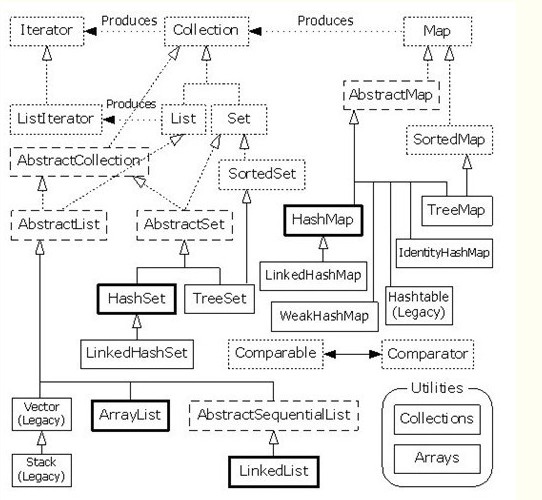
# Java Web学习笔记

## Java基础

### Java集合



#### ① Collection接口

Collection接口是 Set 、List 和 Queue 接口的父接口，提供了多数集合常用的方法声明，包括 add()、remove()、contains() 、size() 、iterator() 等。

1. List

List 关心的是索引，与其他集合相比，List特有的就是和索引相关的一些方法：get(int index) 、 add(int index,Object o) 、 indexOf(Object o) ；List是有序的Collection，使用此接口能够精确的控制每个元素插入的位置。用户能够使用索引（元素在List中的位置，类似于数组下标）来访问List中的元素，这类似于Java的数组；另外List允许有相同的元素。

**ArrayList** 可以将它理解成一个可增长的数组，默认初始大小为10. 成1.5倍加1增长。它提供快速迭代和快速随机访问的能力，其底层使用数组实现的，查询效率较高，但是增删效率低且是线程不安全的。

**LinkedList** 中的元素之间是双链接的，当需要快速插入和删除时LinkedList成为List中的不二选择，其底层使用双向循环链表实现，查询效率低，但增删元素效率比较高。

**Vector** 是ArrayList的线程安全版本，性能比ArrayList要低，现在已经很少使用。

1. Set

**Set关心唯一性，它不允许重复。**

HashSet**当不希望集合中有重复值，并且不关心元素之间的顺序时可以使用此类，使用哈希**[算法](http://lib.csdn.net/base/datastructure)**实现。**

LinkedHashset**当不希望集合中有重复值，并且希望按照元素的插入顺序进行迭代遍历时可采用此类。**

TreeSet**当不希望集合中有重复值，并且希望按照元素的自然顺序进行排序时可以采用此类。（自然顺序意思是某种和插入顺序无关，而是和元素本身的内容和特质有关的排序方式，譬如“abc”排在“abd”前面。）**

1. Queue

Queue用于保存将要执行的任务列表。

**LinkedList** 同样实现了Queue接口，可以实现先进先出的队列。

**PriorityQueue** 用来创建自然排序的优先级队列。

#### ② Map接口

Map关心的是唯一的标识符。他将唯一的键映射到某个元素。当然键和值都是对象。

**HashMap** 当需要键值对表示，又不关心顺序时可采用HashMap，线程不安全，允许key或value为NULL。

**Hashtable** 注意Hashtable中的t是小写的，它是HashMap的线程安全版本，不允许key或value为NULL。

**LinkedHashMap** 当需要键值对，并且关心插入顺序时可采用它。

**TreeMap** 当需要键值对，并关心元素的自然排序时可采用它。

### JAVA泛型

### JAVA IO、NIO

### JAVA线程

## JAVA线程池

Java通过Executors提供四种线程池，分别为：  
**newCachedThreadPool**创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。  
**newFixedThreadPool** 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

**newScheduledThreadPool** 创建一个定长线程池，支持定时及周期性任务执行。  
**newSingleThreadExecutor** 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

