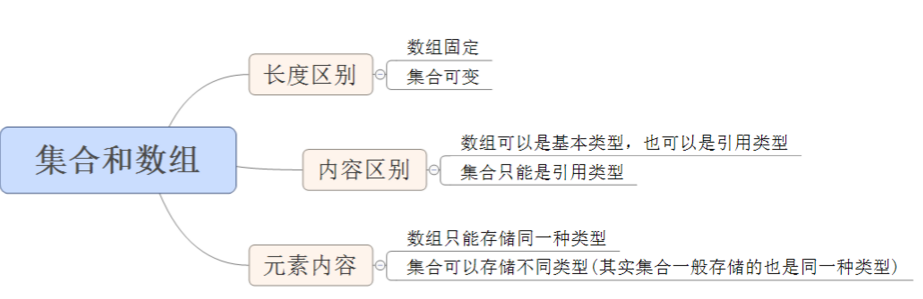
String method = "javascript:testResult('" + content + "')";  
myWebView.loadUrl(method);

通过webview加载，当然testResult()方法需要在html中写好

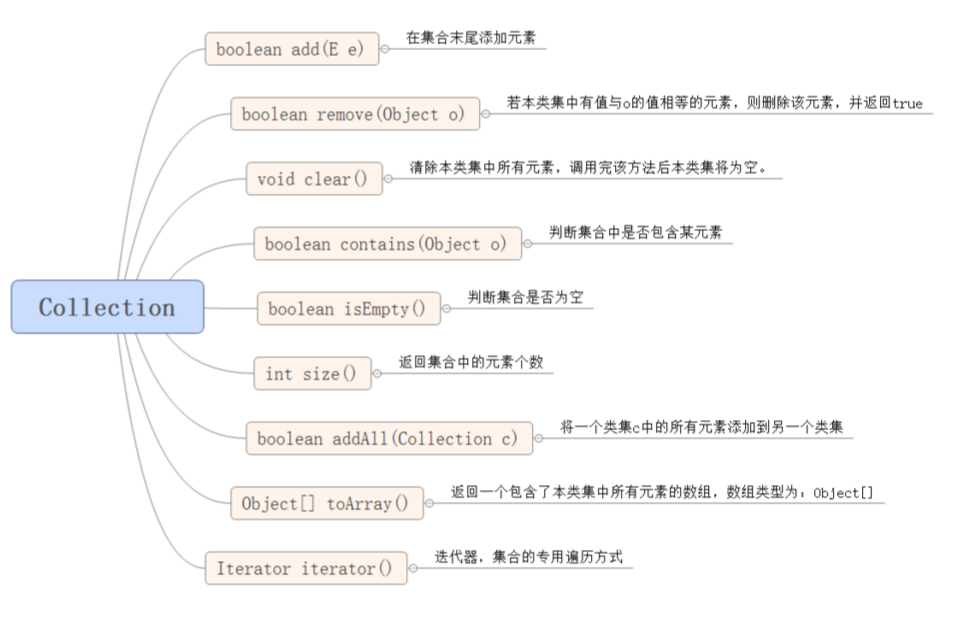
## 数据结构



集合和数组的区别



#### **Collection集合的方法**



#### 常用集合的分类

**Collection** 接口的接口 对象的集合（单列集合）  
├——-**List** 接口：元素按进入先后有序保存，可重复  
│—————-├ **LinkedList**接口实现类， 链表， 插入删除， 没有同步， 线程不安全  
│—————-├ **ArrayList** 接口实现类， 数组， 随机访问， 没有同步， 线程不安全  
│—————-└ **Vector** 接口实现类 数组， 同步， 线程安全  
│ ———————-└ **Stack** 是Vector类的实现类  
└——-**Set** 接口： 仅接收一次，不可重复，并做内部排序  
├—————-└**HashSet** 使用hash表（数组）存储元素  
│————————└**LinkedHashSet** 链表维护元素的插入次序  
└ —————-**TreeSet** 底层实现为二叉树，元素排好序

**Map** 接口 键值对的集合 （双列集合）  
├———**Hashtable** 接口实现类， 同步， 线程安全  
├———**HashMap** 接口实现类 ，没有同步， 线程不安全-  
│—————–├ **LinkedHashMap** 双向链表和哈希表实现  
│—————–└ **WeakHashMap**  
├ ——–**TreeMap** 红黑树对所有的key进行排序  
└———**IdentifyHashMap**

