1. 集合（HashMap、ConcurrentHashMap、各种List，最好结合源码看）

Java集合框架：

Java集合主要分为两大类，一类为Map，一类为Collection

Map的存储特性只要为key-value的存储方案，除了HashMap之外,其他几个实现类的key都不能为null，key的值最好为不可变对象(如:基本数据类型的包装类和String)，因为如何key为可变的对象，其hash值如果被修改，则会出现存储在map中的值无法找到的情况。

Value对于TreeMap和HashMap来说可以为空，但对于CurrentHashMap和HashTable开说是不可以为空的。

Collection分为List和Set以及Queue。List的特性为元素可重复，Set的特性为元素不可重复。Queue的特性是具有队列的特点。

HashMap的底层实现：

jdk1.8之前其底层采用了数组+链表的存储结构，链表的存在主要是使用拉链法来解决元素冲突，但在1.8之后，其采用了在链表的元素超过8个之后，变转为红黑树的存储结构，这种结构在查询的时候时间复杂度为O(logn),极大的提高了在map集合元素较多的情况下的查询效率。

Map的并发安全：

HashMap不是线程安全的，对个线程的同时操作可能导致出现死循环的情况，在这种情况下可能CPU的占用率高达100%。所以jdk1.5之前java采用了HashTable来代替HashMap在多线程情况下的使用，然而HashTable采用在synchronized方法来实现并发，这种方式导致了同时只能有一个线程对HashTable进行操作，导致效率非常的低。随着jdk1.5之后java.util.current包的出现，各种并发包装类出现了。Jdk1.5使用了CurrentHashMap来代替HashTable,这个包装了采用了分段式（segment）加锁的方式，允许多个线程同时操作HashMap,每一个segment都自带一个ReentLock，这样极大的提高了HashMap的操作效率。在jdk1.8之后，java抛弃了分段存储的方式，而直接采用node的存储结构，并配合使用CAS(Compare And Swap)，Synchronized的方式来进行对HashMap的安全控制。

并发和多线程：

线程池：java.util.current.Excutors包中提供了java.util.current.Excutor接口来实现线程池，他的主要作用是对一定数量的线程的生命周期进行管理（包括创建，销毁和响应中断等等）。

Java线程池有4种实现：

newSingleThreadExcutor：只有一个线程的线程池，线程任务串行执行，实用场景主要是需要保证线程的顺序执行的情况下采用。

fixedThreadPool:指定一定数量的线程（一直保持线程活跃）和指定最大允许的线程数量（数量超过该线程后会将线程加入到队列中），这个线程的使用场景为负载较大的服务器，其限定了线程的使用创建数量，防止线程过多导致服务器崩溃

cachedThreadPool：采用无限制的队列来容纳线程，如果任务的处理速度低于接收任务的速度，该线程池则会不断的创建线程，这样可能导致线程过多而耗尽CPU和内存资源。这种线程池适用于任务不多，且任务周期较短的场景。

线程池的调度过程(fiexdThread)：

1线程池启动之后会直接创建核心线程数量的线程

2任务调用excute方法之后，线程池会去判断线程池中任务的数量是否超过了核心线程的数量，没有如果超过则直接采用当前的空闲线程去执行任务

3 超过核心数量之后会判断是否超过了最大数量的线程数，没有就创建一个非核心线程执行任务，在任务完成之后，达到指定时间的空闲非核心线程会被回收

4 超过最大线程数的任务将会被非配到队列中进行排队，线程池采用不断轮询的方式从队列的头部取出任务执行

synchronied java关键字，底层采用jvm软件层面来实现锁，通过锁住一个对象来实现同步策略。在jvm底层实现中，每一个对象都是一个监视器，这个监视器包含多个队列:Contention List,所有请求锁的线程将被首先放到该竞争队列，Eentry List,所有具备执行资格的候选线程会被放入到这个队列，Wait Set调用wait方法阻塞的线程被放入到Wait Set ,OnDeck任何时刻只有一个线程正在竞争锁。Synchornied 是可重入锁和非公平锁，他在先线程开始竞争锁的时候为了减少线程被阻塞之后的系统调用开销，采用了在一定时间内去自旋的方式去等待锁，因为先synchronied的自旋是在刚进入Contention List时候开始的，所以，刚进入竞争队列的锁更有机会获取到锁，所以其实非公平的。Synchornied是不能响应中断的，一个线程在持有锁的时候必须等待任务执行完成之后才能进行释放，可能会影响效率。但是synchronized在任务执行完成之后会自动释放锁。

Lock 是在硬件层面依赖特殊的CPU指令，lock是一个抽象类，底层有多种实现类，他们支持线程中断，锁的超时释放，公平锁等功能。在需要用到这些功能的时候可以使用lock,lock不会自动释放，需要自行unlock.