1. **newCachedThreadPool**

创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。示例代码如下：

1. ExecutorService cachedThreadPool = Executors.newCachedThreadPool();
2. **for** (**int** i = 0; i < 10; i++) {
3. **final** **int** index = i;
4. **try** {
5. Thread.sleep(index \* 1000);
6. } **catch** (InterruptedException e) {
7. e.printStackTrace();
8. }
10. cachedThreadPool.execute(**new** Runnable() {
12. @Override
13. **public** **void** run() {
14. System.out.println(index);
15. }
16. });
17. }

线程池为无限大，当执行第二个任务时第一个任务已经完成，会复用执行第一个任务的线程，而不用每次新建线程。  
**(2) newFixedThreadPool**  
创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。示例代码如下:

1. ExecutorService fixedThreadPool = Executors.newFixedThreadPool(3);
2. **for** (**int** i = 0; i < 10; i++) {
3. **final** **int** index = i;
4. fixedThreadPool.execute(**new** Runnable() {
6. @Override
7. **public** **void** run() {
8. **try** {
9. System.out.println(index);
10. Thread.sleep(2000);
11. } **catch** (InterruptedException e) {
12. // TODO Auto-generated catch block
13. e.printStackTrace();
14. }
15. }
16. });
17. }

因为线程池大小为3，每个任务输出index后sleep 2秒，所以每两秒打印3个数字。  
定长线程池的大小最好根据系统资源进行设置。如Runtime.getRuntime().availableProcessors()。可参考PreloadDataCache。

**(3) newScheduledThreadPool**

创建一个定长线程池，支持定时及周期性任务执行。延迟执行示例代码如下：

1. ScheduledExecutorService scheduledThreadPool = Executors.newScheduledThreadPool(5);
2. scheduledThreadPool.schedule(**new** Runnable() {
4. @Override
5. **public** **void** run() {
6. System.out.println("delay 3 seconds");
7. }
8. }, 3, TimeUnit.SECONDS);

表示延迟3秒执行。  
   
定期执行示例代码如下：

scheduledThreadPool.scheduleAtFixedRate(**new** Runnable() {

1. @Override
2. **public** **void** run() {
3. System.out.println("delay 1 seconds, and excute every 3 seconds");
4. }
5. }, 1, 3, TimeUnit.SECONDS);

表示延迟1秒后每3秒执行一次。  
ScheduledExecutorService比Timer更安全，功能更强大。

**(4) newSingleThreadExecutor**  
创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。示例代码如下：

1. ExecutorService singleThreadExecutor = Executors.newSingleThreadExecutor();
2. **for** (**int** i = 0; i < 10; i++) {
3. **final** **int** index = i;
4. singleThreadExecutor.execute(**new** Runnable() {
6. @Override
7. **public** **void** run() {
8. **try** {
9. System.out.println(index);
10. Thread.sleep(2000);
11. } **catch** (InterruptedException e) {
12. // TODO Auto-generated catch block
13. e.printStackTrace();
14. }
15. }
16. });
17. }

结果依次输出，相当于顺序执行各个任务。  
现行大多数GUI程序都是单线程的。Android中单线程可用于数据库操作，文件操作，应用批量安装，应用批量删除等不适合并发但可能IO阻塞性及影响UI线程响应的操作。

