### 解释一下 OOP（Object-oriented programming） 的概念

1、什么是对象

* 万物皆对象，对象是程序的基本单元
* 类的定义包含数据的形式（成员变量）以及对数据的操作（成员方法），对象是类的实例
* 每个对象都有自己的空间，可以容纳其他对象（成员变量）

2、面向对象三大特性

* 面向对象编程三大特性：封装、继承、多态
* 类可以封装自己的属性和方法（设置为私有）可参考[链接](http://www.cnblogs.com/chenssy/p/3351835.html)
* 封装隐藏了类的内部实现机制，可以在不影响使用的情况下改变类的内部结构，同时也保护了数据
* 继承可以重用父类代码，达到代码复用和可扩展的效果
* 多态就是对象的一个方法在执行时可以有多种状态，即父类引用可以持有子类对象。
* 非静态成员方法的调用：编译时看父类，运行时看子类。即父类类型的引用可以调用父类中定义的所有属性和方法，对于只存在于子类中的方法和属性无法调用
* 多态优点：不用创建一堆子类对象的引用；多态缺点：不能使用子类特有的成员属性和子类特有的成员方法

### 抽象类和接口的区别？[Link](https://arjun-sna.github.io/java/2017/02/02/abstractvsinterface/)

区别：

* 关键字不同：接口使用 implements 来实现；抽象类使用 extends 来继承
* 抽象类可以包含具体方法和抽象方法；接口只可以包含抽象方法（抽象方法指没有实现的方法）
* 抽象类的方法可以使用 public、protected、private 等修饰符；接口的方法默认使用 public abstract 修饰，成员变量默认使用 public static final 修饰
* 一个接口可以继承（extends）多个父接口；一个抽象类只能继承（extends）一个父类
* 接口可以理解为抽象方法的集合；抽象类可以理解为一个没有包含足够的信息去描述具体对象的类，终归还是类

相同点：

* 接口和抽象类都不可以实例化

### 序列化是什么?如何实现它?[参考链接1](https://my.oschina.net/andot/blog/784235)、[参考链接2](https://tech.meituan.com/serialization_vs_deserialization.html)、[参考链接3](https://docs.microsoft.com/zh-cn/dotnet/standard/serialization/)

* 序列化指把应用层的对象或数据结构转换成一段连续的二进制串
* 反序列化指把二进制串转换成应用层的对象或数据结构
* 序列化按可读性可分为两类：二进制序列化、文本序列化（如 XML、JSON）以及半文本序列化
	+ 文本序列化：可读性、可编辑性好
	+ 二进制序列化：不具有可读性、但解析速度更有优势
* 按跨语言能力可分为：特定语言专有序列化和跨语言序列化
	+ 大部分语言内置的序列化都属于特定语言专有序列化，如 Java 的序列化
	+ 文本序列化通常为跨语言序列化，如 JSON、XML等
* Java 序列化是一种将对象转换为字节流的过程，目的是为了将该对象存储到内存中，等后面再次构建该对象时可以获取到该对象先前的状态及数据信息。
* Java 中，有两种方式可以实现序列化，既可以实现 Serializable 接口，也可以实现 Parcelable 接口。
* 然而，在 Android 中，我们不应该使用 Serializable 接口。因为 Serializable 接口使用了反射机制，这个过程相对缓慢，而且往往会产生出很多临时对象，这样可能会触发垃圾回收器频繁地进行垃圾回收。相比而言，Parcelable 接口比 Serializable 接口效率更高，性能方面要高出 10x 多倍。

### 什么是单例？[几种写法](http://wuchong.me/blog/2014/08/28/how-to-correctly-write-singleton-pattern/)

* 单例模式是一种对象创建型模式，指的是一个类只能被初始化一次，即只有一个实例。
* 这个类只能有一个实例
* 它必须自行创建这个实例
* 它必须自行向整个系统提供这个实例

