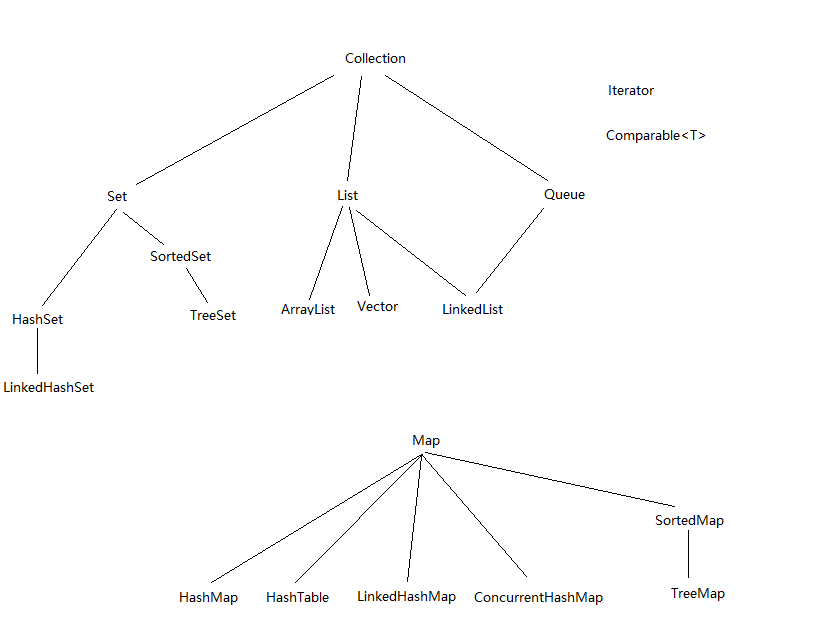
# Java相关

## 容器（HashMap、HashSet、LinkedList、ArrayList、数组等）



Java中集合的重要类在图中

1. 总体介绍

ArrayList是动态数组，初始容量是10，擅长随机访问，线程不安全。

LinkedList是双向链表，也可以当做单链表和堆栈用，不仅实现了List接口，还实现了Queue接口。不支持随机访问，查找某个元素，都是从头或者从尾进行。线程不安全。

