一 、数据库

* mySql 数据库优化

1 设计使用最优字段类型

字段长度尽可能最小，如预知字符长度精确设置；

字段值最好设置非空，以避免查询空比较而降低效率；

使用值类型代替其他类型，如使用性别可设置为int，enum类型。

2 使用连接查询代替子查询

因为子查询产生嵌套执行效率非常低，尽量杜绝使用，而mysql中连接查询不需要在内存中创建临时表执行操作效率更高。

3 事务执行尽可能要短

涉及到必需使用事务进行数据更新处理时，尽可能将事务执行缩短以提高效率。

4 适当使用存储过程

在不涉及数据库迁移时刻考虑使用存储过程进行数据操作，因为存储过程是预编译的可大大提高执行速度。

5 合理使用索引

数据库表中存在大量数据时，可在表中合适字段上建立索引从而提高查询速度。

建立索引基本原则：

1. 字段值很少被修改或永远不会被修改适合建立索引；
2. 经常作为查询条件的字段考虑优先建立索引，但应兼顾考虑原则一；
3. 存在大量重复数据的字段不适合创建索引,相反可能会降低执行效率；
4. 索引的数量在一个表中不宜过多，因为索引页需要被维护。

6 优化sql 语句

尽可能不在不同类型的字段上进行比较运算；

为了发挥索引的效率，sql中尽量避免在建立索引的字段上使用函数操作，会使的索引失去作用；

应最好避免过多使用like 模糊查询，not in ，not between 进行查询操作；

查询字段只选取需要的而杜绝使用select \* 方式；

根据mysql服务器引擎类型(常用Inno和ISAM)优化使用count聚合统计函数等等……

* 关于存储过程

存储过程是数据库系统中预定义的sql语句集, 类似于java中的方法。适用于相对复杂的数据操作并能够大幅度提升执行效率。优点是重用性高，执行速度快；缺点是在不同数据库系统间没有兼容性，无法移植。

* 数据库事务

事务是为了在数据操作过程中确保数据最终的安全正确性一种机制，事务具有原子性、隔离性、一致性和永久性特点(ACID)，是很多sql语句的独立执行的逻辑单元。事务是一个不可分割的整体，要么正确执行全部提交，要么任何执行失败进行撤销事务(如常见的银行转账业务必需使用事务处理)。

延伸问题:

锁的概念：锁是确保数据被并发访问的一种数据库安全处理机制，如数据库事务就是主要依赖于锁机制实现。

锁的分类：总体上锁可分成乐观锁和悲观锁

**乐观锁**是用户自己实现的一种安全机制，通常认为并发访问的机率较小的情况下应用；乐观锁只在数据操作执行提交时才对数据进行检查，乐观锁效率相对较高，乐观锁主要使用版本号和时间戳方式实现。

**悲观锁**是数据库内置锁，通常认为一定产生并发数据访问情形下应用; 悲观锁在数据访问效率上比乐观锁要差,但是处理并发能力要强。mysql数据库系统大致将悲观锁分为表级锁、行级锁和页及锁，常用的ISAM主要实现表级锁，而INNO还实现了行级锁和页级锁；表级锁几乎不会出现死锁的情况，开销较小但是并发能力稍差，行级锁和页级锁开销较大但是处理并发能力更强，表级锁和行级锁还分为共享锁和排它锁(独占锁)，特殊情况下INNO中可以同时使用表级锁和行级锁，这样就产生了意向锁，也就是共享锁升级为排它锁。

* mySql 中的索引

mySql 中的索引通常为B+树索引,Hash索引，全文索引，R-树索引;

对于企业开发中使用最多的Inno引擎，通常使用最多的是B+树索引(效率最高)；Hash索引，全文索引和R-树索引在ISAM引擎中被支持；全文索引不支持中文检索，R-数索引用于物理空间数据类型，使用的极少。

B+树索引和Hash索引的比较

B+ 树索引的本质就是B+ 树在数据库中的实现, 是多路平衡查找树.对于每一次的查询都是从根节点出发,查找到叶子节点方可以获得所查键值（目标关键值数据）,然后根据查询判断是否需要回表查询数据;

hash索引底层就是hash表实现,进行查找时,调用一次hash函数就可以获取到相应的键值,之后进行回表查询获得实际数据。

B+树由于其树形结构特点支持范围查询搜索；而hash索引一般情况等值查找快而精确，但是无法支持范围查找；

B+树支持排序而hash索引无法支持排序，同理无法支持模糊查询和多列索引最左前缀的匹配；

Hash索引必需通过回表实现数据最终查询，B+数满足条件可以某些条件直接通过索引完成查询；

has索引在等值查询上较快,但并不稳定.性能无法预测,在某个键值存在大量重复的时候,发生hash冲突,效率可能极差，而B+树的查询效率比较稳定, 而且树的高度较低(通常2-4层)，对于所有的查询都是从根节点依次到叶子节点.

B+ 树索引主要分为聚集索引和辅助索引**。**

聚集索引是根据每张表的主键建造的一棵B+ 树，叶子节点中存放的是整张表的行记录。一张表只能有一个聚集索引。因为聚集索引在逻辑上是连续的，所以它对于主键的排序查找和范围查找速度非常快，对于mysql数据库表设计为发挥B+数索引检索效率，强烈使用自动增长列设置表的主键。

辅助索引与聚集索引不同的地方在于，辅助索引不是唯一的，它的叶子节点只包含行记录的部分数据以及对应聚集索引的节点位置。通过辅助索引来查找数据时，先遍历辅助索引找到对应主键索引，再通过主键索引查找(回表查询)对应记录。

B+数的联合索引

联合索引是指对表上的多个列进行建立索引。联合索引对对应多个列的指定获取比较快。另外一个好处是联合索引对第二个键已经做好了排序，所以对两个列的排序获取可以避免多做一次排序操作。

mySql哪些查询无法使用索引？

举例说明

使用不等于查询,如 where id <> 3420

列参与了数学运算或者函数,如 where age mod = 2

在字符串like时左边是通配符.如似于'%XX'.

当mysql分析全表扫描比使用索引快的时候不使用索引.

当使用联合索引,前面一个条件为范围查询,后面的即使符合最左前缀原则,也无法使用索引.

如 where age between 14 and 28 OR address = ‘本市’

可使用Explain 指向查询语句检测sql指向情况是否应用了索引设置及其他sql指向信息

Explain select \* from where age >= 18

* mySql数据库事务隔离级别

mysql Inno引擎类型数据库事务级别分为四种

1 READ UNCOMMITTED(未提交读)。

此隔离级别是事务间隔离级别最低，一个事务可以读取到另一个事务没有提交的数据，而另外事务最终进行了撤销操作，导致读取的实际不存在的数据，这种现象称为脏读。脏读是比较严重的问题，因此未提交读隔离级别使用的极少。

2 READ COMMITTED(已提交读)。

此隔离级别是一个事务能够读取 另一个事务的部分已提交数据，这个隔离级别会造成不可重复读现象(两次读取到的数据不一致)，因为另一个事务对已提交的数据进行了二次修改，导致其他事务两次读取数据不一致。

3 REPEATABLE READ(可重复读)。

此隔离级别是Inno类型引擎默认的事务隔离级别，是经常使用的级别，能够解决上述问题，但是仍然存在幻读的现象。在一个事务读取id > 45 的数据行时,对这些行加上了读锁, 此时另外一个事务新插入了一条id = 46的数据,因为是新插入的,所以不会发生上面的锁的排斥,在本事务进行下一次的查询时会发现有一条id=46的数据, 而上次的查询操作并没有此数据行,再进行插入就会有主键冲突的问题。

4 SERIALIZABLE(可串行化)。

能够解决以上所述所有问题，此种隔离级别最为严格，强制其他任何事务必需完全等待本事务彻底执行完毕从而也导致了并发性能的大幅度降低而效率低下，因此也应该慎重使用。

* Java 多线程

1 有哪些方式可以创建线程对象

简单方式可通过继承Thread类、实现Runnable以及Callable 接口；比较复杂的多线程应用中可使用线程池对线程对象进行管理

2 一个线程的生命周期是怎样的

一个线程对象从被创建开始，调用start方法就开始了此线程的任务，直到它执行完成任务死亡并被GC回收或在执行任务中意外终止以及受阻，通常可分为新建阶段，运行阶段，阻塞阶段，恢复运行，死亡几个阶段（如果对线程阻塞的原因和情况比较熟悉可细化阻塞情形）。

3 如何理解线程安全

线程安全的基本意义在于一个多线程应用程序中，不同线程存在同一时间访问共享数据的现象，程序中必需对此做出处理以避免数据访问过程中出现数据竞争而造成数据错误以及可能出现死锁现象的处理，线程安全也叫做线程同步。

4 java多线程应用中同步对象,同步代码块，同步方法有什么不同

从只考虑对象锁而言, 同步方法锁定的是当对象this，此类中使用synchronized 修饰的方法未被当前线程释放this锁定时无法访问；同步对象代码块和同步对象相对类似，通常锁定的执行代码逻辑在某个方法内，相对同步方法数据安全边界较小，同步对象或同步代码块若锁定的对象不是this，则不同线程仍然可同时竞争this的持有权限。

