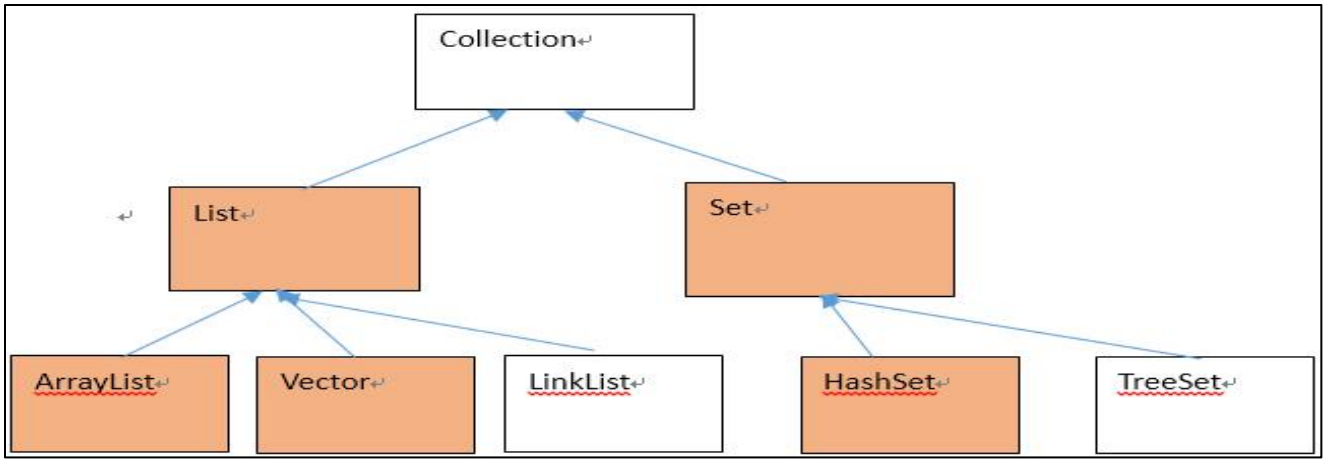
集合的体系结构

集合也叫容器用于存储对象。
我们根据不同的需求和不同的数据结构来对集合做了不同的抽象。



List:

删除：

E	**remove**(int index) 移除列表中指定位置的元素（可选操作）。
boolean	**remove**(Object o) 从此列表中移除第一次出现的指定元素（如果存在）（可选操作）。
boolean	**removeAll**(Collection<?> c) 从列表中移除指定 collection 中包含的其所有元素（可选操作）。
void	**clear**() 移除此 collection 中的所有元素（可选

ArrayList:

List 接口的大小可变数组的实现。实现了所有可选列表操作，并允许包括 null 在内的所有元素。除了实现 List 接口外，此类还提供一些方法来操作内部用来存储列表的数组的大小。

特点：ArrayList 中的元素和可以重复
是有序的集合，长度不固定。
不是线程安全的。
效率高。

LinkedList

List 接口的链接列表实现。实现所有可选的列表操作，并且允许所有元素（包括

null）。除了实现 List 接口外，LinkedList 类还为在列表的开头及结尾 get、remove 和 insert 元素提供了统一的命名方法。这些操作允许将链接列表用作堆栈、队列或双端队列。
不是线程安全的。

栈：

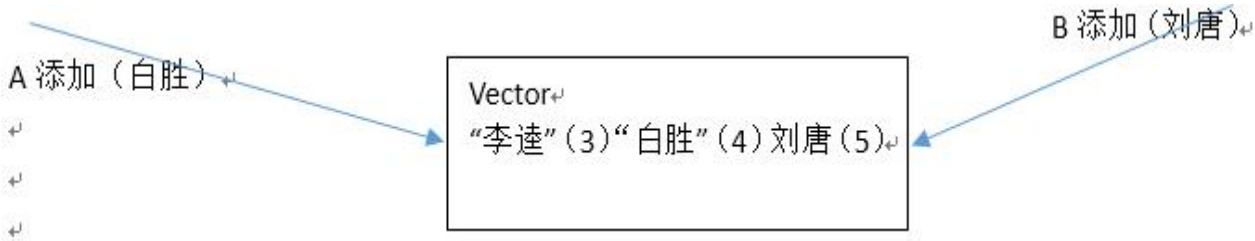
Push：压栈
Pop：出栈
栈的特点是先进后出，后进先出

队列：

队列的特点是：先进先出。

Vector

和 ArrayList 功能类似，最主要的区别就在于 vector 是线程并发安全的。但是缺点是效率比较低



Set 集合

一个不包含重复元素的 collection。更确切地讲，set 不包含满足 e1.equals(e2) 的元素对 e1 和 e2，并且最多包含一个 null 元素。
Set 里面的方法绝大多数都是继承于 Collection。
Set 的特点：

元素不可重复
元素无序的（跟添加的顺序无关）

HashSet

此类实现 Set 接口，由哈希表（实际上是一个 HashMap 实例）支持。它不保证 set 的迭代顺序；特别是它不保证该顺序恒久不变。此类允许使用 null 元素。