线程的通信是通过一个共享变量来完成，其中包括了wait,notify,notify all。这三个方法都是Object的方法，因为线程的等待或唤醒都必须持有锁对象，由锁对象来调用这三个方法。因为在线程通信的时候会出现假唤醒和信号丢失的情况，所以不能使用字符串常亮和全局对象来做为线程的锁。

Volidate: volidate是java提供的一个轻量级的线程安全关键字，它能保证变量的可见性和禁止指令重排序的功能，但是他无法保证操作的原子性，所以volidate必须在特定条件下才能保证线程的安全。实用volidate需要满足两个条件：1对变量的写操作不依赖于变量的当前值，2变量不会与其他状态变量一起纳入不变性条件之中（if a > b）.volidate可以用在一下场景：状态标志if（volidate变量）2.通过volidate可以安全的发布一个不可变量，保证变量的完整性

ThreadLocal 通过创建一个变量的副本来避免多个线程的安全问题，这种变量的实用场景主要是适用于不同线程获取对象后需要对当前对象进行处理，然后用完了就可以抛弃......

好像还不是太懂......

CyclicBarrier java.cuurent.util中的栅栏，它的功能是让多个线程达到某个位置的时候等待其他所有线程到达后才执行。他的底层是通过ReentLock和AQS实现的。它维护一个ReentLock,一个条件队列Condition(基于AQS),一个Generation,参与线程的数量parties和等待的数量count。在一个线程达到指定位置的时候进行count--操作，并将线程置为等待，判断count是否为0，如果为0则将队列中等待的节点插入到同步队列（AQS）之中,然后调用breakBarrier()方法（实际调用的AQS的signalAll方法唤醒所有的线程）。在执行了breakBarrier()方法之后Generation会被赋予一个新值，这意味着CyclicBarrier是可以重用的。

CountDownLatch :它具有计数器的作用，它能使某个线程在一个或多个线程完成之后才执行。

AQS （AbstractQueueSynchronizer）java.util.lock的底层实现的数据结构，其维护一个双向链表，其中存储包含线程的node对象，它包含Syn和NoFairSync和FairSync三个内部类，Syn为抽象类，其他两个分别为公平锁和非公平锁的实现类。AQS添加元素，通过循环和validate int args 、CAS的方式去将元素插入到队列的尾部，在当前线程释放锁的时候，公平锁会使等待队列的第一个节点去获取锁（FIFO），但非公平锁会使刚进入队列的节点去获取锁。

CAS(CompareAndSwap) 乐观锁的实现原理，变量必须为volidate，通过不断循环的方式，每次执行时首先获取当前最新的变量值，使其成为预期值，然后调用compareAndSet（底层调用JNI来完成CPU指令操作，这是原子性的）与当前值进行比较，如果相同则进行赋值操作。这种方式的缺点是：1.会不断的循环会导致消耗CUP，如果JVM支持CPU提供的Pause指令会使效率有所提升。2.有可能会导致ABA问题，解决方法是设置版本号，每次update操作之后都使版本号+1,每次判断需要判断版本号是否符合预期。3.数据只能是单个类型。这两个问题java都通过AtomicRerence来解决了。

IO : 同步阻塞IO，在客户端与服务端交互中对应着一个客户端请求对应一个线程，这样的缺点是在并发量较大的情况下，服务器开启过多的线程，导致内存资源消耗过大，服务器响应缓慢甚至崩溃。

NIO: jdk1.4版本之后推出的同步非阻塞IO模型，其采用了Reactor模式，采用客户端和服务端分别维护一个Selector对象,该对象可以检测一个或多个Channel(全双工的数据通道)上的数据。客户端和服务端分别向多路复用器（selector）上注册事件，selector监听到事件