5 wait（），notify（），notifyAll( ) 线程同步方法为什么要定义在Object类当中而非线程类当中

java 完成线程同步最基本的依赖就是对资源的锁定，任何对象都可能成为被线程独占的资源，如果将这些同步方法定义在线程类当中，则无法使得应用程序中的任何对象拥有锁，这将极大限定线程独占资源的范围，锁将成为线程锁，而java中的锁是对象锁，线程通过持有此对象而获得此锁，从而拥有独占资源而实现线程同步

6 ThreadLocal 在什么场景下使用，主要作用是什么

ThreadLocal 是用来实现线程局部变量的管理，宗旨在于将此变量与某一个线程进行唯一关联，成为当前线程的内部独享资源，通常此对象不应存在外部的引用，在ThreadLocal中使用threadLocalHashCode来区分不同的ThreadLocal,其实现核心是使用内部静态类ThreadLocalMap来存储此局部变量对象，通常当前线程使用set和get方法设置及获取此局部变量对象

7 你知道哪些同步集合与并发集合，有什么不同

常用的同步集合有Hashtable、Properties以及向量集合Vector、其中Properties是Hashtable的子类；并发集合多数在java.util.concurrent包下，包括ConcurrentHashMap、ConcurrentSkipListSet、ConcurrentLinkedQueue等。

同步集合的更新方法使用synchronized 进行锁定整个列表项灵活性较差，效率比较低；并发集合使用底层AQS实现局部更新时检查同步条件，是局部锁定灵活性更强，效率也比较高，因此并发中提倡使用它们。

8 Java中synchronized 和 ReentrantLock 有什么不同

synchronized 是java中隐式实现的重量级重入锁，相对比较笨重不宜扩展，并发能力较弱，主要实现方式是缓存与JVM主内存的一致性管理；ReentrantLock是显示实现的重入锁，它是轻量级的锁定，使用比较灵活容易扩展，并发能力更强，可实现分段锁效果，此锁定方式主要依赖于AQS(AbstractQueuedSynchronizer)抽象工具框架而实现。

9 Thread 类中的holdsLock方法有什么作用，给定的参数具有什么含义

此静态方法用于检测当前线程是否对某对象持有锁，此方法参数是检测当前线程是否持有此对象的锁的目标，如果返回结果为true则当前线程对给定的对象持有锁，否则相反。

10 死锁的形成条件有哪些，如何解决死锁问题

形成死锁的条件有：

2个或多个线程存在循环等待形成死锁；

2个或多个线程存在资源竞争形成死锁；

某线程已经获得的资源(锁)不会被剥夺，而又竞争其他资源形成死锁；

请求与保持，一个线程在请求资源(锁)被阻塞时并一直等待，拥有资源(锁)的线程的状态不会改变。

其中不满足其中任意条件则不会形成死锁，也就是同步中要破坏死锁任意满足条件。

11 多线程程序中的上下文切换指的是什么

上下文切换通常指的是中央处理器对线程运行到阻塞，从阻塞到恢复的管理过程。

12 ReadWriteLock 这个接口有什么用处

ReadWriteLock接口是相对锁的规范，它提供读写相对的两把锁，其一是读取数锁，其二是写入数

据锁，其中读取锁可有多个线程共享，而写入锁只能由一个持有此锁的线程独占，也就是数据修改不可以共享。此接口提供的规范功能能够在并发中更加灵活和提高效率，在某个线程独占写入锁的修改资源时，其他线程可以通过读取锁查看锁修改的数据更新。此接口已存在的实现类是ReentrantReadWriteLock。

13 AbstractQueuedSynchronizer(AQS) 类在java多线程中的设计目的是什么

AQS 是1.5版本增加的一个以FIFO(先进先出)为原则实现依赖的等待队列阻塞锁以及同步监视器抽象框架组件。Java中多数分段锁实现基本都基于此类实现，如ThreadLocal、ConcurrentHashMap等类，此类使用唯一的原子特性的int值状态维护同步监视器。AQS默认支持独占模式同时也可以设置为共享模式；ASQ使用内部类ConditionObject作为Condition的实现，Condition可以看做是锁(Lock)的实现，实际应用中ASQ的实现应作为内部私有类存在。与Java默认的重量级synchronized实现相比，ASQ实现更能体现多线程并发的优势和灵活性。

* Java 读写操作

1 Java中用来表示文件和目录的是哪个类，是否允许被继承

Java SE 平台类库中的File类是唯一用来表示磁盘文件或目录的抽象映射，使用方法isFile()或isDirectory()检查是具体文件还是目录。

2 Java中常用的文件流按照文件类型大体可分为几类，分别是什么

Java中按照文件类型大体可分为常用的字节流和字符流，字节流基于字节处理读写操作，字符流基于字符处理读写操作。

3 File类中的mkdir()和mkdirs() 有什么不同

这两个方法都是用来创建磁盘目录，目录都是由当前文件对象的抽象路径而指定，二者区别在于

mkdir()只会创建最末端的目录并且要求其父目录(如果抽象路径中有)必需存在，否则创建失败；mkdirs()会完全依照文件对象描述的抽象路径创建多级目录，如果父路径存在则使用，如果不存在则首先创建父目录然后依次创建子目录。

4 文件读写完毕为什么一定要关闭文件流

Java 文件对象是对磁盘目标文件的映射，基于此文件建立的读写操作流就是操作物理磁盘数据，如果在处理读写操作完成后不关闭文件流，则目标磁盘物理文件一直处于被实际操作的状态，其他程序无法继续对此磁盘文件进行相关修改删除移动等操作，因此必须在文件流使用完毕后显示调用close方法关闭文件流。

5 什么是序列化和反序列化，有哪些限制条件

序列化是指将java应用程序中的对象保存到物理磁盘文件进行存储的操作；反序列化是将物理磁盘上的序列化对象从磁盘读取到JVM内存的操作。能够被序列化的java对象必需是序列化所属类型实例，此类通常实现序列化接口Serializable，序列化是对象操作与类无关，因此序列化类的静态成员属性及瞬时转态属性不参与序列化，只有实例属性才会参与序列化。

java io 包中ObjectOutputStream 提供将java对象写入到文件序列化实现，ObjectInputStream 则提供将文件中的序列化对象读取到JVM内存的反序列化实现。

6 Java 中NIO是什么

NIO 指非阻塞文件输入输出流，NIO主要由个核心组件构成。分别是Channel 代表通道，是实现非阻塞文件读写的核心是通往硬件的连接；Buffer抽象类是实现NIO文件操作时的缓冲区，它是一个线性系列，NIO中的Buffer可灵活分为读写缓冲区和只读缓冲区以增加数据安全读取，Buffer对于多线程操作是不安全的，应该在多线程中注意这个问题，Buffer还提供了更加简洁的链条式方法调用以方便操作；Selector 抽象类是NIO中Channel 的多路选择器，可用来以单线程处理多个Channel 。通常Selector对于多线程是安全的，因此在多线程文件读写操作中NIO使用更加灵活。

7 如果给定一个目录，如何实现对此目录下的子目录及子文件实现遍历操作

在未知目标目录有任意级子目录及文件时，通常可采用递归方式对目录下的文件进行迭代，如果是目录子递归如果是具体文件则返回递归结束

8 普通字节流与缓冲流有什么区别

普通的字节流通常基于目标文件直接建立并直接对目标文件进行读写操作。而缓冲流并非直接基于目标文件建立而是基于一个字节流建立，也就是缓冲流并非从目标文件进行读写操作而是基于缓冲区进行读写，对于较大的文件而言使用缓冲流进行操作会提高效率及处理的灵活性，因此建议使用缓冲流是有很多好处的。

9 NotSerializableException 异常类型实例什么时候会被抛出

NotSerializableException 是非序列化操作异常，在将java对象通过序列化流写入到目标文件时，如果此对象所属类未实现序列化接口Serializable则JVM会抛出此异常实例

10 EOFException 异常是哪种文件操作什么情况下会出现

EOFException 实例表示一个二进制输入流操作中的异常，当在对目标文件进行输入操作时，如果意外到达文件末尾则JVM会抛出此类型异常实例，如果在正常读取到文件末尾而返回某个值代表读取结束时不同的，如在反序列化过程中读取文件意外到达文件末尾可能会出现此异常

11 描述一下RandomAccessFile的应用

RandomAccessFile 是为随机读写操作提供的输入输出API，可以对目标文件同时进行写入和读取操作，这两种模式的确定决定于构建RandomAccessFile时给定的第二个字符串参数；RandomAccessFile中使用四个int常量值检测此实例是哪种模式操作，常量值分别为1只读模式，只支持对文件进行读取，任何对文件的写入都会发生IO异常；2读写模式，如果目标文件不存在则会首选创建此文件；4 读写模式，并保证目标文件在硬盘介质存在时将目标数据及元数据通同步更新到文件；8 读写模式，并保证目标文件在硬盘介质存在时将目标数据通同步更新到文件；

模式字符串参数对应int常量的标准为1 = “r” , 2 =”rw”, 4 = “rws”, 8 = “rwd”。如果模式参数不在此四种之内则抛出非法参数异常。由于同时存在对文件进行读写操作，此类提供还提供了安全管理检查，主要通过调用系统安全管理器的checkRead检查是否支持读取，调用checkWrite检查是否支持写入。

