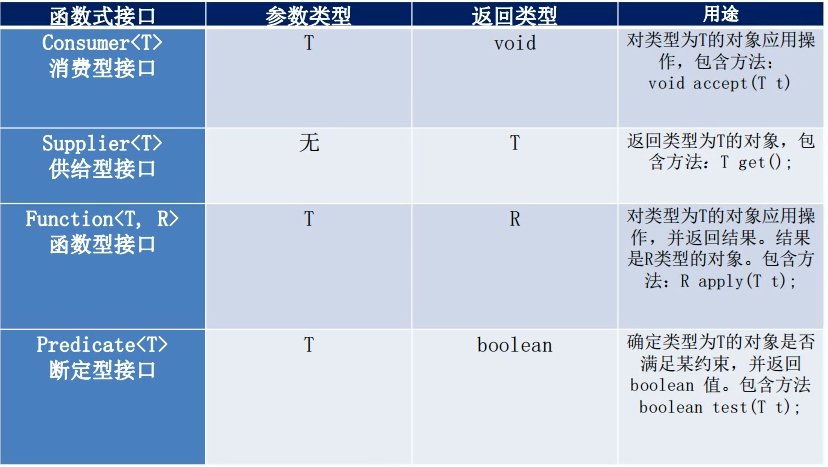
## JUC

1,，wait放锁，sleep不放锁

1. java多线程
   1. 高聚低和前提下，线程操作资源类
   2. 判断/干活/通知
   3. 多线程交互中，必须要防止多线程的虚假唤醒，判断只能用while，不能用if
   4. 标志位
2. 线程8锁：如果同步方法加了静态static就是使用的类锁，所有对象用一把锁，如果没有加静态，则每个实例对象都有一把锁，获取或者需要使用类锁的线程和对象锁的线程不相关，即，使用对象锁的线程不需要使用类锁的对象释放锁，相当于两把锁，一把类锁，一把对象锁。所有静态方法用的是同一把锁，即类本身
3. ArrayList是线程不安全的，Vector是线程安全的
4. 读写分离：CopyOnWriteArrayList



1. HashSet的构造器直接new了HashMap，HashSet的底层就是HashMap：add方法调用了map中的put方法，HashSet只保存了key，value使用一个object常量填充，value并不重要，只要有key就可以，所以说hashSet底层就是HashMap。
2. HashMap:底层是Node类型的数组+Node类型的链表+红黑树
   1. 数组的初始大小是16，负载因子是0.75
   2. 扩容条件：负载因子0.75\*当前大小
   3. Hash哈希值是用key算出来的，所以会进行覆盖，因为哈希值一样，找到的位置也是一样的，所以hashMap不能有重复的值
   4. 单向链表转红黑树的条件是，单向链表达到8，或数据长度小于64，小于就进行扩容，否则就转红黑树，调用treeifyBin方法转为红黑树
   5. 只要是带hash的大多不允许重复值
3. 常见高级异常：
   1. ConcurrentModificationException:并发修改异常
4. Arraylist扩容的话会扩容为原来的一半，HashMap会扩容为原来的一倍
5. Callable和runnable接口的区别
   1. 是否有返回值
   2. 是否抛异常
   3. 方法名不一样
6. Callable必须在线程执行完成之后获取返回值，call方法只会调用一次
7. ReadWriteLock:读读可共享，写读要独占，写写也要独占，保证一次性数据操作，原子性
8. 阻塞队列
   1. 当队列是空的，从队列中获取元素的操作就会被阻塞
   2. 当队列是满的，从队列中添加元素的操作就会被阻塞
9. 阻塞队列实现类
   1. ArrayBlockingQueue:右数组组成的有界阻塞队列
   2. LinkedBlockingQueue：由链表组成的有界（但大小默认值为integer.MAX.VALUE）阻塞队列
   3. PriorityBlockingQueue:支持优先级排序的无界阻塞队列
   4. DelayQueue:使用优先级队列实现的延迟无界阻塞队列
   5. SynchronousQueue:不存储元素的阻塞队列，也即单个元素的队列
   6. LinkedTransferQueue:由链表组成的无界阻塞队列
   7. LinkedBlokingDeque:由链表组成的双向阻塞队列
10. 实例化时，等号= 左边是引用在栈，右边是实例对象在堆
    1. 每一个方法都会在栈中生成独一份的空间，方法中的所有变量都只声明在独一份的空间，变量的赋值操作，除了实例引用调用set方法，是会修改实例指向的位置，但是普通引用例如声明一个String变量，该变量的值就存在当前执行时方法所在的栈中，每次调用方法都是独一份栈空间，相互之间不受影响。例如：同一个方法之间比如递归，每次调用一次即就在栈中开辟新空间，所以，每个方法之间的变量互不影响，如果方法执行完毕，则会释放该方法所占用的栈空间。
11. 线程池的优势
    1. 线程池所做的工作只是要控制运行的线程数量，处理过程中将任务放入队列，然后在线程创建后启动这些任务，如果线程数超过了最大数量，超出数量的线程排队等候，等其他县城内执行完毕再从队列中取出任务来执行
    2. 主要特点：线程复用，控制最大并发数，管理线程
    3. 降低资源消耗
    4. 提高响应速度
    5. 提高线程的可管理性
12. 线程池的工具类底层跟阻塞队列有关
    1. Executors.newFixedThreadPool(5); ：底层是LinkedBlockingQueue<Runnable>();
    2. Executors.newSingleThreadPool(); ：底层是LinkedBlockingQueue<Runnable>();
    3. Executors.newCachedThreadPool(); ：底层是SynchronousQueue<Runnable>();
13. 线程池的七大参数
    1. CorePoolSize:线程池中的常驻核心线程数
    2. maximumPoolSize:线程池中能够容纳同时执行的最大线程数，此值必须大于1
    3. keepAliveTime:多余的空闲线程的存活时间当前池中线程数量超过corePoolSize时，当空闲时间达到KeepAliveTime时，多与线程会被销毁，直到只剩下corePoolSize个线程为止
    4. Unit:keepAliveTime的单位
    5. WorkQueue:任务队列，被提交但尚未被执行的任务
    6. threadFactory:表示生成线程池中工作线程的线程工厂，用于创建线程，一般默认的即可
    7. Handler：拒绝策略，表示当队列满了，并且工作线程大于等于线程池的最大线程数（maximumPoolSize）时如果来拒绝请求执行的runnable的策略
14. 线程池底层工作原理  
    
15. 不允许使用Executors工具类创建线程池
16. 四种拒绝策略
    1. AbortPolicy(默认)：直接抛出RejectedExecutionException异常组织系统正常运行
    2. CallerRunPolicy:“调用者运行”一种调节机制，该策略即不会抛弃任务，也不会抛出异常，而是将某些任务回退到调用者，从而降低新任务的流量
    3. DiscardOldestPolicy:抛弃队列中等待最久的任务，然后把当前任务加入到队列中尝试再次提交当前任务
    4. DiscardPolicy:该策略默默的丢弃无法处理的任务，不予任何处理也不抛异常。如果允许任务丢失，这是最好的一种策略
17. 四大函数式接口



1. 方法引用需要一个具体的实现方法
2. Stream流是什么，集合讲的是数据，流讲的是计算
3. 分支合并框架
4. 异步回调
5. 双亲委派的优势
   1. 避免重复被加载
   2. 避免核心类被篡改
6. 类的显示加载和隐时加载
   1. 显示加载
      1. classloader.loadClass( className)
      2. Class.forName( className)
   2. 隐时加载
      1. 继承
      2. 实现接口
      3. 域变量
      4. 方法定义
      5. 方法定义中的本地变量