Vector 差不多是线程安全的ArrayList。

Stack 继承字Vector，是线程安全的堆栈。

HashSet

TreeSet

HashMap 数组+单链表(红黑树)的实现方式。1.7以前的节点是Entry，1.8以后的源码是node。

HashTable

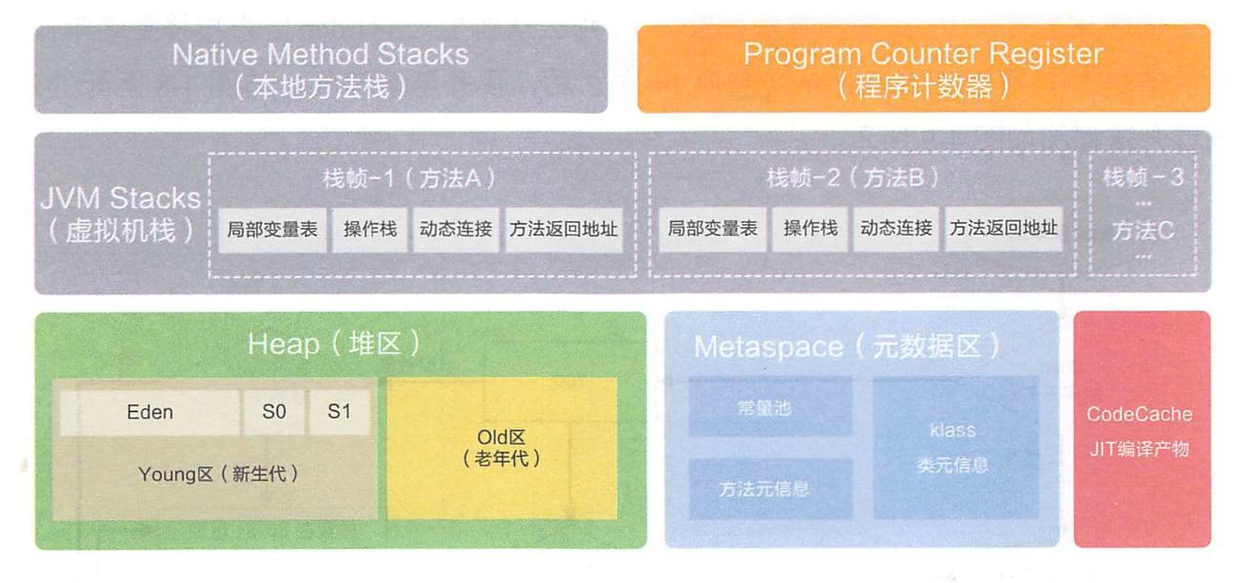
ConcurrentHashMap

1. 是否有序，是否可重复性

List和Queue都是有序的可重复性的，Set和Map都是无序的

1. 是否线程安全
2. HashMap，HashTable和ConcurrentHashMap的区别

## 内存模型



图中元数据区又叫方法区，是被所有线程共享的区域。

1. 本地方法栈：Native方法相关，不大需要关心
2. 虚拟机栈：线程私有。本地变量不会更改，随线程的创建而创建，每个方法都会创建一个栈帧。
3. 程序计数器：线程私有。如果CPU执行对的是JVM方法，那该寄存器中保存的就是当前执行的指令的地址，如果是Native方法， 则为空。
4. 堆内存：所有线程共享，虚拟机启动时已创建，所有的对象和数组都在堆上分配。
5. 方法区：图上的元数据区。也是所有线程共享，主要存储类的信息，常量池等。在逻辑上是属于堆的一部分，为了区分，又叫非堆。

《深入理解Java虚拟机》

内存模型详解：<https://blog.csdn.net/kdy527/article/details/86511693>

JVM和JMM详解：<https://www.cnblogs.com/czwbig/p/11127124.html>

## 垃圾回收算法（JVM）

自动垃圾回收

垃圾回收的区域是内存中的**堆**区域。

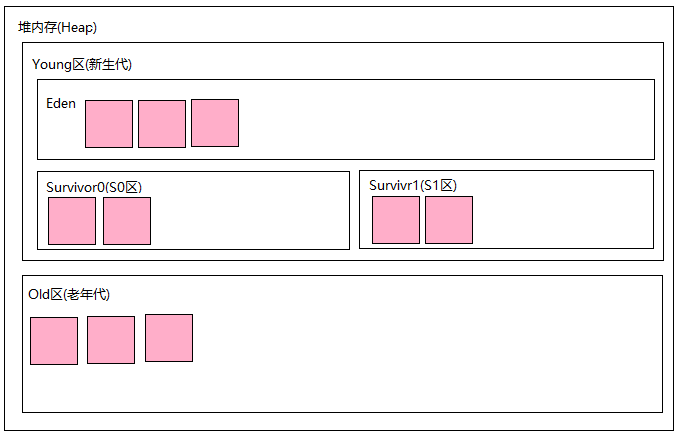
步骤为三步。

1. 标记：标记出内存区域中哪些对象正在被引用，哪些对象没有被引用。
2. 清除：将标记出来的未引用对象清除掉，留出空间。
3. 压缩：清除之后，已使用的内存空间不连续，要将其放到一起，这样之后分配给新对象就更简单快捷。



这里有一些缺陷，最大的一个就是标记，如果每次标记都遍历堆内存空间，会有很大开销。所以，在堆中又分了几个部分，即分代来分别GC，分别是新生代，老年代，永生代。

但是永生代在1.7之后就被移除了，1.8被元空间取代。区别是元空间不在虚拟机中，而是本地内存。关于永生代这里就不讨论更多了，回归垃圾回收。



在新生代中的GC，又叫Minor GC，老年代中的GC，又叫Major GC，而整个堆空间的GC叫Full GC。

Minor GC：

默认情况，大多数新创建的对象会在Eden区(大对象特殊处理)。Minor GC之前，对象只会存在于Eden区和，某个S区(From 区)，另一个S区(To 区)是空的，

经过Minor GC后，Eden区和From区未到达年龄阈值的对象会被移到(复制)另一个空闲的To区中。且在S区中的对象，每经过一次Minor GC，年龄会加1，当S区中的对象年龄到一定程度(可配置)，在GC时，会被移动到老年区。然后Eden区和From区会被清空，两个S区，From和To的角色交换，等待下一次GC。