* 关于java5泛型特性

1总体描述一下泛型。

泛型被称为类型安全，简称为GJ，是Java5版本最为主要的新特性。泛型本身并不是数据类型而是一种Java5及以上版本支持的Java语法，泛型语法只在编译阶段进行检查，而在运行时恢复到未使用泛型的原生状态，Java内部使用擦除法在运行前将泛型擦除。

泛型的最大目的就是能够使传统的数据类型及参数的使用更加符合实际需求并具有非运行时动态特点，极其灵活；能够代替类似于使用Object等上级父类作为参数时，而在具体传递子类实现为实例的类型强制转换问题，对于返回结果同样如此，并对集合中元素的使用提供极大的方便。由于泛型是一种编译期语法，所以能够在编译阶段检查开发者编写代码时传递参数及使用的数据是否符合泛型要求，减少了出错的可能从而也提高了代码的编写效率。

2 泛型都可以出现在哪些地方

泛型可以应用在类与接口的定义中；

泛型可以作为类的成员实例属性；

泛型可以作为方法的参数与返回类型；

泛型经常用在规范集合存储数据元素。

3 泛型的上届和下届

泛型上届和下届使得泛型的参数类型更具有可扩展的弹性。

泛型上届和下届使用通配符?与extends和super关键字组合语法构成。

上届的含义是泛型有上限类型限制而无下限类型限制，语法规则是<? extends E>, E为上限类型限定, ?为符合是E或其子类型的泛型参数。

下届的含义是泛型有下限类型限制而无上限类型限制，语法规则是<? super E>, E为下限类型限定, ?为符合是E或其父类型的泛型参数。

4 泛型的使用限制

泛型的参数类型只能是类java所有平台预定义类型及开发者自定义类型，不能是简单值类型。

同一种泛型可以对应多个版本（因为参数类型是不确定的），不同版本的泛型类实例是不兼容

的，因为泛型不是一种数据类型。

泛型的参数类型可以使用 extends 或super语法，称为“有界泛型”。

泛型做为属性定义不可以使用static进行修饰。

泛型做为属性定义时不可以进行实例化，因为此阶段无法定义泛型的实际类型是什么。

5 简单描述一下泛型是如何被擦除的

在Java编译器对带有泛型的java代码进行编译时，首先会执行类型检查和类型推断，在确认正确后按照java规则生成普通的不带泛型的字节码文件，也就是上泛型代码片段必需被转化成一般的、不带泛型的规范Java 程序代码后再进行处理的，编译器自动完成了从 GJ 到普通 Java 的翻译，Java 虚拟机运行时对泛型其实一无所知也不会关心，此过程被叫做 type erasure（类型擦除）。

* Java 网络编程

1 TCP 和 UDP协议有什么主要区别

TCP和UDP同属于互联网信息共享交换IP协议。TCP称之为传输控制协议，是面向安全连接的一种网络协议，TCP协议具有很高的安全可靠性，它具有严格的验证和重发机制能够确保数据按照一定的顺序安全到达目标主机。

UDP称为用户数字报协议，是面向无连接的非验证网络协议，UDP以执行效率为特点通常不对数据进行检查和修改也无需与目标主机进行连接确认，所以不能保证数据的安全正确到达目标主机，也无法保证数据发送的先后顺序，可能出现数据丢失现象，与TCP相比UDP设计结构简单网络开销比较少传输速度快，但是不安全。

2 TCP是如何实现确保通信安全的(复杂)

TCP主要通过以下管理方式实现数据的安全可靠传输。

**校验和**。大体上是对数据段（16位整数）进行规则内计算发送方与接收方进行一致性验证。

**序列化和确认应答机制**。TCP将传输中的每个字节数据进行编码成序列号。在连接传输数据中会对对方进行确认并应答，其中主要是ACK报文，序列号包含在报文中，通过对ACK序列号验证以确定数据持续传输如何进行。

**超时重组**。超时重组确保在数据传输过程中由于网络等情况导致接收方迟迟不能接收到数据而重新发送数据。

**连接管理**。通常主要指TCP的三次握手和四次挥手的安全连接确认过程以确保数据传输时的连接是可靠的，连接管理是TCP数据安全可靠的最基本保证。

**流量控制**。流量控制是在接收方持续接收到发送方的数据报进行处理时而缓冲数据区边界溢出而发送方又持续不断发送数据时无法接收而丢失数据包的情况，TCP能够利用此机制实现数据重发或根据接收方速度决定发送速度以保障数据的安全可靠。

**拥塞控制管理**。这个机制只要是应对数据大量传输而网络拥堵造成大量丢失数据包现象的优化控制实现，主要是通过发送方的慢启动方式试探性发送较少的数据检查网络状况后决定数据传输量的一种动态控制方法从而控制数据的传输优化TCP传输管理并确保数据更安全可靠。

3 TCP的安全连接管理

TCP通过较通俗三次握手和四次挥手实现安全连接管理从而确保数据的安全可靠传输。

三次握手指的是建立安全连接以实现数据传输。

以双方通电话为例。甲方拨通乙方电话询问:“在吗”？如果乙方在则回答:”我在，有什么事情吗”？甲方确认对方在:”我现在有些事情想和你仔细聊一下，如何有时间可以吧”？乙方第三次确认：“没问题，我有时间你说吧，我听着呢。”于是甲乙双方确认后开始进行通话。

四次挥手指的是通话完成后安全确认断开连接。

甲方：“我事情都说清楚了，如果你没有其他事情我想通话到此结束了”？乙方则说:”好的，我也没有什么事情了”。甲方再次确认：“那好，我现在就挂断电话”，乙方:”好的，再见”。于是安全结束通信。

4 Java 中URL 代表什么都包括哪些内容

Java中的URL是网络资源的描述，称为统一资源定位符，泛指互联网计算机通信中访问资源的一种虚拟映射。URL以字符串为描述内容构成一个远程主机的访问路径。它通常包括四部分内容，依次是网络协议、主机IP(域名)地址、访问端口号、目标应用资源，如：

<https://124.20.208:9090/edusys/index.html>

5 说说阻塞和非阻塞的区别

对于信息资源访问而言，客户端向远程服务器发出一个请求，由于远程服务器某些原因，未做出相应的回应，此时客户端会进行等待，直到服务器满足条件。对于服务器而言也是如此，在等待客户端请求时，如果没有任何客户端发出请求连接，服务器程序也能在未满足连接接入时进行等待直到有客户端发出连接并成功接入，如java.net组件包中的服务器套接字ServerSocket的accept方法就是一个阻塞方法，而ServerSocket也就是一个阻塞套接字组件。

非阻塞则与阻塞相反，客户端向远程服务器发出请求，如果条件未满足时通常会接到一个服务器返回信息告知请求情况，而客户端无需一直等待。

6 请描述一下java.net包下的一些网络编程核心组件

java.net 包主要提供基于网络的CS模式套接字编程API。提供基于常见的两种TCP和UDP协议的编程类库。基于安全连接的TCP最为核心的是服务器端程序构建基础的ServerSocket，它代表服务器端阻塞套接字应用，代表客户端程序的套接字组件Socket类。

基非连接的UDP用户数据报包套接字主要是DatagramSocket组件类以及代表映射数据报包的DatagramPacket组件类。

此包下还提供SocketAddress 以及其子类InetSocketAddress 作为主机地址的信息映射与套接字结合实现CS模式的网络应用程序。

7 完成基于TCP和UDP的上传文件到服务器的CS模式网络程序。

* Servlet相关

1 什么是Servlet

Servlet 是Java EE标准组件，Servlet程序运行在Servlet容器中(Tomcat, Jetty，Resin，GlassFish )在Java WEB MVC模式中被定位于处理客户端请求并做出响应的控制器组件。

Servlet 接口规范了控制器组件的相关抽象核心方法定义，主要包括init初始化方法，此方法接收一个ServletConfig接口实例为参数,此参数映射Servlet实例的初始化信息；service核心处理请求并做出响应的服务方法，此方法接收ServletRequest和ServletResponse接口为参数，分别映射客户端的请求信息和服务器的响应信息；destroy方法的是在当前Servelt实例从容器内销毁前执行资源释放的被容器最后调用的方法。init初始化方法必需在启动运行Servlet容器时Servlet实例被成功创建后执行调用，从而对Servlet实例进行相关初始化工作，此方法只会被调用一次，任何错误执行都将引发ServletException异常；service方法是Servlet被请求时执行处理核心请求业务的方法逻辑，此方法会在每次请求成功后都将被执行，它的执行此时取决于Servlet被请求多少次则执行调用多少次。destroy方法与init方法一样，在Servlet被销毁前只有容器执行一次从而释放相关资源。Servlet 一但被创建将一直存活于容器中，可以通过一个Servlet实例处理N个用户的N个请求及响应，相比传统的CGI具有占有资源少的较大优点。

2 GenericServlet 与 Servlet接口是什么关系

GenericServlet是实现Servlet接口和ServletConfig 接口的抽象类，它为开发者根据软件系统业务实现用户请求做出响应提供了基础。

GenericServlet 对Servlet中定义的init(ServletConfig) 进行了简单实现，并增加无参数的init初始化方法以供给GenericServlet子类进行重写扩展初始化工作；另外此类还增加了getServletContext方法并返回ServletContext上下文接口实例，实际上此方法内部是通过容器创建的ServletConfig获取由容器负责创建的唯一ServletContext接口对象引用。GenericServlet是一个未限定使用某个固定网络访问协议的Servlet，因此任何合法的标准的网络协议通常都可以应用在此GenericServlet控制器组件上。实际开发中基于http或https协议的较多，因此在实现基于http协议的Java WEB程序时往往更多的是定义继承自HttpServlet的设计方式。