**数组**：采用一段连续的存储单元来存储数据。对于指定下标的查找，时间复杂度为O(1)；通过给定值进行查找，需要遍历数组，逐一比对给定关键字和数组元素，时间复杂度为O(n)，当然，对于有序数组，则可采用二分查找，插值查找，斐波那契查找等方式，可将查找复杂度提高为O(logn)；对于一般的插入删除操作，涉及到数组元素的移动，其平均复杂度也为O(n)

**线性链表**：对于链表的新增，删除等操作（在找到指定操作位置后），仅需处理结点间的引用即可，时间复杂度为O(1)，而查找操作需要遍历链表逐一进行比对，复杂度为O(n)

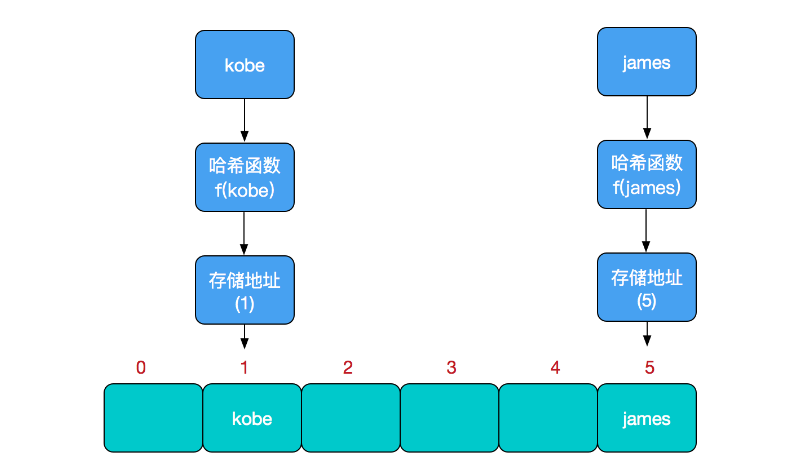
**二叉树**：对一棵相对平衡的有序二叉树，对其进行插入，查找，删除等操作，平均复杂度均为O(logn)。

**哈希表**：相比上述几种数据结构，在哈希表中进行添加，删除，查找等操作，性能十分之高，不考虑哈希冲突的情况下，仅需一次定位即可完成，时间复杂度为O(1)，接下来我们就来看看哈希表是如何实现达到惊艳的常数阶O(1)的。

　　我们知道，数据结构的物理存储结构只有两种：**顺序存储结构**和**链式存储结构**（像栈，队列，树，图等是从逻辑结构去抽象的，映射到内存中，也这两种物理组织形式），而在上面我们提到过，在数组中根据下标查找某个元素，一次定位就可以达到，哈希表利用了这种特性，**哈希表的主干就是数组**。

　　比如我们要新增或查找某个元素，我们通过把当前元素的关键字 通过某个函数映射到数组中的某个位置，通过数组下标一次定位就可完成操作。

**存储位置 = f(关键字)**

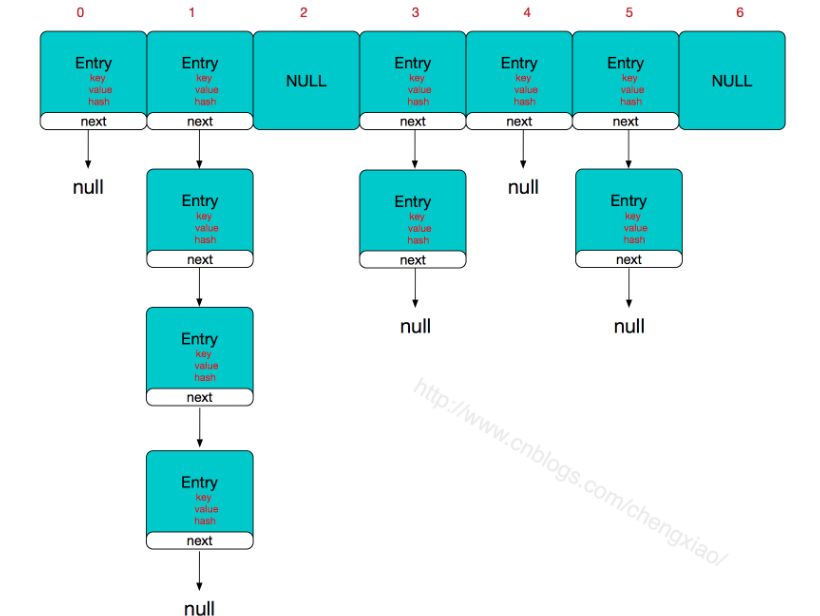
　其中，这个函数f一般称为**哈希函数**，这个函数的设计好坏会直接影响到哈希表的优劣。举个例子，比如我们要在哈希表中执行插入操作：