年轻代中的垃圾回收算法是复制算法，这样不会产生内存碎片。但是赋值对象会花费一些时间。

Major GC：

采用标记清除(标记压缩)算法。

过程：遍历老年代对象，发现对象有引用，就标记。再遍历一次，发现未标记的对象，清除。

优缺点：进入老年代的对象，一般比较稳定，被清除的概率不高，如果使用复制算法，被复制的对象将会有很多，所以这种算法对资源的开销很低。但是回收后会出现内存碎片。

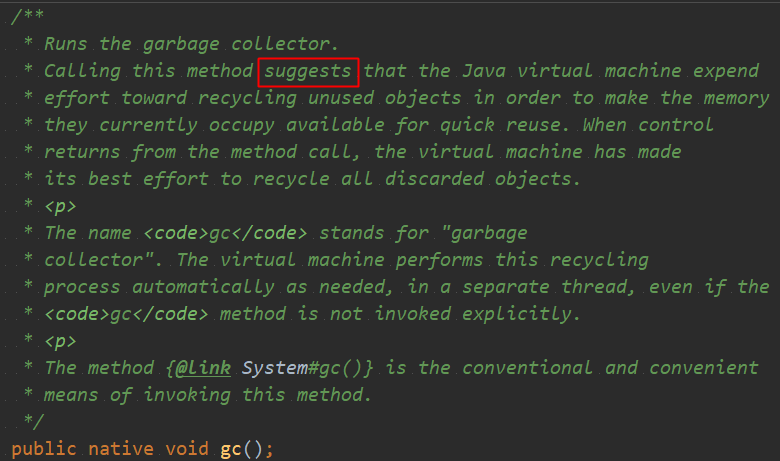
如果清除后再对内存进行压缩（上文的第三步），这样就能解决内存碎片化的问题。

图解新生代GC机制：<https://blog.csdn.net/hollis_chuang/article/details/91349868>

老年代新生代的GC原理：<https://blog.csdn.net/weixin_41843918/article/details/80855657>

## 垃圾回收机制和调用 System.gc()的区别？

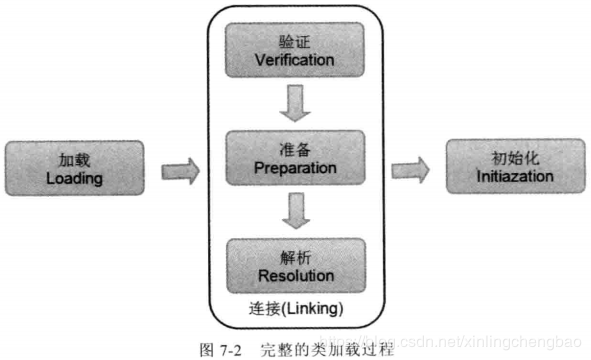
调用System.gc()，只是告诉JVM希望进行一次垃圾回收。Native的代码看不到。



## 类加载过程（需要多看看，重在理解，对于热修复和插件化比较重要）

类加载过程即是指JVM虚拟机把.class文件中类信息加载进内存，并进行解析生成对应的class对象的过程。

举个通俗点的例子来说，JVM在执行某段代码时，遇到了class A， 然而此时内存中并没有class A的相关信息，于是JVM就会到相应的class文件中去寻找class A的类信息，并加载进内存中，这就是我们所说的类加载过程。



<https://www.cnblogs.com/luohanguo/p/9469851.html>

## 反射

## 多线程和线程池

## 设计模式（六大基本原则、项目中常用的设计模式、手写单例等）

设计模式-刘伟：<https://blog.csdn.net/lovelion/article/details/17517213>

**模式是在特定环境下人们解决某类重复出现问题的一套成功或有效的解决方案。**

1. 六大原则
   1. 单一职责原则：一个类只负责一个功能领域中相应的职责，降低耦合度。
   2. 里氏代换原则：所有引用基类（父类）的地方，都必须能透明地使用其子类的对象。子类可以扩展父类的功能，但是不能更改。就是多态。
   3. 依赖倒转原则：也就是面向接口编程，而不是针对细节变成。抽象不应该依赖于细节，细节应当依赖于抽象。对象和传递的参数应该尽量都是接口。
   4. 迪米特法则：一个软件实体应当尽可能少地与其他实体发生作用，降低耦合度。
   5. 接口隔离原则：让借口不要那么复杂。复杂的业务尽量分解成多个单独的接口。也可以说是接口的单一职责原则。
   6. 开闭原则：一个软件应该对扩展开放，对修改关闭，也就是说尽量要在不更改源代码的基础之上来进行更新扩展。
2. Android中常用的设计模式
   1. 单例。