3 HttpServlet 与GenericServlet有什么不同

HttpServlet 是 GenericServlet的子类，HttpServlet是特定与http协议的控制器组件在GenericServlet基础上增加很多特有的方法和成员属性定义。最核心的就是用以处理http方式请求和响应方式处理基于客户端不同方式的请求方法doGet和doPost方法，这个两个方法接收HttpServletRequest 和HttpServletResponse做为参数，HttpServletRequest和HttpServletResponse分别是ServletResponse和ServletResponse的子接口，用来映射Http方式的请求信息和http方式的响应信息，HttpServlet、HttpServletRequest、HttpServletResponse是javax.servlet.http组件包下的接口，是最核心的处理HTTP请求的API，通常实现基于HTTP协议的java WEB应用如果使用Servlet接口做为处理器则通常定义继承自HttpServlet的自定义Servlet实现并根据清清裤重写doGet或doPost方法来实现处理客户请求。

4 描述一下一个Servlet实例的生命周期是怎样的

一个Servlet实例的生命周期通常可以划分为四个阶段。第一阶段，新建阶段。在Servlet被请求时，如果此Servlet存在则使用其处理请求业务，如果不存在则容器负责创建此Servlet实例；第二阶段，初始化阶段。初始化阶段是在Servlet实例被唯一一次创建成功后被调用的方法，容器将根据是否传入参数决定是否传入一个ServletConfig实例给init方法从而实现对Servlet的初始化工作，此方法也只会别执行一次；第三阶段，服务阶段，也是Servlet最为核心处理请求业务的阶段，此方法将根据用户的请求次数被调用等同的次数并接受用户的请求信息对象和响应信息对象作为参数以提供用户的请求信息，其中包括请求头和请求行主要信息，对于HttpServlet而言，容器永远根据请求方式绝对调用对应的处理方法，如果post请求则调用doPost方法，get请求则调用doGet方法；第四阶段，销毁阶段。在容器结束运行而释放当前上下文时，此阶段会调用destroy方法从而释放所占用的资源，最终Servlet将被GC执行回收释放内存空间。此四个阶段中新建，初始化，销毁只会被执行一次，对应方法也会被调用一次，而服务器阶段的service方法将会根据请求次数执行对应相等次数的方法调用。

5 get和post请求方式各种有什么特点与区别

get和post是基于HTTP协议的常用两种请求处理方式，通常情况get方式将用户请求参数暴露式拼接在请求的url之后而post方式则将请求参数隐匿在请求主体内，因此get通常适合请求获取服务器资源而post通常适合发送隐秘敏感数据；get方式一般都会被浏览器默认进行缓存而post通常不使用缓存；从编码格式而言get基本支持唯一的url编码而post支持多种方式编码；get对传送的参数有长度限制而post没有限制。

从协议角度而言HTTP是TCP/IP范畴，基于HTTP协议的get和post方式也是标准的TCP链接，

无论是get还是post方式在发送请求中都会向服务器发送TCP数据包。而通常大多数浏览器系统会在get方式发送一个数据包，其中包括HTTP头信息和数据主体信息，post方式会发送两个数据包，首先想服务器请求发送HTTP头，如果服务器响应可以继续则再发送数据主体包。所以通常任务get方式效率高于post方式，但是实际情况并非如此。在网络相对比较畅通的情况下，发送一次和发送两次数据包的失效几乎是等同的。而在网络环境比较差的情况下，发送两次数据包更有助于TCP执行验证数据包的完整性，这也是post的一些优点，因此使用哪种方式要看具体情况，没有绝对的，而考虑到数据的安全性，如果是用户信息提交多数情况会选择使用post而数据加载通常更多时候可以选择get方式。

6 ServletContext 表示什么它有什么作用

ServletContext 接口是Servlet方式构建java web应用中对应用程序上下文的一个映射，通常可以将其看做是当前web 应用程序的上下文，它在jvm中就是一个内存区域也是最大的java web 作用域。一个基于Servlet的java web 应用只会有一个ServletContext对象，此对象在应用被发布时由Servlet容器自行创建，而此上下文在当前应用中被所有Servlet共享。此对象在程序中不能被创建和修改，只能被获取，通常任何一个Servlet实例可以通过getServletContext方法获取当前应用的上下文对象。而每次获取的上下文对象都是同一个内存空间的引用，ServletContext包含最为常用的setAttribute和getAttribute方法用来在上下文作用域中存储和获取必要的数据，严格而言此上下文作用域中只应该存储与当前上下文有关的共享信息，如网络访问在线人数等，不应该将任何某一个客户端数据存储在此上下文当中。

7 一个Servlet如何进行配置才能处理用户请求及做出响应

在定义一个类继承自HttpServlet或实现Servelt接口后，需要对Servelt进行定义配置和映射访问配置才能在部署应用是通过容器请求从而能够处理用户请求。在Servelt标准3.0之前一个Servlet必需通过在web部署描述文件(web.xml)中进行配置才能够被使用。在web.xml文件中通常使用<servlet>标记配置Servlet定义，其中子标记<servlet-name>定义目标Servlet的逻辑名称；<servlet-class>子标记定义目标Servlet的实际完全限定类名称；如果需要在创建Servlet后实现初始化配置还可以使用<init-param>子元素进行初始化参数配置。

Servlet如果被正确请求还需要使用<servlet-mapping>对Servlet进行请求映射配置，其中子元素<servlet-name>与上述的定义配置含义相同，必需将此逻辑名称与上面的<servlet-name>一一对应，而且必须使用<url-pattern>为如何请求目标Servlet配置映射的url，url必需以 / 作为前缀，<url-pattern>标记可以配置多个从而定义不同的url映射同一个Servlet。

在Servelt 3.0及之后还支持通过注解@WebServlet方式对Servlet进行配置。

8 RequestDispatcher接口的include和forward方法的作用是什么

RequestDispatcher 接口必需和某个用户请求对象ServletRequest进行关联，使用任何ServletRequest接口的实例或其子接口实例调用getRequestDispatcher 方法会返回一个与当前请求对象相关的RequestDispatcher接口。此接口是一个请求处理发报机，作用是通过此接口中的两个方法include和forward实现请求某个网络可访问资源的嵌入嵌入和转发工作，这些网络资源通常是可用的Servlet，JSP以及html文件，include用来包含目标资源而forward用来将请求转发到目标资源，在调动这两个方法时需要传递请求和响应对象，所以这两个方法最终的效果无论是调用include还是forward方法都是是同一个请求操作。而在获取RequestDispatcher接口实例时则应提供欲包含或欲转发的目标资源url。

9 请求转发和请求重定向有什么不同，在什么情境下使用哪种方式

实现请求转发和重定向是两种不同方式跳转资源页面的机制。转发是服务器端行为，是将用户的当前请求转发到目标资源，这种方式与当前请求紧密相连，目标资源与当前请求同属于一个请求作用域，也就是同一个请求，在目标资源页面可以获取当前请求域中以及任何用户提交的请求参数，因此在同一个请求域下最终跳转的资源url是没有变化的，仍然是转发前的最初用户请求的url。

重定向跳转不是服务器行为而是客户端行为，含义就是服务器告诉客户端本次请求结束，请重新请求新的资源，这个新资源就是用户重新发出另外一个新请求的目标URL访问，所以重定向是两个客户端请求，同时也是存在两个不同的请求作用域，这是与转发最大的差异，第一个请求中的请求参数以及请求作用域中任何存储的数据都无法在重定向新的请求中获取，而最后重定向后用户最终url和之前的请求url是完全不同的。

通常转发机制更适合在最初的请求域中存储的数据在新资源页能够在请求域中获取，已达到共享一个请求作用域的目的；重定向通常适合不共享用户初次请求任何与请求相关的数据，包括数据库的访问过程，另外请求重定向能够方便避免最后页面刷新所带来的重复提交信息的问题，比如登录成功后相对比较适合使用请求重定向进行处理。

10 HTTP1.0、HTTP1.1 与 HTTP2.0 有哪些不同

HTTP1.0 是继HTTP0.9(基本废弃)只支持GET请求方式及只识别服务器响应的HTML格式的升级版本。HTTP1.0新增了POST和HEAD数据头命令，每个TCP连接只能发送唯一的一个请求，发送数据完成后就关闭连接，如果需要发出新的请求则必须重新建立一个连接。  
HTTP1.0头信息采用ASCII 编码格式，数据主体可采用任何格式，服务器在响应客户端时使用Content-Type告知客户端是什么内容。

HTTP1.1 新增了DELETE 、PUT、OPTIONS 和PATCH方法，但是多数浏览器还不支持。HTTP1.1还新增加了持续长连接处理机制，此机制默认在一个连接建立后默认是不关闭的，这与HTTP1.0有本质区别，因此可以用来重复发出请求，与HTTP1.0相比改善提高了客户端的访问效率。