之后

Spring IOC ：其大致的初始化过程分为资源定位，解析，注册。如FileSystemXmlApplication的容器创建过程由refresh()方法开始，调用父类的构造方法为容器设置好资源加载器，具体实现在父类的父类AbstractApplicationContext中进行，再调用父类AbstarctRefreshableConfigAppilcation的setConfigLocations对资源进行定位，FileSystemXmlApplication在资源定位好了之后通过父类的refresh方法对资源进行载入，该类为资源的载入做了一些准备工作之后，便调用子类AbstarctApplicationContext的obtainBeanFactory()方法，这个方法又调用了其子类的AbstaracrRereshableApplicationContext的freshBeanFactory()方法，该方法创建了IOC容器并设置了一些启动参数，并调用了子类AbstarctXmlApplicationContext的loadBeanDefitions()方法，该方法创建了一个资源加载器XmlBeanDefinitionReader,为该加载器设置了SAX xml解析器，将解析器设置到容器当中，并再次调用了loadBeanDefinition()方法，该方法会去加载资源，在这个地方不同的的Spring容器会有不同的实现了，因为FileSystemXmlApplication已经对及资源加载过了，这个调用getResource()方法将直接返回空，而直接调用之前加载好的资源定位。调用reder.loadBeanDefinitions(configLocatins),XmlBeanDefinitionReader的父类AbstarctBeanDefitionReader定义了该方法的实现，首先获取之前设置的资源加载器，将之前加载的资源文件改变为IOC容器封装的资源，并委派XmlBeanDefinitionReader进行资源的加载，XmlBeanDefinitionReader又调用父类DefaultResourceLoader的getResource()方法获取资源，FlieSystemXmlApplicationContext提供了getResourceByPath()来解决既不是类路径也不是url的资源方式。XmlBeanDefinitionReader得到资源之后便使用loadBeanDefinitions()方法对资源进行解析，DocumentLoader将资源转化为Documnet对象，得到dom对象之后XmlBeanDefinitionReader边调用registerBeanDefinitions()对dom进行符合Spring规则的bean解析，具体过程是通过DefaultBeanDefinitationDocumentReader进行，解析的操作是BeanDefinitionParseDelegate进行，解析好的document对象便是包含了Bean的所有信息的BeanDefinition对象，再调用BeanDefinitionReaderUtil向IOC容器进行注册，底层采用HashMap对象进行存储。在第一次调用容器的getBean方法之后，或者在配制文件中配制lazy-init之后容器才会真正的生成一个实例对象。通过beanFactory生成单例对象，FactoryBean是容器对象本身。创建对象的过程一般是通过JDK反射的set方法进行 。

SpringMVC 处理请求的过程 ：前端适配器接收到浏览器的请求之后，调用处理器映射器（HandlerMapping）进行映射，处理器映射器根据请求地址返回一个执行链（包括拦截器链和Handler），前端映射器再获取一个处理器适配器，首先调用拦截器方法执行拦截器preHandler()，再通过handel()方法执行处理器的方法，在这之前还需要经过一些数据转化，数据格式和数据验证的工作。然后handel()方法返回来一个视图模型(ModelAndView),再执行拦截器的postHandler()方法，然后将视图模型交给视图解析器进行数据和模型的解析和渲染，将渲染后的结果返回给浏览器，一个请求就完成啦。

Java异常：java异常有一个公共的祖先(Throwable)，然后又分为Error和Exception两种类型，Error主要包含了比较严重的程序错误，这种错入一般与代码无关，比如虚拟机的内存溢出（OutOfMemoryError）类定义错误（NoClassFoundError）,这些都是不受检查的异常，一般也不需要程序处理。Exception包含RuntimeException，这种是程序执行过程中代码问题导致的异常，如（NullPointException,IndexOutOfBoundsException）,这种异常是不受检查的异常，可以进行捕获处理，应该仔细检查代码避免这种异常的出现。除了不受检查异常之外还有就是受检查异常，必须要显示的抛出和处理异常，不然编译器是不允许通过的。这种异常包括IOException,SQLException;

Redis:Redis包含多种数据结构，Key(只存放一个键，可以设置过期时间)、String（键值对的形式，一个键对应一个值）、List（队列）、Set(包括有序和无序两种)、Hash（包含多个键值对）。Redis是单线程的，所以可以用来处理很多的并发情况，比如秒杀系统、分布式锁。Redis的查询速度非常的快（内存操作）所以很多时候使用Redis来做缓存。在数据存储方面，Redis有两种存储方式，RDB这种方式是通过一定时间间隔直接存储内存快照的方式来进行数据存储，在存储的时候会启用一个子线程（fork）来进行同步存储。这种存储方式的缺点是会丢失部分数据，但是是Redis的默认存储方式。还有一种存储方式是AOF,这种方式是通过命令的追加来进行持久化，这种方式的优点是更加安全，数据丢失不会很严重，但这种方式的的缺点是需要占用更多的磁盘空间，而且在数据还原的时候更慢。

Mysql常用的两种引擎:InnoDB、MyISAM。区别：InnoDB支持事务，MyISAM不支持事务；InnoDB支持行锁，MyISAM只支持表锁；InnoDB支持Hash索引，MyISAM支持全文索引，MyISAM的数据可压缩，而InnoDB支持外键。InnoDB的底层采用B+树为数据结构，B+相对于B-树来说具有更低的高度。所以在磁盘查找中普遍使用B+树作为存储结构。InnoDB有两种索引方式，一种是聚簇索引，一种是非聚簇索引。聚簇索引是以主键为索引构建的一个B+树，在叶子结点存储着数据行的所有数据，因为是以主键顺序为索引，所以聚簇索引只能有一个。但是可以有多个非聚簇索引，非聚簇索引构建的B+树中根节点存储着主键，找到主键之后再通过主键的索引取出对应行的数据。