单例模式几种写法对比：

1、懒汉式：非线程安全

2、懒汉式（加锁）：线程安全了，但效率低，任何时候只能有一个线程调用方法

3、恶汉式：非懒加载，单例对象依赖参数或配置文件时就无法使用了

4、双重校验锁：推荐

5、匿名内部类：推荐！静态内部类是私有的，除 getInstance 没有其他方法可访问它，所以是懒加载；读取实例时不会进行同步，没有性能缺陷

6、枚举：Android 中不推荐

### 什么是匿名内部类？[参考链接](http://wiki.xuchongyang.com/Java/%E5%9F%BA%E7%A1%80%E7%9F%A5%E8%AF%86%E7%82%B9.html)

* 匿名内部类即没有名字的内部类，只可被使用一次
* 使用匿名内部类前提条件：必须继承自一个父类或实现一个接口
* 语法格式为 new SuperType(construction parameters){}
* 语法格式为 new 父类构造器(参数列表)|实现接口(){}
* 直接将抽象类的方法或者接口方法在大括号中实现，可以省略一个类的书写
* 匿名内部类中不能定义构造方法，但可以使用初始化语块代替构造方法

### 对字符串进行 == 和 equals() 操作时有什么区别？

* == 用来判断内存地址是否相同
* equals()用来判断字符串的内容是否相同

### hashCode() 和 equals() 何时使用？

* 相等（相同）的对象必须具有相等的哈希码（或者散列码）
* 如果两个对象的 hashCode 相同，它们并不一定相同
* 往 Set 集合中添加元素时，要保证元素不重复，先调用该元素的 hasCode 方法，定位到它应该放置的物理位置。 如果这个位置上没有元素，就可以直接存储在这个位置上，不用再进行任何比较；如果这个位置上已经有元素了，就调用它的 equals 方法与新元素进行比较，相同的话就不存了，不相同就散列其它的地址。
* 集合查找时，hashcode 能大大降低对象比较次数，提高查找效率

### Java 中的 final, finally 和 finalize?

* final：修饰基本数据类型时，代表常量（不可修改）；修饰对象引用时，引用不可重新赋值，而对象依然可以修改；修饰方法时，代表该方法不可被重写；修饰类时，代表该类不可被继承
* finally：异常处理时 try／catch／finally 语句中的 finally 语句块无论有没有异常都会执行 finally 语句块，当然也有特殊情况不执行 finally
* finalize：垃圾回收器要回收对象的时候，首先会调用这个类的 finalize 方法，可以在该方法中释放调用本地方法时分配的内存

### 什么是内存泄露，Java 是如何处理它的？

* 存在下面的这种对象，这些对象不会被 GC 回收，却占用着内存，即为内存泄漏（简单说：存在已申请的无用内存无法被回收）
	+ 该对象是可达的，即还在被引用着
	+ 该对象是无用的，即程序以后不会再使用该对象
* 长生命周期的对象持有短生命周期的引用，就很可能会出现内存泄露
* 内存泄漏可能的情况：
	+ 静态集合类引起的
	+ 非静态内部类
	+ 单例模式
	+ HashSet 集合元素的属性被修改
	+ 各种连接问题

### 垃圾收集器是什么?它是如何工作的

* 垃圾收集是一种自动的内存管理机制
* 所有的对象实例都在 JVM 管理的堆区域分配内存
* 只要对象被引用，JVM 就会认为它还存活于进程中
* 一旦对象不再被引用，垃圾收集器将删除它并重新声明为未使用的内存

### 比较 Array 和 ArrayList[链接](http://blog.qianlicao.cn/translate/2016/03/09/array-vs-arraylist/)

不同点：

1、必答：

* 灵活性：ArrayList 优于 Array，ArrayList 是动态的；Array 是静态的，创建了数组就无法更改它的大小
* 元素类型：Array 既可以保存基本类型，也可以保存对象；ArrayList 不可以保存基本类型，可以保存封装类
* 实现上：Array 是本地的程序设计组件或者数据结构，ArrayList 是一个 Java 集合类的类
* 性能上：Array 会优于 ArrayList

2、其他

* 类型安全方面：ArrayList 是类型安全的，支持编译时检查；Array 不是类型安全的，支持运行时检查
* 泛型：ArrayList 可以显示的支持泛型，Array 不可以
* 维度：Array 可以是多维度的，ArrayList 并不支持指定维度

相同点：

* 数据结构：都是基于 index 的数据结构
* 空值：都可以存储空值（null），但只有 Object 的数组可以这样，基本类型会存储他们的默认值
* 重复：都允许元素重复

### 比较 HashSet 和 TreeSet

* TreeSet 会自动按自然排序法给元素排序，HashSet 不会
* 如果不需要使用排序功能,应该使用 HashSet,因为其性能优于 TreeSet
* 如果 TreeSet 传入的是自定义的对象,必须让该对象实现 Comparable 接口