HTTP2.0是Google公司研发的遵循SPDY(speedy简写，一种由google研发的用以升级增强TCP的协议标准，含义为” 更快”)协议的HTTP协议升级版本。HTTP2.0一改以前1.x版本  
头信息以及主体数据格式而采用二进制方式进行处理，被称为头信息帧和数据帧使得更具有健壮性；HTTP2.0还采用TCP连接复用可以在一个连接中客户端同时发送并行的多个请求和回应而且顺序可以自由控制，这样处理相比HTTP1.x的队列头的阻塞现象频发；HTTP2.0同时对信息头启用了压缩机制，从而使得请求头内容更加短小并在客户端与服务器端同时维护一个头部信息表，此表中主要存储信息索引，通过索引来提高传输效率；另外HTTP还弥补了HTTP1.x不支持服务器主动push的操作，从而能够使得服务器不经过客户端请求便可以单一放主动向客户端发送资源实现信息推送。

因此HTTP1.x存在以下缺点。

1.0一次只允许在一个TCP连接上发起唯一的一个请求；HTTP1.1虽然能够处理部分请求并发，但是仍然存在队列头阻塞频发的问题；HTTP1.x只支持单向请求，只能由客户端主动发起而服务器无法主动push；HTTP1.x请求报文与响应报文信息冗余量过大使得传输负担重；HTTP1.x无论是信息头和主体数据都不采用压缩因此数据传输比较慢。

而HTTP2.0则具有以下优势从而弥补了HTTP1.x的诸多弊端。

二级制编码处理更加灵活健壮，可任意分割数据；

多路复用解决头部拥塞的现象使得数据加载更快；

支持服务器主动push实现消息推送；

安全性做了近一步加强，通过黑名单机制限制使用很多种不安全的加密算法。

11 HttpSession 表示什么，如何获取此对象

HttpSession 是javax.servlet.http 包下的接口，用来表示java Web 应用中的用户HTTP会话对象。HttpSession接口实例在服务器中整个生命周期映射一次用户与服务器的交互过程，此对象不能在程序中由用户显示创建，通常由任何请求对象调用getSession获取此对象，getSession方法提供boolean类型参数用以表示如何创建会话对象，如果传入false则获取已经创建的用户会话对象，如果未创建则返回null，如果传入true值或未设置参数则在会话存在时返回对象引用，否则创建新的会话对象，HttpSession 经常用来保存只和用户相关联的信息。

12 HttpSession会话对象如何被维护在服务器上的生存周期，以及服务器如何区分不同的用户会话对象？

HttpSession 接口中setMaxInactiveInterval 方法用来设置用户会话的过期时间，参数是表示秒的int类型，在java web应用中还可以通过web.xml 文件以及在服务器中进行HttpSession的会话超时设置。三种方式优先级为setMaxInactiveInterval方法，web.xml中<session-timeout>设置，单位为分钟，服务器中web.xml中<session-timeout>设置，单位为分钟，默认为30分钟，可根据实际情况进行调整。服务器中设置适合所有部署到当前服务器中的任何应用；应用程序中web.xml适合所有当前与应用相关的用户会话；setMaxInactiveInterval适合当前用户会话。在会话超过最最大生存时间而未与服务器交互时，会话对象将自动由服务器负责清除。

HttpSession接口会话实例在服务器中用唯一的id值记录用户，服务器用此id区分不同的用户会话对象，程序中可通过getId方法返回会话的唯一标识。

13 java web 应用中如何区分用户的会话是新的会话或者是旧的会话对象？

HttpSession 接口定义规范方法isNew方法用来返回boolean值以检验当前会话是新的会话还是旧的会话对象，如果此方法返回true则是新建会话，否则返回false则为旧的用户会话对象。

14 ServletRequest 的方法getAttribute和getParameter方法作用是什么

ServletRequest的getAttribute是在当前请求作用域中获取指定名称的属性数据，此方法入参为一个字符串参数，标识试图从当前请求域中获取此名称对象，如果存在则返回此对象引用，返回结果为Object类型；getParameter方法是从当前请求对象获取用户的请求参数，指定字符串参数为请求参数的名称，如果存在则返回此名称参数的字符串值，如果不存在此请求参数则返回null, getAttribute可获取作用域中存在的任何类型属性数据而getParameter只能返回存在的请求参数字符串值。

15 java web 应用中有哪些方式可以实现会话跟踪

由于HTTP会话是无状态会话，客户端与服务器的任何请求交互信息在服务器处理完请求后不会保留任何信息，通常在java web中可使用会话授权，url重写传递会话id，隐藏表单域以及使用Cookie实现java web 会话的跟踪实现。

16 Cookie是什么，在java web应用中有什么用处

Cookie 是javax.servlet.http包下的类，用来映射服务器存储客户端机器上的文本信息，在服务器创建Cookie对象后，设置Cookie对象的存储文本信息，使用请求HttpServletResponse 对象的addCookie方法Cookie对象发送至客户端浏览器并将Cookie中的文本信息存储在客户端硬盘上，在客户端再次访问服务器时会随请求将Cookie信息带回服务器，服务器可通过相关方法获取Cookie信息从而实现对客户端的识别，Cookie若想取得预期效果则必需在客户端浏览器的支持，若客户端浏览器设置不保留Cookie到本地硬盘，则在客户端退出浏览器后Cookie将不存在。Cookie经常在电子商务应用中向客户推送商品信息以及处理用户自动保存登录密码等实现。

17 哪些原因会导致用户重复提交请求，有哪些解决办法

常见的客户端浏览器点击二次提交按钮，点击浏览器刷新按钮，使用浏览器后退按钮导致重复操作，浏览器重复HTTP请求url。

常见的一些解决办法包括使用JS实现对提交按钮的一次有效点击后禁用。如果不涉及从请求域中放置响应给用户的数据，可采用服务器端的重定向代替请求转发避免刷新时导致重复用户请求。

采用会话域存储变量标准或使用表单隐藏域传递标准令牌进行检验是否是重复提交请求。数据库访问也可以采用锁定策略，但是此方式效率比较差，在使用其他方式可避免时尽量少应用此方式。

如果是服务器端进行处理尽可能采用AOP方式进行设计，可提供代码的重用性。

18 如何为java web 应用配置通用类型错误的资源页面导航

配置java web 用户访问发生错误页面定位可根据具体情况进行灵活配置统一错误页面导航，如果应用中使用了JSP技术，则在jsp中可以使用page指令的属性errorPage设定错误页面url，前提是错误目标错误导航页面如何想使用错误对象输出错误信息则必需指定JSP错误页面的page指令的isErrorPage 属性值为true。

如果不考虑JSP页面而只在某些类型错误发生时给用户一个更好的体验并在整个web应用中具有通用性，可以在web.xml 部署描述文件中使用标记<error-page>进行设置，其中使用<error-code>或者<exception-type>设置错误代码或错误类型；使用<location>设置错误导航页面的URL，值得注意的是<error-code>和<exception-type>不能同时使用。

19 java web 应用中的四个作用域都是什么，它们有什么关系？

java web四个作用域通常指当前页作用域，在jsp中被称为page域，对应的jsp隐式对象名称是pageContext; 当前请求域，也就是request域，对应jsp隐式对象request；web会话域，也就是HttpSession域，对应jsp隐式对象session；应用程序上下文域，对应jsp隐式对象application，此作用域与ServletContext 是一个作用域对象。

所有四个作用域的作用都是为了在当前域范围内存储任何可用的java对象，以实现在可访问的作用域内传递信息。page域中的对象只在当前页面有效，出此页面访问就无法被访问；request域只存储可供同一个请求范围内的请求信息，如果当前请求参数及被同一个请求域共享的数据对象；request域中数据对服务器的转发行为都是适用，如果进行重定向，则无法访问请求域中放置的任何数据对象；session 域通常只存储与当前用户相关的信息，如用户对象；应用上下文域存储与整个应用相关联的数据对象，也是最大的作用域，四个作用域逻辑上是附属包含关系，上下文域中可包含N个会话域；一个会话域中可包含N个用户请求；一个请求中可包含N个页面。但是四个作用域不是相同的，也就是小的作用域中存储的数据对象不会在大的作用域中能够被访问获取。

20 描述一下service方法和doGet及doPost方法的关系

service 方法是Servlet接口中用来处理用户请求的核心服务方法，是实现处理用户请求并作出响应的逻辑实现核心，此方法在目标Servlet被成功请求后有Servlet服务器协调使用当前Servlet对象执行调用从而处理用户请求并作出最终响应，此方法接收ServletRequest对象和ServletResponse对象作为入参。

doGet和doPost 不是Servlet接口中定义的方法，此方法是HttpServlet 抽象类中后增加的方法，其目的就是能够对基于HTTP请求的get和post方式请求做出针对性处理，通常用户在基础此HttpServlet抽象类时可以对doGet或doPost方法之一或全部进行重新以实现请求处理逻辑，doGet和doPost方法接收的入参是HttpServletRequest和HttpServletResponse，因此只适合基于HTTP的请求方式；由于Servlet的规范service才是能够被最终执行处理的核心业务处理服务方法，所以在HttpServlet抽象类中对父接口的service方法进行了重写，其核心就是通过请求对象获取用户实际使用的是哪种HTTP请求方式从而决定调用doGet或是doPost方法，这就是doGet或doPost方法最终会被执行调用的原因所在，如果不需要考虑用户使用哪种HTTP请求方式，用户在继承HttpServlet抽象类实现请求页处理时，只要重写service方法即可实现请求处理。