HashSet 的唯一性：

在 HashSet 做添加的时候会逐个来判断当前集合中的对象和要添加的对象的比较通过以下的条件判断两个对象是否相等

* e.hash == hash && ((k = e.key) == key || key.equals(k))

* hash 值必须相等并且两个对象的地址值相等或者 equals 返回 true

注意：当此方法被重写时，通常有必要重写 hashCode 方法，以维护 hashCode 方法的常规协定，该协定声明相等对象必须具有相等的哈希码。

HashSet

特点：

1.元素唯一性

2.无序行

3.允许 null 存在一个

4.不是线程安全（效率高）

LinkedHashSet

具有可预知迭代顺序的 Set 接口的哈希表和链接列表实现。此实现与 HashSet 的不同之外在于，后者维护着一个运行于所有条目的双重链接列表。此链接列表定义了迭代顺序，即按照将元素插入到 set 中的顺序（插入顺序）进行迭代。

特点：

1.元素唯一性

2.有序的

3.允许 null 存在一个

4.不是线程安全（效率高）

TreeSet

TreeSet 可以支持自定义排序，如果 TreeSet 所存储的对象的类没有实现 Comparable 接口就会报错 ClassCastException。所以我们如果想要使用 TreeSet 来对自定义的对象来排序必须实现 Comparable 接口。

特点：

1.元素唯一性

2.可自定义排序的

3.不允许 null 存在

4.不是线程安全

可变参数

可变参数的语法:

修饰符 返回值类型 方法名（数据类型…变量）{

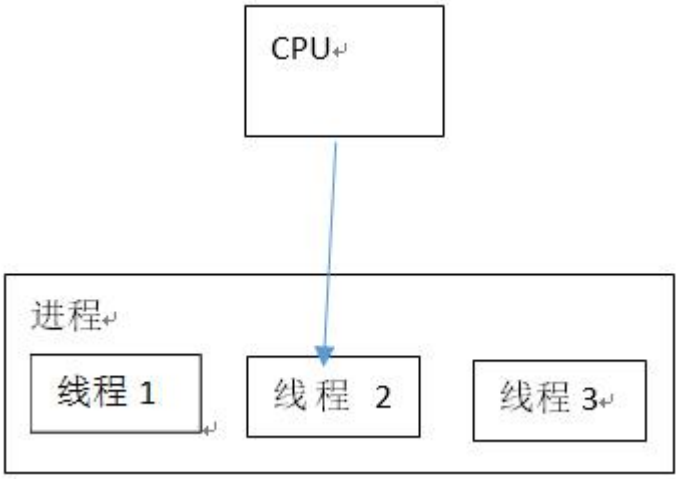
```
}
public static int sum(int ... p){
    return 0;
}
```
注意：参数 p 实际是一个数组，p 都是同一种类型

线程的实现

线程实现的第一种方式
创建新执行线程有两种方法。
一种方法是将类声明为 Thread 的子类。该子类应重写 Thread 类的 run 方法。接下来可以分配并启动该子类的实例。
线程启动的 时候使用线程的 start 方法而不是 run
另一种方法是声明实现 Runnable 接口的类。该类然后实现 run 方法。然后可以分配该类的实例，在创建 Thread 时作为一个参数来传递并启动.

