## 集合

Collection继承于Iterator迭代器，是一种设计模式，也是一个对象，它可以遍历序列中的对象，而开发人员不需要关注序列的底层结构。“轻量级”对象，创建它的代价很小，只能单向移动，获取下一个，判断是否还有下一个，删除等操作

### List：都有什么类型？

### ArrayList和LinkedList的区别：

底层数据结构：ArrayList：数组；LinkedList：双向链表

扩容：ArrayList是之前的1.5倍；LinkedList没有扩容没有初始大小

线程安全：都不安全，非同步

Vector 和ArrayList比较：Vector 多了一个增长系数，线程安全（对于单一操作），

### Queue: 队列

Queue:继承于Collection，先进先出;底层数据结构：数组；

POLL和remove方法的区别在于，对于空的队列返回不同，poll返回null，remove会抛出异常没有这种元素；peek和element一样

Deque继承于Queue, 是双端队列，依然是先进先出，可以从两端增加和删除，获取数据。

ArrayDeque继承于Deque，底层数据：数组；两个索引表head，tail；扩容：保证是2的幂方， Remove是通过poll实现的

Stack 和 Queue：

Stack : 继承于Vector；先进后出；底层数据结构：数组

## 2-Map

### HashMap：

1. 底层数据结构：数组+链表（1.7）；数组+链表+红黑树（1.8）；
2. 扩容：原来bucket的2倍；牵涉到rehash，复制数据，耗性能
3. （1.7）需要优化点：冲突严重时，链表越来越长，查询效率会变低，时间复杂度O（n）。(1.8）:优化后O（logn）查询效率提高；
4. 为什么是线程不安全？答案：在多线程环境下，存在多个元素进行put，如果hash相同，相同的key可能会添加到同一个位置，其中一个线程中的数据丢失，或者扩容时多个线程都在rehash，可能会导致一个线程成功，其他线程都失败，丢失数据，会出现死循环。
5. get的实现？put的实现？hash的实现？

Put：计算hashcode，得出index，未碰撞，放入bucket中，碰撞了，放入链表，如果链表过长，放入树中。

Get：通过key，一次命中返回值，或者从表中或者树中查找值

6，数组存储的是：链表，链表中的每个节点就是哈希表中的值

### ConcurrentHashMap：

1. 底层数据结构：数组+链表（1.7）；数组+链表+红黑树（1.8）；
2. 锁机制：（1.7）分段锁：ReentrantLock一个线程占用segment，不会影响到其他的segment；（1.8）CAS+Synchronized 保证线程并发安全性。
3. 核心数据和链表都是volatile（可见性，不能保证原子性）修饰，所以在put时依然需要加锁保证原子性

## 3-Jdk版本更新

## 4-String，StringBuffer，StringBuilder

String: 字符串变量：理论上不可变，因为里面的值被final修饰，也没有set方法；都是创建新的对象，即使是拼接或者修改，引用指向new 值；频繁的修改值，堆内存就会被分配的越多，内存不足时会影响GC工作

后两者：都是字符串变量，当值发生修改时不会产生新的对象，区别在于StringBuffer是线程安全的，里面的部分操作被syncronized修饰，

## 5-数组：初始化类型？如何分配内存？什么是引用对象，引用变量？

答：两种：静态初始化，动态初始化；系统在堆内存中分配一块连续的内存给数组；基础数据类型是存在内存中的栈区，引用数据类型是引用存在栈区，而引用指向的值是存放在堆中的。

## 6-什么是重写，覆盖，重载？三者的区别？

都是针对方法的。

重写，覆盖（Override）：返回值，方法名，参数的顺序，类型都要完全一致，里面的函数重新实现。父类与子类之间的多态性的体现。

重载：返回值，方法名完全一致，参数列表不同的函数实现。类内部多态性的表现；

## 7-extends，implements区别

1. 当抽象类中包含有非抽象方法，那么只可以extends，不可以implements

## 8-抽象类，普通类，接口？

抽象类，接口：不可以实例化，抽象方法和接口中的方法只有声明，没有方法体。子类需要实现抽象方法，接口中的方法，不然子类需要定义为抽象类。两者都没有变量（默认是public static final）。

抽象类，普通类：有构造函数，可以有普通方法

抽象方法只能存在于抽象类或者接口中，不可以存在于普通类。

## 9-BIO、NIO、AIO 有什么区别？

1. 流的种类？

流的流向：输入流，输出流；

操作单元：字节流，字符流；

1. BIO面向流，阻塞io，NIO面向缓冲区，不阻塞io，有选择器；
2. 概念的理解：

BIO：当一个线程进行读或者写的时候，这个线程是被阻塞的，直到请求结果返回。这个线程在此期间什么事情也不会去做。一个请求一个应答；

NIO：将这个任务分成多个小任务，由一个线程所有的io事件，并负责分发，利用事件驱动机制，事件到了再触发，例如：线程调用读/写，不需要等待读/写完成，这个线程可以同时去做别的事情。