### Java 中的类型转换

Java 中的类型转换分为基本数据类型和引用数据类型

* 基本数据类型
* 基本数据类型的转换分为自动转换和强制转换
	+ 自动转换是从位数低的类型向位数高的类型转换
	+ 强制类型转是从位数高的类型向位数低的类型转换，需要转型的数据前加上“()”，然后在括号内加入需要转化的数据类型，转换会导致精度丢失
* 引用数据类型
	+ 子类可以转换成父类
	+ 父类转换为子类不一定可行：当父类（引用）引用的为子类对象时，此时父类可以正常转换为子类；而当父类（引用）引用的确实为父类对象时，此时无法转换为子类，会抛出 ClassCastException 异常

### 静态绑定和动态绑定的区别

1、什么是绑定？

绑定指的是一个方法的调用与方法所在的类(方法主体)关联起来。

2、定义

* 静态绑定：编译过程中编译器已经准确的知道这个方法是属于哪个类的了
* 动态绑定：需要在运行时根据对象具体的类型来绑定，来选择调用哪个类的方法

3、区分

* private 方法、static 方法、final 方法、构造器方法都是静态绑定的
* 其他方法全部为动态绑定

### 方法重载和重写的区别[链接](http://droidyue.com/blog/2014/12/28/static-biding-and-dynamic-binding-in-java/index.html)

* 重载是在同一个类中完成的，重写父类需要有子类。
* 方法重载时返回值类型、参数列表可以不同；方法重写时返回值类型、参数列表必须相同
* 重载的方法使用静态绑定完成，发生在编译时；重写的方法使用动态绑定完成，发生在运行时。性能：重载比重写更有效率
* private 方法、static 方法、final 方法都是可以重载，但不可以重写

### 什么是访问修饰符？它们能做什么？

* 访问修饰符是指能够控制类、成员变量、方法的使用权限的关键字
* 可以使用它们来保护对类、变量、方法和构造方法的访问
* public：共有的，对所有类可见
* protected：受保护的，对同一包内的类和所有子类可见
* private：私有的，在同一类内可见
* 默认：在同一包内可见。默认不使用任何修饰符。

### 接口可以继承另一个接口吗？

可以，Java 允许一个接口继承多个父接口。

### Java 中 static 关键字是什么意思？[链接](http://www.cnblogs.com/chenssy/p/3386721.html)

* static 表示“全局”或者“静态”的意思，用来修饰成员变量、成员方法或者内部类，也可以修饰代码块
* static 所蕴含“静态”的概念表示着它是不可恢复的，即在那个地方，你修改了，他是不会变回原样的，你清理了，他就不会回来了
* 被 static 修饰的成员变量和成员方法是独立于该类的，它不依赖于某个特定的实例变量，也就是说它被该类的所有实例共享
* 所有实例的引用都指向同一个地方，任何一个实例对其的修改都会导致其他实例的变化
* 静态变量是随着类加载时被完成初始化的；静态方法也是类加载时就存在了，所以不可为 abstract，必须实现；静态代码块也会随着类的加载一块执行

### Java 中静态方法可以被重写吗？

* 不可以被重写，静态方法隶属于某个类，只和类相关

### 什么是多态？什么是继承？[链接](http://www.cnblogs.com/chenssy/p/3354884.html)

多态：

* 多态就是父类引用可以持有子类对象。非静态成员方法的调用：编译时看父类，运行时看子类
* 多态优点：不用创建一堆子类对象的引用
* 多态缺点：不能使用子类特有的成员属性和子类特有的成员方法

继承：

* 继承是使用已存在的类的定义作为基础建立新类的技术，新类的定义可以增加新的数据或新的功能，也可以用父类的功能，但不能选择性地继承父类
* 通过使用继承我们能够非常方便地复用以前的代码，大大提高开发的效率

### Integer 和 int 之间的区别[自动装箱、自动拆箱](http://droidyue.com/blog/2015/04/07/autoboxing-and-autounboxing-in-java/index.html)

* int 是基本数据类型，Integer 是包装类
* Java中，会对 -128 到 127 的 Integer 对象进行缓存，当创建新的 Integer 对象时，如果符合这个这个范围，并且已有存在的相同值的对象，则返回这个对象，否则创建新的 Integer 对象