21 说说你对javax.servlet.Filter 接口的理解

Filter 接口是在Servlet 2.3版本中新增的规范，主要目的就是通过此接口中的核心doFilter实现对应用中用户请求前的通用性访问过滤功能，正如此接口名称描述一样，它就像一个类似于与生活中的矿泉水生产中的水过滤一样，在出厂前进行过滤以达到固定的某些目的。

Filter 的定义规范就是AOP模式的抽象表示，AOP是面向切面编程的设计思想，其含义是将主流业务和非主流业务进行拆分，以避免主流业务实现与非主流业务实现代码的耦合性，并提供高重用性，AOP实现应遵循插拔式规则，在使用时安装，不使用时卸载，而不会更改主流业务任何实现。

因此在java web应用中，任何Filter的实现都是以配置的形式设置此过滤器作用在哪些用户请求的url上，通常java web中可以在web.xml 配置文件中使用<filter>与<filter-mapping>对过滤器进行定义初始化和过滤映射配置，在Servlet3.0 后也支持使用注解@WebFilter进行配置。

其实Filter几乎和Servlet一样，具有类似的生命周期，其中的init 方法和destroy 方法分别初始化过滤器和销毁前释放资源，而doFilter实现过滤业务。

最常见的应用如用户身份验证；记录的审核；数据加密或压缩过滤；用户访问时触发资源检验权限等AOP实现。

22 什么是过滤器链条

过滤器链条是java web用户请中通过多个filter过滤器配置实现的AOP拦截机制，此机制实现用户到的目标请求前被多个过滤器进行拦截而完成检查验证的过程，如用户下载文件时需要验证是否已经等了是否具有足够的下载积分等操作。在java web中web.xml 部署描述文件如果定义并配置了多个filter，默认情况下在对拦截的url请求实现请求过滤，这些过滤器按照先后顺序执行对请求的过滤，任何过滤器都可以终止用户的当前请求，如果在调用doFilter方法时，执行过滤器链条的doFilter方法调用，则请求继续交给下一个过滤器进行过滤，以此类推就形成了过滤器的连续执行，这就是过滤器链条实现。

23 如何实现自定义的过滤器

当在需要使用AOP模式处理用户的请求时，开发者可以自定义类实现javax.servlet. Filter接口并重写其中的抽象方法，利用过滤器实现类中的doFilter重写方法实现对某个用户请求的过滤来完成AOP业务。

24 web.xml 部署描述文件中的<Listener>标记用来配置什么，有什么用处？

web.xml 部署描述文件中的<Listener>标记用来配置实现一个ServletContextListener接口，ServletContextListener接口是java 标准的事件监听接口EventListener的子接口。此接口规范对web应用上下文启动进行某些必要的初始化工作，此接口实现必需在任何web.xml文件中配置的servlet或filter之前进行初始化，也就是在部署启动web应用时，此监听接口将被实例化并调用其中的规范方法contextInitialized，此方法的作用是实现对上下文进行初始化工作，此方法接收一个ServletContextEvent 类实例作为入参，此参数是一个web应用上下文的事件对象，利用此对象的getServletContext可以获取被实例化的web上下文对象引用，从而利用获取的上下文对象实现一些特殊的初始化工作，典型的就是在使用springframework IOC框架作为web应用bean工厂管理bean对象时通过此<Listener>配置上下文监听接口从而对springframework的上下文配置文件进行操作以实例化被托管的bean对象并放置关联在上下文当中。

ServletContextListener还规范了contextDestroyed 方法，此方法的作用是在当前web上下文进行销毁时做一些必要的工作，如通知任何基于此web上下文进行工作的Servlet及Filter，告诉上下文即将被销毁，如果此上下文存在任何必要的资源释放在应在此方法中编辑代码实现资源的释放操作。

25 java 虚拟机什么情况下会出现StackOverflowError(堆栈溢出错误)

对于java应用功能的实现是通过方法调用执行而完成，在java中一个方法在执行调用过程中会对应创建一个虚拟机栈，称之为线程栈帧，线程栈帧是私有的，此栈帧中只存储与当前线程有关系的各种必要信息，包括局部变量表，操作数栈，动态链接，方法出口等信息。如果在线程请求数超过了当前JVM所设置的最多限度时，JVM就会抛出StackOverflowError错误，并导致JVM终止。

在设置了JVM可以进行动态扩展的情形下，如果内存已经临界极限使用上限，JVM也尝试了申请新内存空间进行扩展空间，但是无法达到目的，则JVM就会抛出OutOfMemory (内存不足)，通常如果应用中启动过多的线程则发生此错误。在配置JVM时可使用参数-Xss调整限制JVM栈的大小。

26 请描述一下JVM的内存模型

JVM主要由以下部分构成内存模型。

程序计数器：当前线程所执行的字节码文件的行号指示器，作用是记录当前正在执行的虚拟机字

节指令地址功能，对于计数器而言是每个线程私有的。

JVM栈：栈的主要作用是存放基本数据类型、对象的引用、方法出口等信息，栈对于java线程也是私有，线程间不能共享。

JVM堆：JVM中内存最大的一块存储空间，它是所有Java对象实例、数组都存放在内存区域，可以称之为实体地址空间区，java垃圾回收目标指向的地方就是这个内存堆，JVM堆对于任何线程都是共享的。

方法区：方法区主要用来存储已经被加载的字节码信息、静态变量、常量、即时编译器编译后的代码数据等，方法区被称为JVM永久带，方法区对于各线程也是共享。

Native(本地)方法栈：方法栈和JVM栈类似，它是Native方法的使用区域，通常和底层实现有关，对于线程也是私有的。

27 请说说你了解的一些JVM配置参数以及都是什么含义

-Xmx ：设置JVM堆最大使用上限。

-Xms： 设置JVM初始堆大小。

-Xmn：设置GC年轻代大小。

-Xss： 设置单个线程的堆栈大小。

-XX:MaxPermSize：设置持久代大小。

-XX:NewRatio=?: 设置年轻代与年老代的比值（？通常为一个整数）。

-XX:SurvivorRatio=?：设置年轻代的Eden区与Survivor区的大小比值。

-XX:MaxTenuringThreshold=?：设置垃圾最大年龄，如果?为0则年轻代对象不经过Survivor区，直接进入年老代

28 JVM 中的GC指的是什么

GC 是Garbage Collection的缩写，意思是垃圾清理，GC是JVM中低级别运行的服务进程，按某些规则用来对内存中不在被程序所使用的数据对象进行释放占用空间的功能实现，因此Java开发人员不必想C或C++语言中需要手动释放内存空间。

29 被GC释放的空间包括哪些，哪些不需要释放，GC什么时候进行垃圾清理

从JVM内存结构来讲，存储在Java堆内存中以及方法区上的对象此适用于GC进行清理，而JVM栈，本地方法栈和程序计数器是无法被GC执行垃圾清理的。

某个堆中对象被发现已不存在有效引用(此对象不在具有任何原生句柄或副本句柄)时，此对象将被GC；

当一个类所属的对象都被回收后，此类也将被GC；

如果某个类的ClassLoader已经被GC则被通过此ClassLoader加载的类也将被GC;

某个类所对应的放射类Class对象如果发现已经不存任何引用，则此类将被GC;

当在JVM堆已满的情况下将执行GC操作；

在代码编辑中程序员手动调用System类的gc()方法后，JVM 垃圾回收器会尝试执行GC，但是这取决于GC是否能够获取空闲CPU的使用权，因为GC低级别线程，通常必需在CPU空闲时才能获取使用权。

30 垃圾回收器如何判断某个对象满足被回收的条件从而执行对此对象所占内存空间的释放？

GC通过以下一些算法来判断某个对象是否符合被回收释放的条件。

1. 对象引用计数法

主要是通过对象上增加引用计数器，这个计数器维护着对象被引用的一个整值，某对象被引用一次则此值增加1，如果引用失效则减1，如果某对象为0则表示此对象已不存在有效引用，将被回收内存空间，此算法实际应用极少被采用，因为此中算法无法对特殊的循环引用进行区分，如A引用B，B又引用A将无法被GC使用这种算法识别。

1. 可达性算法-搜索法

垃圾回收器通过名称为GC roots的对象作为起点开始向下依次搜索，从而形成一个搜索路径，这个路径被叫做对象引用链条。当某个对象在roots没有任何引用链相连时，说明此对象已经不存在任何引用是不可用对象，GC将对此对象进行回收从而释放空间；对于root通常包含以下方面：字节码Class对象；有效的Thread线程；本地栈stack local；java 本地接口和java全局接口；同步监视器等；

31 Java8与以前版本JVM堆的划分有什么不同？

在Java8版本以前通常JVM堆被划分为young generation(年轻代)；old generation(老年代)和

permanent generation(永久区); 年轻代又被化分为Eden和Survivor。

在Java8版本时对JVM堆进行了调整，最为主要的就是取消堆上的permanent generation而使用

Matespace VM 进行代替，MVM则被设置在Native Method中，MVM可以进行自动增长从而避免了Java8本本以前的OOME（OutOfMemoryError）错误，而且MVM也可以通过参数MaxMetaspaceSize进行设置以避免浪费本地内存。