哈希冲突：对某个元素进行哈希运算，得到一个存储地址，然后要进行插入的时候，发现已经被其他元素占用了，其实这就是所谓的哈希冲突，也叫哈希碰撞。

解决方案：开放定址法（发生冲突，继续寻找下一块未被占用的存储地址），再散列函数法，链地址法，而HashMap即是采用了链地址法，也就是数组+链表的方式。

HashMap的主干是一个Entry数组。Entry是HashMap的基本组成单元，每一个Entry包含一个key-value键值对。

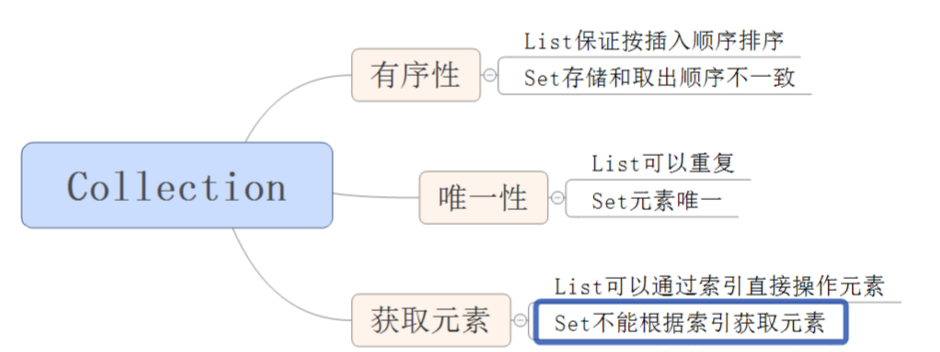
 HashMap的整体结构如下



HashMap由数组+链表组成的，数组是HashMap的主体，链表则是主要为了解决哈希冲突而存在的，如果定位到的数组位置不含链表（当前entry的next指向null）,那么对于查找，添加等操作很快，仅需一次寻址即可；如果定位到的数组包含链表，对于添加操作，其时间复杂度为O(n)，首先遍历链表，存在即覆盖，否则新增；对于查找操作来讲，仍需遍历链表，然后通过key对象的equals方法逐一比对查找。所以，性能考虑，HashMap中的链表出现越少，性能才会越好。

## **List和Set集合详解**

##### **list和set的区别**



#### List

（1）ArrayList：底层数据结构是数组，查询快，增删慢，线程不安全，效率高，可以存储重复元素  
（2）LinkedList 底层数据结构是链表，查询慢，增删快，线程不安全，效率高，可以存储重复元素  
（3）Vector:底层数据结构是数组，查询快，增删慢，线程安全，效率低，可以存储重复元素

#### 垃圾回收机制

1. 判断是否存活的两种方法：引用计数法 、 可达性分析
2. 引用计数法：

为每一个创建的对象分配一个引用计数器，用来存储该对象被引用的个数。当该个数为零，意味着没有人再使用这个对象，可以认为“对象死亡”。但是，这种方案存在严重的问题，就是无法检测“循环引用”：当两个对象互相引用，即时它俩都不被外界任何东西引用，它俩的计数都不为零，因此永远不会被回收。而实际上对于开发者而言，这两个对象已经完全没有用处了。

因此，Java 里没有采用这样的方案来判定对象的“存活性”。

3.可达性分析：

这种方案是目前主流语言里采用的对象存活性判断方案。基本思路是把所有引用的对象想象成一棵树，从树的根结点 GC Roots 出发，持续遍历找出所有连接的树枝对象，这些对象则被称为“可达”对象，或称“存活”对象。其余的对象则被视为“死亡”的“不可达”对象，或称“垃圾”。

锁的机制