### Java 中的对象是否会以引用传递或者值传递？详细说明。[参考链接](http://xuchongyang.com/2017/10/11/Java-%E5%BC%95%E7%94%A8%E5%AD%A6%E4%B9%A0/)

Java 中的对象以按值传递的形式进行调用，所有方法得到的都是参数值的拷贝

* 对于基本数据类型的参数，方法得到的是其值的拷贝
* 对于引用数据类型的参数，方法得到的也是其值（所指向对象的内存地址）的拷贝
* 方法均不可以修改参数变量的值，对于引用类型参数来说，就是不能让引用类型参数指向新的对象
* 但是方法可以修改引用类型参数所指向对象的内容，因为方法得到的是该对象地址的拷贝，可以根据地址获取到该对象

### 什么是 ThreadPoolExecutor？ [Link](https://blog.mindorks.com/threadpoolexecutor-in-android-8e9d22330ee3)

* 线程池的目的是实现线程复用，减少频繁创建和销毁线程，提高系统性能
* ThreadPoolExecutor 是线程池的核心类，用于创建线程池
* ThreadPoolExecutor 的构造方法包括如下参数：
	+ corePoolSize 核心线程池大小
	+ maximumPoolSize 线程池最大容量
	+ keepAliveTime 线程池空闲时，线程存活时间
	+ TimeUnit 时间单位
	+ ThreadFactory 线程工厂
	+ BlockingQueue 任务队列
	+ RejectedExecutionHandler 线程拒绝策略

我们通常通过 Executors 类的如下几个静态方法来创建不同类型的线程池：

* newCachedThreadPool：可缓存线程池，无限大
* newFixedThreadPool：定长线程池
* newScheduledThreadPool：定长线程池，支持定时及周期性任务执行
* newSingleThreadExecutor：单个线程池

### 本地变量、实例变量以及类变量之间的区别？

* 类体由两部分组成，变量定义和方法定义
* 本地变量即局部变量，定义在方法内部或者方法的形参中
* 实例变量为为非静态变量，每一个对象的实例变量都不同
* 类变量为静态变量，属于类所有

### 什么是反射？ [Link](http://tutorials.jenkov.com/java-reflection/index.html)

### 在 Java 中什么是强引用、软引用、弱引用以及虚引用？[参考链接1](http://xuchongyang.com/2017/10/11/Java-%E5%BC%95%E7%94%A8%E5%AD%A6%E4%B9%A0/)[参考链接2](http://blog.csdn.net/mazhimazh/article/details/19752475)

* 强引用：是使用最普遍的引用。如果一个对象具有强引用，那垃圾回收器绝不会回收它。
* 软引用：如果一个对象只具有软引用，且内存空间足够，垃圾回收器就不会回收它；如果内存空间不足了，就会回收这些对象的内存。
* 弱引用：只具有弱引用的对象拥有比软引用更短暂的生命周期，在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现了只具有弱引用的对象，不管当前内存空间足够与否，都会回收它的内存。不过，由于垃圾回收器是一个优先级很低的线程，因此不一定会很快发现那些只具有弱引用的对象。
* 如果一个对象仅持有虚引用，那么它就和没有任何引用一样，在任何时候都可能被垃圾回收器回收。一般用于跟踪对象被垃圾回收器回收过程。

### 什么是依赖注入？能说几个依赖注入的库么？你使用过哪些？

### 关键字 synchronized 的作用是什么？[参考链接](http://www.cnblogs.com/GnagWang/archive/2011/02/27/1966606.html#!comments)

* 当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码。
* 当两个并发线程访问同一个对象 object 中的这个 synchronized(this) 同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。
* 然而，当一个线程访问 object 的一个 synchronized(this) 同步代码块时，另一个线程仍然可以访问该 object 中的非 synchronized(this) 同步代码块。
* 尤其关键的是，当一个线程访问 object 的一个 synchronized(this) 同步代码块时，其他线程对 object 中 *所有其它 synchronized(this) 同步代码块* 的访问将被阻塞。
* 上一个例子同样适用其它同步代码块。也就是说，当一个线程访问 object 的一个 synchronized(this) 同步代码块时，它就获得了这个 object 的对象锁。结果，其它线程对该 object 对象 *所有同步代码部分* 的访问都被暂时阻塞。