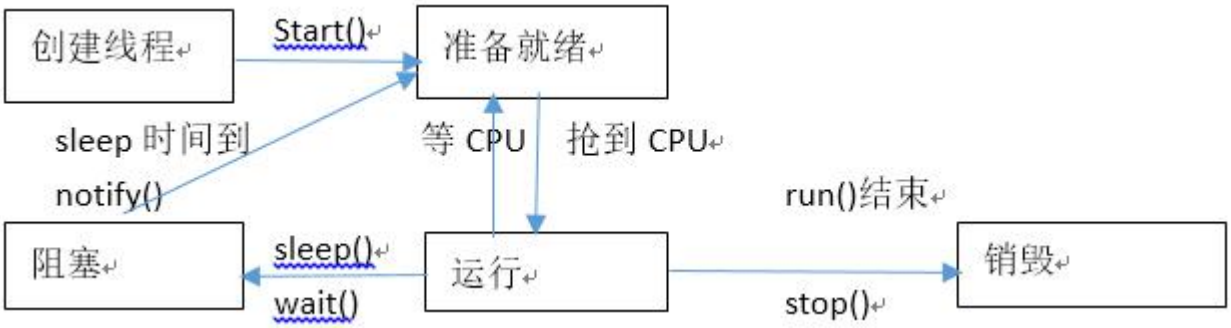
线程的执行原理

线程的并发执行通过多个线程不断的切换 CPU 的资源，这个速度非常快，我们感知不到，我们能感知到的就是三个线程在并发的执行。



线程的生命周期

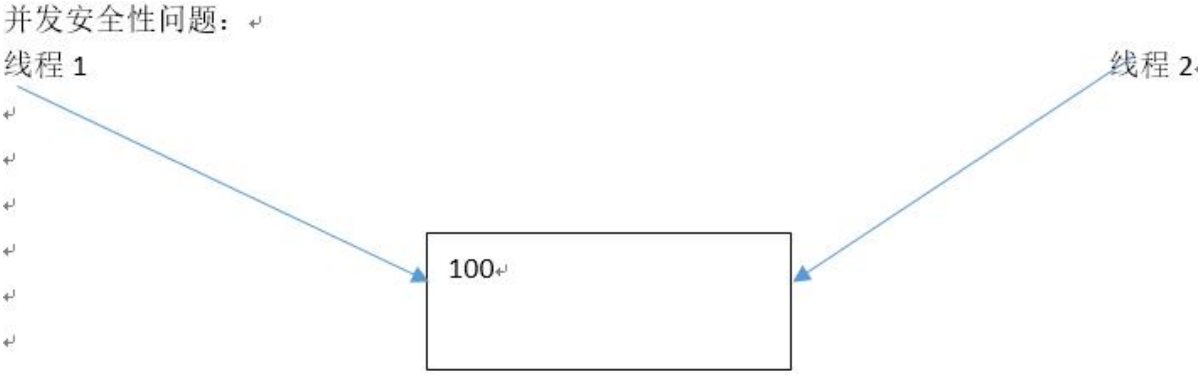
1.新建: 线程被 new 出来
2.准备就绪：线程具有执行的资格，即线程调用了 start()，没有执行的权利
3.运行：具备执行的资格和具备执行的权利
4.阻塞：没有执行的资格和执行权利
5.销毁: 线程的对象变成垃圾，释放资源。



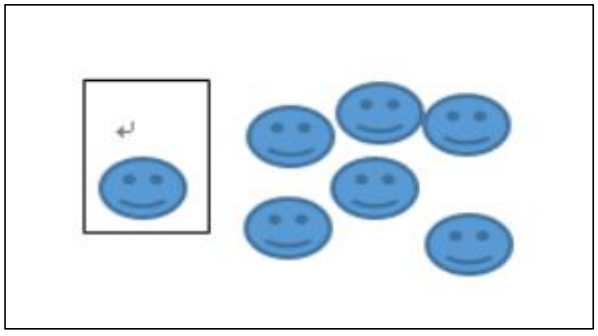
并发

互联网的项目中存在着大量的并发的案例，如卖火车票，电商网站。
范例：火车站有 100 张票，4 个窗口同时买票。
分析：4 个窗口是 4 个线程同时在运行，100 票是 4 个线程的共享资源。
采用继承 Thread 来实现

针对线程的安全性问题，我们需要使用同步(就是要加锁，共享资源只能一个人



同时访问)锁。



语法:
synchronized(锁对象){
//操作共享资源的代码
}
同步代码加在什么地方？
1.代码被多个线程访问
2.代码中有共享的数据
3.共享数据被多条语句操作
Synchronized 同步代码块的锁对象可以是任意类对象（线程的实现方式是使用继承于 Thread），这个对象必须是线程类共享（静态的）
Synchronized 是可以加在方法上，如果是静态方法 Synchronized 的所对象就是类的类对象，如果不是静态的方法，Synchronized 如果加在对象方法上，那么他的锁是 this

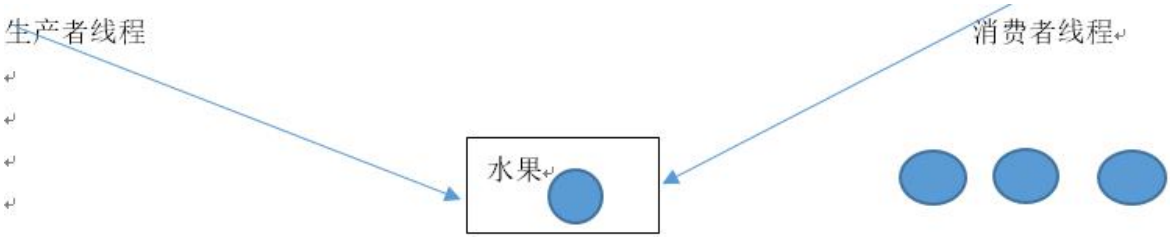
线程的休眠

线程的休眠是很必要的，在做服务端的时候为了减少服务器的压力我们需要休眠

如果休眠是在同步代码块中执行，休眠不会让出锁对象。

线程间的通信

生产者和消费者



生产者生成水，如果水果没有被买走那么就不生产处于等待状态，如果水果被消费者买走就的时候消费者会通知生产者告诉他我们已经把水果买走了请生产，消费者同理，如果水果已经生产出来那么就买走，买走之后再通知生产者水果已经没了请生产。

注意：

1.线程间的通信共享数据一定要有同步代码块 synchronized
2.一定要有 wait 和 notify，而且二者一定是成对出现。
3.生产者和消费者的线程实现一定是在 while（true）里面