32 JVM 堆年轻代young划分为Eden和Survivor而且Survivor被划分为2个区，有什么作用

JVM young区之所以被划分为Eden和Survivor，主要和GC有关，如果没有Survivor区则Eden区每进行一次清理时，未被清理而存活下来的对象将被送到old区老年代。如此老年代将很快被填满，而导致老年代的内存空间要远大于新生代，这样在进行一次Full GC消耗的时间要比年轻代Minor GC所用的时间长得多,因此将young分为Eden和Survivor，Survivor则充当一个缓冲地带。

Survivor区存在意义在于能够减少被送到老年代的对象，在进行Minor GC时，幸存对象会暂存于Survivor区，从而减少old老年代Full GC的发生，只有经历16次Minor GC还在新生代中存活的对象，才会被送到老年代。

young被划分为两个Survivor区的好处是能够解决磁盘碎片问题，在刚刚被新建的对象在Eden中，经历一次Minor GC，Eden中的存活对象将会被移送到第一块survivor S0，Eden区将被清空；当Eden区再被填满时，再触发一次Minor GC时，Eden和S0中的存活对象又将复制被移送到第二块survivor S1，GC的这种复制算法保证了S1中来自S0和Eden两部分的存活对象占用连续的内存空间，避免了碎片现象的频繁发生。

33 Java 垃圾回收器有哪些算法

1. 标记清除法

标记清除法是GC最基本的算法，此算法是对被回收的对象做标记计数，然后进行扫描，有回收标记的则进行回收，主要有两个步骤：做标记和执行清除。此算法效率不高，很少被使用，而且容易产生内存碎片，如果有大对象(如较长的字符串)就需要连续的内存空间，这就需要进行碎片整理。

1. 复制算法

通常复制算法适用于年轻代GC操作，新生代分为三部分，Eden区和Survivor S0以及S1区 ，通常JVM把Eden区和Survivor区的比例调为8:1，保证其中一块Survivor区是基本空闲的，这样在执行GC的时候，将不需要进行回收的对象放在空闲的Survivor区，然后将Eden区和第一块Survivor区进行完全清理，如果第二块Survivor区的空间不够大时JVM将暂时借持久代的内存暂存对象。

1. 标记整理算法

标记整理算法也叫压缩发此算法和和标记清除算法前面执行算法非常类似，只是在标记不需要进行回收的对象后，将标记过的对象移动到一起，使得内存能够连续避免磁盘碎片的产生，如此只要将未标记的对象进行清理可以了，标记整理算法主要用于持久代回收。

34 你知道哪些垃圾回收器

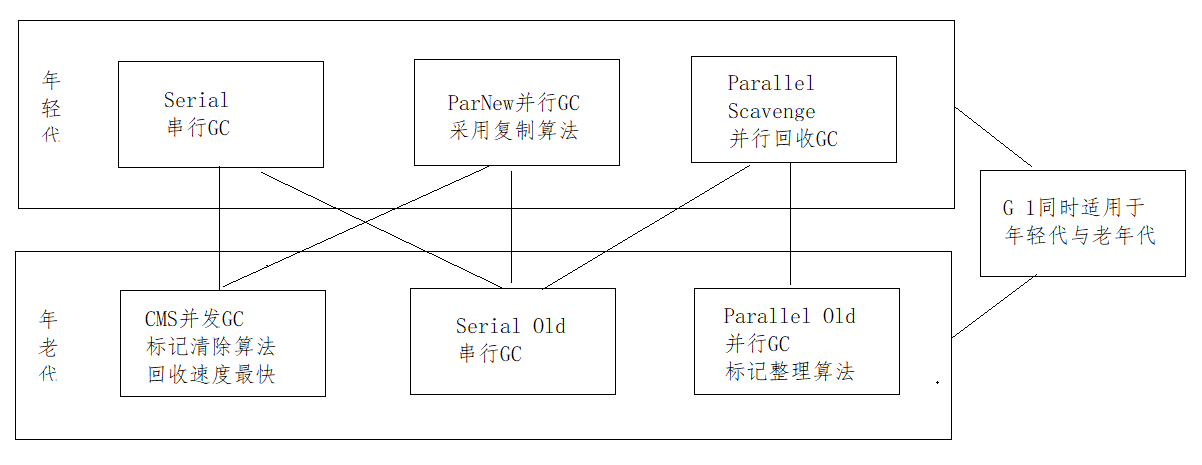
大多数GC可划分为年轻代GC和老年代GC两种。

主要的新生代GC有：串行化GC、并行GC、并行回收GC；

主要的老年代GC有：串行GC、并行GC、CMS GC;

java8后增加了G1 ，这个GC是比较特殊的它能够用于年轻代同时也适用于老年代垃圾回收。

可以记住以下图形并在面试被问及此问题时手绘给面试官



35 说说Java Minor GC、Major GC和Full GC有什么不同

Minor GC是新生代GC，此GC操作发生在年轻代上的垃圾回收，在新生代无法为新生对象分配内存空间的时就会发生Minor GC 。因为新生代中大多数对象的生命周期都很短，所以发生Minor GC的频率很高，但是它的回收速度很快，所以此GC算法以速度为优先原则。

Major GC清理Tenured区，它只适用于老年代回收操作，当出现Major GC通常会出现至少一次Minor GC，因为老年代中对象都来自于年轻代被清楚的暂存对象，在未发生Minor GC时不会发生Major GC操作。

Full GC是针对整个新生代、老生代、元空间（java8及以上版本用于取替perm gen）的全局范围的GC。此GC操作通常是在元空间已满(java8及以上版本)操作，java8之前为永久区已满, 由于在java8及以后此元空间可以动态增加，所以Full GC在java8及更高版本中执行极少，由于是Full GC需要清理全部可能被回收的对象所以开销比较大。

36 JVM 一次完整的GC操作流程大概是怎样的？

JVM 堆大体划分为年轻代和年老代，年轻代又被划分为Eden和S0与S1；GC大体执行流程是：

在年轻代Eden 区的空间满的时候，JVM将会触发一次 Minor GC，以收集新生代的垃圾，并将存活下来的对象移送到 Survivor区；

如果是大对象，如较长的字符串将直接被移送到年老区，以达到快速彻底回收大对象而释放内存空间；

如果对象在Eden区出生(新建对象)并经过首次Minor GC后还存活，而且能够被Survivor容纳的话则为此对象年龄设为1，在每经过一次Minor GC，年龄增加1，当年龄超过一定限制，通常值为15，则此对象被晋升到老年态。JVM将此对象移送到年老区；

老年区空间如果被填充满而无法容纳更多的对象时，Minor GC 之后通常就会进行Full GC，Full GC 清理整个内存堆 – 包括年轻代和年老代；

Major GC 操作发生在老年代，清理老年区空间，此GC的发生通常会在至少1或N次Minor GC，Major GC 操作比Minor GC慢很多。

37 CMS垃圾收集器和G1垃圾收集有哪些主要区别？

CMS收集器是老年代并行的收集器，主要是配合新生代的Seria串行l和ParNew并行收集器一起使用；

G1收集器收集同时适用于是老年代和新生代，不需要结合其他收集器使用；

CMS收集器以停顿时间最短，这是此收集器的目标；

G1收集器可预测垃圾回收的停顿时间而CMS则不能；

CMS收集器是使用“标记-清除”算法进行的垃圾回收，容易产生内存碎片；

G1收集器使用的是“标记-整理”算法，进行了空间整合，降低了内存空间碎片的产生。

38 请说明java 字节码（类）文件被加载的过程

一个java字节码文件从开始被加载使用到卸载经历以下几个阶段。

加载阶段。类的加载最初通过一个类的完全限定名来获取类的二进制字节流；然后将此字节流内容转化成方法区的运行时数据结构；最后在java堆中建立一个代表这个类的Class字节码对象，作为方法区类对象相关数据的访问入口，此阶段主要通过类加载ClassLoader完成。

链接阶段。链接阶段主要工作是执行验证，准备和解析操作。

执行对字节码的验证，以确保被加载类的正确性；准备操作，为字节码的静态变量分配内存空间，并初始化默认值；解析操作，主要是把类中的符号引用转换为直接引用；

初始化阶段。Class对象被创建后则对类进行初始化操作，主要是访问某个类或接口的静态变量，或者对该静态变量赋值，调用类的静态方法 ，如这个类存在直接父类，并且这个类还没有被初始化则进行初始化直接的父类。Class 通常初始化顺序为：静态变量、静态块，成员变量， 构造器；

应用阶段。在初始化正确完成后，此Class对象贮存在JVM堆中以供给程序使用；

卸载阶段。当一个Class不在使用时，将被从JVM中卸载。

39 Java中你知道哪些类加载器ClassLoader

Java 中通常可以将类加载器分为主要的四种，分别是：

BootstrapClassLoader启动类加载器。它主要负责加载Java rt.jar中的所有的class 。   
ExtensionClassLoader扩展类加载器。它主要负责加载java平台中扩展功能的一些jar文件。   
AppClassLoader系统类加载器。它负责加载classpath中指定的jar包以及目录中的class。   
CustomClassLoader用户自定义加载器。 此类加载器隶属于当前应用程序系统，由开发者自定义，继承自ClassLoader，只有在特殊情况下才会使用，通常无需定义这个实现。