### 为什么说 String 不可变的？[参考链接](http://blog.csdn.net/zhangjg_blog/article/details/18319521)

* 什么是不可变对象
	+ 如果一个对象，在它创建完成之后，不能再改变它的状态，那么这个对象就是不可变的。
	+ 不能改变状态的意思是，不能改变对象内的成员变量，包括基本数据类型的值不能改变，引用类型的变量不能指向其他的对象，引用类型指向的对象的状态也不能改变。
* 在 Java 中 String 类其实就是对字符数组的封装
* String 类中有一个 char 数组的 value 引用变量，value 指向真正的数组对象
* JDK 1.6 及以前还有 offset 和 count 两个变量
* 这三个变量都是 private final 的，即一旦这三个值被初始化，就不能再改变了
* 但是可以通过反射改变 value 变量引用的对象，进而改变 String 对象

### 修饰符 transient 和 volatile 的作用是什么？

### finalize() 方法的作用是什么？

### 异常捕获中的 try{} finally{} 块儿是如何工作的?

### 类的实例化和类的初始化之间的区别是什么?[参考链接1](https://www.caoqq.net/java-class-initialization.html)、[参考链接2](http://blog.csdn.net/czhpxl007/article/details/50558319)、[参考链接3](http://blog.csdn.net/justloveyou_/article/details/72466416)

* 1.执行顺序不同：
	+ （类）初始化执行顺序：静态成员变量／静态代码块
	+ （对象）实例化执行顺序：成员变量／构造代码块 -> 构造方法
	+ 以上，静态成员变量／静态代码块的执行顺序为代码中的定义顺序，成员变量／构造代码块的执行顺序也为代码中的定义顺序
* 2.执行时机不同
* 类的初始化时机：
	+ 类的一个实例被创建时
	+ 类的一个静态方法被调用时
	+ 类声明的一个静态变量被赋值时
	+ 类声明的一个静态变量被使用且这个变量不是常量时
	+ 一个类初始化时，如果父类没初始化则需要先初始化父类
* 类的实例化时机
	+ 当创建一个类的新的实例时（可以通过 new 关键字、反射机制、clone 方法、反序列化方式等）

### 静态块何时运行?[链接](http://www.jianshu.com/p/8a3d0699a923)

* 静态块在 JVM 加载类时执行，也就是在类进行初始化时执行，仅执行一次
* 一个类中可以有多个静态代码块
* 类调用时，先执行静态代码块，然后才执行主函数
* 静态代码块其实就是给类初始化的，而构造代码块是给对象初始化的
* 静态代码块中的变量是局部变量，与普通函数中的局部变量性质没有区别

### 解释一下 Java 中的泛型?[参考链接](http://www.infoq.com/cn/articles/cf-java-generics)

* 泛型即宽泛的数据类型
* 允许在定义类和接口的时候使用类型参数（type parameter）
* 声明的类型参数在使用时用具体的类型来替换
* 使用泛型的时候加上的类型参数，会被编译器在编译的时候去掉，这个过程就称为类型擦除

### String、StringBuffer 和StringBuilder 的区别在哪里?

* String 是不可变对象，因此每次对 String 对象进行更改时，都会生成一个新的对象，然后将指针指向新的对象。
* 使用 StringBuffer 类时，每次都是对 StringBuffer 对象本身进行操作，并不是生成新的对象并改变引用。线程安全！
* StringBuilder 和 StringBuffer 非常类似，但是是非线程安全的，在单线程中性能比 StringBuffer 高。非线程安全！

### StringBuilder 是怎么避免不可变字符串分配的问题？

### 什么是自动装箱和拆箱？

* 自动装箱就是 Java 自动将原始类型值转换成对应的对象，比如将 int 的变量转换成 Integer 对象
* 反之将 Integer 对象自动转换成 int 类型值，这个过程叫做拆箱
* 自动装箱时编译器调用 valueOf 将原始类型值转换成对象
* 自动拆箱时，编译器通过调用类似 intValue(), doubleValue() 这类的方法将对象转换成原始类型值
* 自动装箱、拆箱主要发生在
	+ 赋值时
	+ 方法调用时