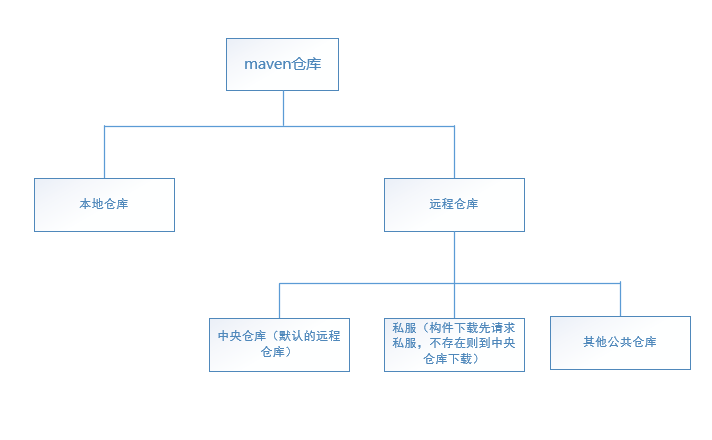
## 工具（Maven,Git）

① maven依赖范围Scope：compile、test、provided、runtime、system。

依赖原则：

1. 路径优先原则
2. 第一声明优先

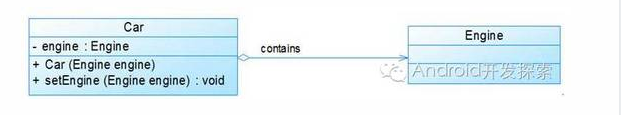
② maven仓库的结构



## UML基础

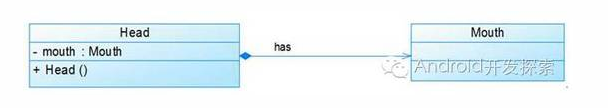
① 聚合关系和组合关系的区别：

聚合关系



上 图中的Car类与Engine类就是聚合关系（Car类中包含一个Engine类型的成员变量）。由上图我们可以看到，UML中 聚合关系用带空心菱形和 箭头的直线表示。聚合关系强调是“整体”包含“部分”，但是“部分”可以脱离“整体”而单独存在。比如上图中汽车包含了发动机，而发动机脱离了汽车也能够 单独存在。

组合关系



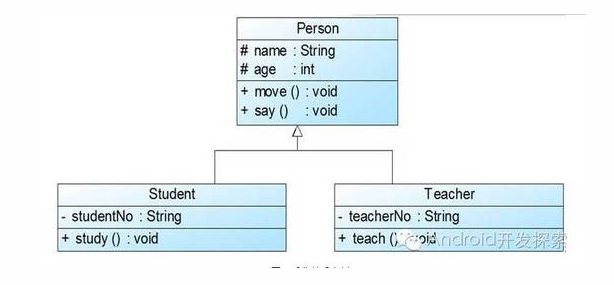
组合关系与聚合关系间的最大不同在于：这里的“部分”脱离了“整体”便不复存在。比如下图：

显然，嘴是头的一部分且不能脱离了头而单独存在。在UML类图中，组合关系用一个带实心菱形和箭头的直线表示。

② 继承关系和接口实现关系

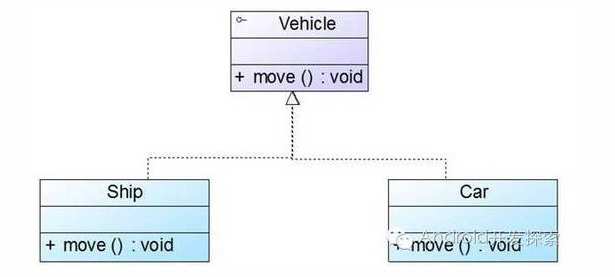
继承关系：

继承关系对应的是extend关键字，在UML类图中用带空心三角形的直线表示，如下图所示中，Student类与Teacher类继承了Person类。

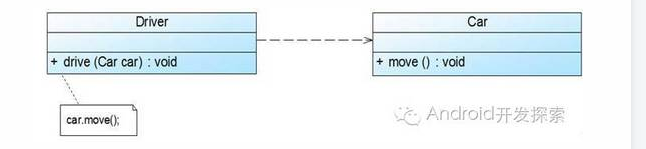


接口实现关系：

这种关系对应implements关键字，比如下图中，Car类与Ship类都实现了Vehicle接口。



③ 依赖关系



从上图中我们可以看到，Driver的drive方法只有传入了一个Car对象才能发挥作用，因此我们说Driver类依赖于Car类。在UML类图中，依赖关系用一条带有箭头的虚线表示。