各种框架，自己写的各种工具类。

* 1. 建造者模式。

Dialog，自定义符合设计师要求的Dialog。

如果我们需要创建比较复杂的对象，且需要很好的灵活性和扩展性时，可以使用建造者模式。

这些对象所创建的产品几班有很多的共同点，组成部分相似。

还可以分步骤来创建，将创建步骤分在不同的方法中，能够有更清晰的逻辑。

缺点是建造者本身会比较复杂。

* 1. 策略模式。

一类事情可以有很多种解决办法，可以在不同的情况下选择不同的策略。

属性动画中的插值器就是这种体现。

* 1. 职责链模式。

View的测量定位，绘制，事件传递，事件分发。

在软件开发中，如果遇到有多个对象可以处理同一请求时可以应用职责链模式。

* 1. 观察者模式

广播BroadCast，RxJava，LiveData，EventBus等等。事件响应机制。

对象之间的一对多的依赖关系，当被观察着发生改变时，所有的观察者都等得到通知并被自动更新。

* 1. 适配器模式

ListView，RecyclerView，ViewPager

兼容不协调的对象。最常用的Adapter都是UI和数据之间的适配兼容。

将目标类和适配者类解耦，可以方便的更换适配器，方便扩展和灵活性。

## Java 四大引用

后来，Java 对引用的概念进行了扩充，将引用分为强引用（Strong Reference）、软引用（Soft Reference）、弱引用（Weak Reference）、虚引用（Phantom Reference）4 种，这 4 种引用强度依次逐渐减弱。

1. 强引用就是指在程序代码之中普遍存在的，类似"Object obj=new Object()"这类的引用，垃圾收集器永远不会回收存活的强引用对象。
2. 软引用：还有用但并非必需的对象。在系统 将要发生内存溢出异常之前 ，将会把这些对象列进回收范围之中进行第二次回收。
3. 弱引用也是用来描述非必需对象的，被弱引用关联的对象 只能生存到下一次垃圾收集发生之前 。当垃圾收集器工作时，无论内存是否足够，都会回收掉只被弱引用关联的对象。
4. 虚引用是最弱的一种引用关系。 无法通过虚引用来取得一个对象实例 。为一个对象设置虚引用关联的唯一目的就是能在这个对象被收集器回收时收到一个系统通知。