## 10-多线程

### 10-1：并行与并发？

并发：当有多个线程在操作时，如果系统只有一个cpu，则它不可能同时进行一个以上的线程，只能把cpu划分成等量的时间片，分配给线程，在一个时间片一个线程进行，其他的线程处于等待状态。

并行：系统中有多个cpu，有可能并行，一个cpu执行一个线程，另一个cpu执行另一个线程。

### 10-2：守护线程，？

是用来服务于用户线程（普通线程），不需要上层逻辑介入。

当线程只剩下守护线程，jvm就会退出；如果还有任意一个用户线程，jvm就不会退出；不能去访问固定资源，因为他会在任意时间发生中断，典型例子：垃圾回收站;

### 10-3：thread中的run和start方法？

Run：一个普通方法的调用；

Start：会创建一个新的线程并启动

### 10-4：thread和runnable关系？

Thread实行了runnable，使得run支持多线程，由于Java的单继承，推荐多使用runnable

### 10-5：sleep和wait区别？

Thread的sleep只会让出cpu，不会导致锁行为的改变

Object的wait不仅会让出cpu，还会释放已经占用的同步资源锁

### 10-6：notify和notifyAll的区别？

notifyAll：唤醒所有处于等待池的线程全部进入锁池中去竞争获取锁的机会

notify：随机唤醒一个处于等待池中的线程进入锁池中去竞争获取锁的机会

### 10-7：如何处理线程的返回值？

1. 主线程等待法；
2. 使用Thred类的join（）阻塞当前线程去等待子线程处理完毕
3. 通过callable接口实现，通过FutureTask 或者线程池中获获取

### 10-8：线程的六个状态

1. 新建（new）：创建后未启动的线程的状态
2. 运行Runnable：running和ready
3. 无限期等待：不会被cpu分配事件片，需要显示的唤醒；（Object的wait；Thread的join();LockSupport.park()）
4. 限期等待：在一定时间内会被系统自动唤醒；例如Thread的sleep方法，Object的wait（时间），Thread的join（时间），LockSupport.parkNanos()….LockSuppport.parkUntil()
5. 阻塞：等待获取排它锁（写锁）
6. 结束：线程已终止，结束执行

### 10-9：Runnable和Callable区别？

Runnable：可以直接被thread执行，但是没有返回值

Callable：执行之后有返回值，需要用线程池来执行。

### 10-10：四种线程池？

Execuroe接口

线程池来源：当并发的情况多的时候，而每个线程请求时间很短，频繁的创建和销毁线程，会大大降低系统的效率。那么有没有一种方法可以在执行完一个任务，并不销毁，还可以继执行别的任务。

什么场景下使用线程池：单个任务处理时间短；需要的任务处理量大；

好处：降低资源消耗（通过重复利用已创建的线程来降低不断地创建销毁线程所带来的消耗）；提高响应速度（不需要等待创建线程）；提高线程的管理性（统一分配，调优，监控）。

newCachedThreadPool：可缓存线程池，如果线程池超过处理需要，可以灵活回收空闲线程，如果没有可回收的，就新建线程。

newFixedThreadPool：定长线程池，可以控制线程并发量，超出的要等待

newScheduledThreadPool：定长线程池，支持定时和周期性任务

newSingleThreadPool：单线程话线程池，

#### 10-10-1：五种线程池状态？

1. Running：可以接收新提交的任务，也能处理阻塞队列中的任务；
2. Shutdown：关闭，不再接受新的任务，却可以继续处理阻塞队列中任务;
3. Stop：不接受新的，也不处理队列中的任务，会中断正在处理的任务；调用shutdownNow
4. Tidying：所有的任务执行完，workerCount（有效线程数）变为0，
5. Terminated：什么也不做进入这个状态的条件是：不是running，不是tidying或者terminated，是shutdown并且workerQueue为空，workCount为0，设置tidying为true