### 枚举和迭代器有什么区别？

### Java 中 *fail-fast* 和 *fail-safe* 的区别？

### 什么是 Java 优先级队列？

### 什么是注解？

### 多进程和多线程的区别[（参考！经常看！）](http://markxu.coding.me/wiki/%E5%A4%9A%E7%BA%BF%E7%A8%8B/Java%E5%9F%BA%E7%A1%80%EF%BC%9A%E5%A4%9A%E7%BA%BF%E7%A8%8B%E5%9F%BA%E7%A1%80%E7%9F%A5%E8%AF%86.html)

* 本质区别在于：每个进程拥有自己的一整套变量，而线程则共享数据。
* 多进程：

进程是程序的一次执行。计算机在同一刻运行多个程序，每个程序称为一个进程（计算机将 CPU 的时间片分配给每一个进程）

* 多线程：

线程是 CPU 的基本调度单位。一个程序同时执行多个任务，每个任务称为一个线程

补充：线程的状态切换

### 进程间通讯方法，线程间通讯方法

1、进程间通讯

* Broadcast
* Intent／Bundle
* File 共享
* Messenger
* AIDL
* ContentProvider
* Socket

2、线程间通讯

* Handler（AsyncTask、Message、runOnUiThread 等）
* EventBus
* LocalBroadcast

### sleep 和 wait 的区别

二者都可以暂停当前线程，释放 CPU 控制权，区别在于：

作用于谁和是否释放锁？

* wait 方法作用于 Object，sleep 方法作用于 Thread
* Object.wait 方法在释放 CPU 的同时，释放了对象锁的控制，使得其他线程可以使用同步控制块或方法；Thread.sleep 方法没有释放锁

### JNI 是什么

Java native interface，Java 本地接口

### 概述 Java 垃圾回收机制，如何更有效的管理内存，减少 OOM 的概率[参考](http://www.cnblogs.com/vamei/archive/2013/04/28/3048353.html)

1、内存结构：

* 内存分为栈（satck）和堆（heap）两部分
* JVM 中栈记录了方法调用，每个线程拥有一个栈（栈的每一帧中保存有该方法调用的参数、局部变量和返回地址）
* 栈中被调用方法运行结束时，相应的帧也会删除，参数和局部变量占用的空间也会释放
* 堆是 JVM 中可以自由分配给对象的区域，堆区由所有线程共享

2、垃圾回收

* JVM 自动清空堆中无用对象占用的空间就是垃圾回收
* 当一个对象没有引用指向它时，为不可达对象，此时会被回收

3、对象是否回收的依据

* 引用计数算法
* 可达性分析算法

4、回收基础

* Mark and sweep 机制：每个对象都有用于表示该对象是否可达的标记信息。垃圾回收启动时，Java 程序暂停运行，JVM 从根出发，找到所有可达对象并标记。然后扫描整个堆找到不可达对象，清空它们占用的内存
* Copy and sweep 机制：堆被分为两个区域，对象存活于两个堆中的一个。垃圾回收启动时，Java 程序暂停运行，JVM 从根出发找到所有可到达对象，把所有可到达对象复制到空白区域中并紧密排列（并修改由于对象地址变化引起的引用变化）。最后直接清空对象原先所存活的区域，使其成为新的空白区域。
* 对象比较长寿适用于 mark and sweep 机制；对象比较活跃则适用于 copy and sweep 机制，避免出现空隙
* 世代指对象经历过的垃圾回收的次数，堆分为三个世代：永久世代、成熟世代、年轻世代
* 永久世代的对象不会被垃圾回收
* 年轻世代进一步分为三个区域：eden 区、from 区、to 区
* 新生对象指从上次垃圾回收后创建的对象，存在于 eden 区。from 区和 to 区相当于 copy and sweep 中的两个区

5、分代回收

* 新建对象无法放入 eden 区时，会触发 minor collection，JVM 会采用 copy and sweep 机制将 eden 区和 from 区的可到达对象复制到 to 区，进行一次垃圾回收清空 eden 区和 from 区，to 区则存放着紧密排列的对象。接着 from 区成为了新的 to 区，原来的 to 区成为了新的 from 区
* 当 minor collection 时发现 to 区也放不下时，会将部分对象放入成熟世代
* 即使 to 区没有满，JVM 也会移动世代足够久远的对象到成熟世代
* 如果成熟世代放满对象无法放入新的对象，会触发 minor collection，JVM 采用 mark and sweep 机制对成熟世代进行垃圾回收