## Java 的泛型

Java中的泛型只在编译期有效，运行时是没有泛型的概念的。

例子：Integer和String泛型的List的class是同一个java.util.ArrayList，这个原始类型

<https://www.cnblogs.com/tong-yuan/p/generic.html>

## final、finally、finalize 的区别

## 接口、抽象类的区别

# Android 相关

## 自定义 View

## 事件拦截分发

## 解决过的一些性能问题，在项目中的实际运用

## 性能优化工具

## 性能优化 （讲讲你自己项目中做过的性能优化）

## Http[s]请求慢的解决办法（DNS、携带数据、直接访问 IP）

## 缓存自己如何实现（LRUCache 原理）

## 图形图像相关：OpenGL ES 管线流程、EGL 的认识、Shader 相关

## SurfaceView、TextureView、GLSurfaceView 区别及使用场景

## 动画、差值器、估值器（Android中的View动画和属性动画 、Android 动画 介绍与使用）

## MVC、MVP、MVVM

## Handler、ThreadLocal、AsyncTask、IntentService 原理及应用

## Gradle（Groovy 语法、Gradle 插件开发基础）

## 热修复、插件化

## 组件化架构思路

## 系统打包流程

## Android 有哪些存储数据的方式。

## SharedPrefrence 源码和问题点；

## sqlite 相关

## 如何判断一个 APP 在前台还是后台？

## AMS 、PMS

## Activity 启动流程，App 启动流程

## Binder 机制（IPC、AIDL 的使用）

## 为什么使用 Parcelable，好处是什么？

## Android 图像显示相关流程，Vsync 信号等

# 算法与数据结构

## 时间复杂度 / 空间复杂度

## 常用的排序算法有哪些？

## 字符串反转

## 链表反转（头插法）

## 如何查找第一个只出现一次的字符（Hash查找）

## 如何查找两个子视图的共同父视图？

## 无序数组中的中位数(快排思想)

## 如何给定一个整数数组和一个目标值，找出数组中和为目标值的两个数。

## 二叉树前序、中序、后序遍历

## 最大 K 问题

## 广度、深度优先搜索算法

## String 转 int。核心算法就三行代码，不过临界条件很多，除了判空，还需要注意负数、Integer 的最大最小值边界等；

## 如何判断一个单链表有环？

## 100 亿个单词，找出出现频率最高的单词。要求几种方案；

## 链表每 k 位逆序；

## 镜像二叉树；

## 找出一个无序数组中出现超过一半次数的数字；

## 计算二叉树的最大深度，要求非递归算法。

## String 方式计算加法。

# 网络

## 网络框架对比和源码分析

## 网络七层协议有哪些？

## Http 和 Https 的区别？Https为什么更加安全？

## HTTPS的连接建立流程

## 解释一下 三次握手 和 四次挥手

## TCP 和 UDP的区别

## Cookie和Session

## DNS是什么？

## DNS解析过程

## HTTP报文结构

## HTTP与HTTPS的区别以及如何实现安全性

## 如何验证证书的合法性?

## https中哪里用了对称加密，哪里用了非对称加密，对加密算法（如RSA）等是否有了解?

## client如何确定自己发送的消息被server收到?

## 谈谈你对WebSocket的理解

## WebSocket与socket的区别

## 谈谈你对安卓签名的理解。

## 请解释安卓为啥要加签名机制?

## 视频加密传输

## App 是如何沙箱化，为什么要这么做？

## 权限管理系统（底层的权限是如何进行 grant 的）？

# 源码理解

## Glide ：加载、缓存、LRU 算法 (如何自己设计一个大图加载框架) （LRUCache 原理）

## EventBus

## LeakCanary

## ARouter

## 插件化（不同插件化机制原理与流派，优缺点。局限性）

## 热修复

## RXJava （RxJava 的线程切换原理）

## Retrofit （Retrofit 在 OkHttp 上做了哪些封装？动态代理和静态代理的区别，是怎么实现的）

## OkHttp

# Kotlin 相关

## 从原理分析Kotlin的延迟初始化: lateinit var和by lazy

## 使用Kotlin Reified 让泛型更简单安全

## Kotlin里的Extension Functions实现原理分析

## Kotlin系列之顶层函数和属性

## Kotlin 兼容 Java 遇到的最大的 “坑”

## Kotlin 的协程用力瞥一眼

## Kotlin 协程「挂起」的本质

## 到底什么是「非阻塞式」挂起？协程真的更轻量级吗？

## 资源混淆是如何影响到Kotlin协程的

## Kotlin Coroutines(协程) 完全解析

## 破解 Kotlin 协程

# Flutter相关

## Dart 当中的 「..」表示什么意思？

## Dart 的作用域

## Dart 是不是单线程模型？是如何运行的？

## Dart 是如何实现多任务并行的？

## 说一下Dart异步编程中的 Future关键字？

## 说一下Dart异步编程中的 Stream数据流？

## Stream 有哪两种订阅模式？分别是怎么调用的？

## await for 如何使用？

## 说一下 mixin机制？

## 请简单介绍下Flutter框架，以及它的优缺点？

## 介绍下Flutter的理念架构

## 介绍下FFlutter的FrameWork层和Engine层，以及它们的作用

## 介绍下Widget、State、Context 概念 - Widget

## 简述Widget的StatelessWidget和StatefulWidget两种状态组件类

## StatefulWidget 的生命周期

## 简述Widgets、RenderObjects 和 Elements的关系

## 什么是状态管理，你了解哪些状态管理框架？

## 简述Flutter的绘制流程

## 简述Flutter的线程管理模型

## Flutter 是如何与原生Android、iOS进行通信的？

## 简述Flutter 的热重载