#### 10-10-2： submit()和 execute()方法有什么区别

接收参数不同，submit有三种；

Submit有返回值，execute没有

Submit会抛出异常

### 10-11：如何保证多线程安全性？

使用同步synchronized或者lock

只允许单个线程执行。

### 10-12：CAS技术？

缓存锁：当物理机计算机中发出现数据缓存不一致的时候，之前是采用总线锁，锁住cpu和内存之前的通信，其他处理器不能处理其他内存地址的数据，所以改为缓存锁来优化。实现缓存一致性协议：每个处理通过嗅探在总线上传输的数据来检查自己的数据是否过期。设计到缓存锁主要是cas技术

CompareAndSwap：内存地址（主内存变量的内存地址），旧值（工作内存缓存的），新值。当且仅当主内存中的值等于旧值才更新旧值。否则不执行。无论是否更新，都会返回旧值。

### 10-13：Monitor（jvm）对象监视器

偏斜锁，轻量级锁，重量锁

如果monitor的进入数为0，则该线程进入monitor，然后进入数设置为1，该线程为持有者。如果已经被其他线程占用，那么该线程进入阻塞状态，直到进入数变为0；

### 10-14：死锁？

产生现象：两个或多个线程在争夺临界资源，发生相互等待锁的现象，若无外力作用，他们都将无法继续推进下去。

原因：系统资源不足；进程的调度顺序不对；资源分配不相等

如何防止：确定好顺序；超时释放

### 10-15：synchronized实现？

底层原理：

## 11：反射

### 11-1：what？

动态的获取对象信息以及动态调用对象的方法的功能称为java的反射机制。

划重点：运行时

### 11-2：getFields（）和getDeclaredFields（）区别：

前者是：返回该类或接口的可访问public字段

后者：返回该类中或者接口的受保护protected，公共的，默认的，私有的字段，但是不包括继承的字段。

### 11-3：用途？

在开发各种通用框架。例如Spring都是配置化（例如通过xml文件配置bean），为了保证框架的通用性，需要动态加载配置文件中不同的对象或者类，调用不同的方法就需要用到反射.

### 11-4：序列化？

What：将java对象序列化为二进制文件。

反序列化：将字节流（二进制文件）转换成Java对象。

How：实现serializable接口，使用ObjectInputStram ObjectOutStream对对象读写

When：当java对象需要在网络上传输或者 持久化的存储到文件中。

不能被实例化：被static ，trasient声明的成员变量

### 11-5：代理

代理模式是设计模式的一种，提供了对目标对象的额外访问方式，也就是说：通过代理对象访问目标对象，这样可以在不修改原目标对象的前提下，提供额外的功能操作，扩展目标对象的功能。

分为两种：

1. 静态代理：

方式：代理对象和目标对象都要实现一样的接口

缺点：冗余（由于要实现一致的接口，会导致过多不需要 的代理类）；不易维护（接口的方法增加时，目标对象，代理对象都需要修改）

1. 动态代理由于利用了JDK API，动态的在内存中构建对象，从而实现对目标对象的代理过程。又被称为：jdk代理，接口代理。 特点：代理对象不需要实现接口，但是目标对象必须实现接口，否则不能使用动态代理。设计到的类：反射中的Proxy,InvocationHandler。
2. cglib（Code Generation Library），第三方代码生成类库。运行时在内存中动态生成一个子类从而实现对目标对象的功能扩展。

特点：1，JDK实现的动态代理，目标对象必须使用一个或者多个接口，如果想要实现没有接口的目标对象功能扩展，可以使用cglib。2，强大的高性能的，可以在运行时扩展java类，与实现接口，例如Spring AOP，DYN AOP ,的拦截。3,无需实现接口，通过生成类字节码实现代理，比反射快，不存在性能问题，要继承目标类，需要重写方法，不可以final

静态代理和动态代理区别：

* 静态代理在编译时就已经实现，编译完成后代理对象就是一个实际的class文件
* 动态代理是在运行时动态生成，即在运行时生成类字节码class文件，并加载到JVM中。

## 12：克隆

What：在内存中将一个已有的对象复制出另一个与之相同的对象；

种类：

浅拷贝：

使用默认的clone方法，对于原始数据的拷贝，如果是基础数据类型拷贝的是值，如果是引用型，那么拷贝的就是就是“引用”。执行快，效率较深拷贝高。数据是相关的。什么时候推荐使用：对象不需要修改。

深拷贝：

重写clone方法，不仅仅只调用父类的c方法，还需要调用调用clone方法；可以做到克隆数据和元数据分离，什么时候推荐使用：如果对象存在引用型。耗时，效率低。

方法：例如：复制构造器（可以浅拷贝，也可以深拷贝），

## 13，JVM

### 13-1：内存：

1. 栈：存放基础数据类型和对象的引用
2. 堆：用来存放所有的Java对象
3. 方法区：包含所有的class和static变量