* 设计模式相关

1 如何理解设计模式和技术

设计模式完全不能和某个技术混淆，设计模式是一种生活实践中总结出来的用于特定某些场景下的一种设计思想，它所体现的是设计重用，它与各个行业领域没有直接关系，可能适用于不同的领域，可以说设计模式是跨领域和行业的。而技术是某种行业中为解决特种问题所创造出来的一种可以被称为技术工具的应用，技术更针对性解决问题，并且某种技术都可以不同程度在实现设计时使用到某些设计模式来构建解决方案，所以设计模式不应受任何技术支配，而某种技术都可以根据情况使用某种设计模式。同时设计模式极大程度影响某个领域技术的发展和变革，而多个领域的不同技术对于设计模式也存在激励作用，促使更多设计模式的产生。

2 面向对象设计原则是怎样的

通常面向对象设计原则应遵循以下原则。

1)、开闭原则：对应用扩展功能实现开放，而扩展功能时不应修改原设计即对修改关闭；

2)、 里氏代换原则：当派生类可以替换基类，软件单元的功能不受到影响时，基类才能真正被复用，而派生类也能够在基类的基础上增加新的行为功能；

3)、依赖倒置原则：开闭原则的基础,对接口进行编程设计,依赖于抽象而不依赖于具体，即所谓的面向接口进行设计；

4)、接口隔离原则：不同功能模块尽可能减少接口的继承，使用多个隔离的接口降低设计的耦合性；

5)、迪米特法则（最少知道原则）：一个实体应尽可能最少量与其他实体之间发生相互作用，使得系统功能模块相对独立，这也是降低耦合性依赖性的设计原则；

6)、合成复用原则：在最少知道原则基础上尽量使用合成/聚合的方式，少使用继承。因为继承实际上破坏了类的封装性产生了高耦合性，超类的方法可能会被子类修改。

3 Java 语言中设计模式可以分为几类都是什么

Java 语言中关于设计模式可以总体分为三大类。

建造模式或创建模式。是最为基本的常用设计模式，作为普通程序员也应了解掌握并运用，其中包括，工厂模式，创建(构造者)者模式，原型模式和单例模式，工厂模式又细化为工厂方法模式和抽象工厂模式共五种。

结构模式。包括代理模式，适配器模式，装饰模式，桥接模式，享元模式，组合模式和外观模式共七种。

行为模式。主要包括责任链模式，观察者模式，策略模式，命令模式，备忘录模式，迭代器模式等十一种。

4 Java 核心面向对象的设计思想是什么

Java 核心面向对象的设计思想是面向抽象面向接口进行设计。

5 什么是单例模式，它的主要用途是什么

单例模式实例是建造模式的一种，单例模式是指一个应用程序中只有一个某类型的实例，主要用途是在一个应用程序中只需要一个这样的实例即可完成相关业务操作，单例模式最主要的作用在于其存储空间可以被当前应用程序中共享，由于单例模式被整个应用程序共享，所以应注意不同用户线程的数据安全问题。

6 如何创建一个单例模式实例会更加安全

通常创建更安全的单例模式可以通过私有的内部类来实现比较安全。

7 简要说明一下工厂模式的实现与传统的对象创建方式有什么不同

传统的对象创建方式是在哪个类中里需要某对象实例时则在此类中创建此对象，这样出现的问题就是对象的创建比较分散，不易于管理，需求者必须自己需要时负责创建，对于当前类而言分散了业务精力，就如同需要吃饭时，都是自己来做饭一样。工厂模式创建对象解决了对象分散创建的弊端，而将对象的创建集中在一个工厂类中统一创建管理，当某个类需要某个对象时，可以从工厂中获取，而需求类没有任何创建对象的额外负担，如同需要吃饭时，点外卖一样。此种方式通常被称为get方式获取，工厂方式的弊端是增加了额外的工厂类及维护对象创建的方法定义和实现，典型的工厂设计模式如应用比较广泛的springframework IOC 外部工厂容器框架，此框架唯一与应用中自定义工厂不同是springframework IOC容器是个外部工厂设计实现。

8 说明一下java中链条式语法与哪种设计模式有关

链条式语法是某个对象调用相关方法后仍然返回本对象的实例引用，从而可以形成联系的方法调用，这种链条式语法主要是简化应用程序代码，使用范围是这写能够被联系调用的方法必需返回此对象的引用，通常就是this。其弊端是非链条末端无法返回本对象引用外的任何数据，这种链条式语法是创建模式中的构造者模式，构造者指的就是当前对象引用本身，此设计模式是局部功能设计的一种应用，无法应用在整体应用设计中。

9 Java中的静态代理和动态代理有什么不同

通常代理模式的实现总体包含三个角色。

抽象接口。声明真实对象和代理对象的共同接口；

代理角色 (代理实现者)。其中主要包含了被代理的对象，以实现目标对象方法的调用；

真实角色 (被代理对象)。使用者最终要引用的目标对象。

对于静态代理而言，具有如下特点。

代理角色和真实角色拥有共同的抽象接口；

一个真实的被代理对象对应一个代理实现类；

通常被代理对象必需存在于代理类中被固定不变；

设计实现简单清晰，但是会产生更多的代理实现类，当被代理对象无限增加时会造成类爆炸，这是静态代理设计实现的最多弊端。

动态代理则具有以下特点。

代理类的代理实现代码逻辑被固定下变,不会因为被代理对象的增加和业务的逐渐增长而改变;

被代理对象在应用程序运行时才被确定，形成动态模式，其原因是动态代理是一种接口方式的代理

设计；

动态代理实现AOP(面向切面)编程更加灵活,这是静态代理无法实现的 ；

高度解耦，可以实现应用不同逻辑层之间的解耦。

值得注意的是，虽然代理模式是AOP的实现，但是通常与AOP拦截器或过滤器不同是代理最终的执行应该得以让用户最终能够获取目标被代理对象方法的执行，而拦截器或过滤器有可能会终止用户的目标方法访问。

10 代理模式在哪些场景中比较适用？

代理模式如同实际生活中的工厂生产的产品，这些产品最终是为供给需要的每个人使用，而这些产品的销售由于某些原因并不适合让消费者直接到工厂进行购买，如此就是出现了产品代理商，最终消费者在代理商购买使用的产品与工厂所生产的产品一样的功能，但是代理商可以对代理的产品进行额外的二次包装，但是并不改变实际被产品的原功能。代理模式的应用就是当某个类需要调用另外一个类的功能时，并不是直接使用目标对象调用其方法，而是通过一个代理类实现间接获取目标类的方法调用，这样对于目标类就对使用者规避了与相互作用的问题，但是对于使用者而言与直接调用目标类的功能是完全相同的，这就要求代理实现类最终应具有被代理对象的引用从而保证调用实际目标类的方法，但是在调用前与后代理类可以增加额外功能修饰。

11 java 是如何实现动态代理的？

Java 动态代理实现通常由InvocationHandler 接口和Proxy类组成，它们都是java.lang. reflect反射包中的组件。

InvocationHandler接口是实现Java动态代理的规范接口，自定义实现的动态代理类必需实现此接口。此接口是一个函数式接口，定义了唯一实现动态代理的Object invoke(Object obj,Method method, Object[ ] args) 抽象方法，此方法必需在其动态代理子类中实现代理功能。

方法第一个参数是动态代理实现的实例，第二个参数是目标被代理对象的父接口方法，第三个参数是传递给目标方法的参数，如果目标方法没有参数，则此处应为null，此方法声明Throwable类型错误，可以处理任何异常。

Proxy 类提供用来创建动态代理类和实例的静态方法，此类还是由这些方法创建的所有动态代理类的基类。Proxy 类最常用的就是newProxyInstance 静态方法，用来根据给定的参数创建返回一个代理实例引用；此方法有3个参数，第一个是被代理类的类加载器，第二个是被代理类的所有父接口Class数组，第三个是实现动态代理类的实例，在使用此方法返回对象执行目标方法时则执行动态代理实现类的invoke方法，从而实现动态代理。

12 适配器模式比较适合什么样的场景？

适配器模式主要是在原有的类(抽象类)或接口上提供新功能,适配器模式实现通常必需应用继承或扩展接口,适配器类可以是任意抽象类或普通类，一般设计中通常为抽象类或接口提供适当的适配器,以屏蔽普通派生类必需实现抽象父类或接口中定义的抽象方法必需显示定义在代码代码中的尴尬；适配器最适合某抽象类或接口中定义非常多的方法，而子类为能够在业务场景下至需要某一个方法时，避免代码中显示定义本来不在此场景下使用的方法，如果接口中的方法只有一个或则非常少，通常适配器模式没必要使用，因为适配器的实现和其子类的具体应用直接都是耦合性比较强的设计方式。

13 策略模式适合什么样的设计场景采用？

策略模式属于java设计模式行为模式的范畴。

策略模式是一个类的行为或其中逻辑算法可以在运行时根据具体场景被动态更改，也就是业务行为不同时动态切换所应对应的不同策略从而满足业务要求，策略模式适用于行为相似，算法接近 又有所区别的应用场景，可以很好解决类似于局部代码使用if else 逻辑，策略模式避免了繁琐逻辑判断的无休止更改，对于程序的后续扩展具有一定的意义，关键代码实现是定义同一行为的接口。

14 责任链模式是怎样的？

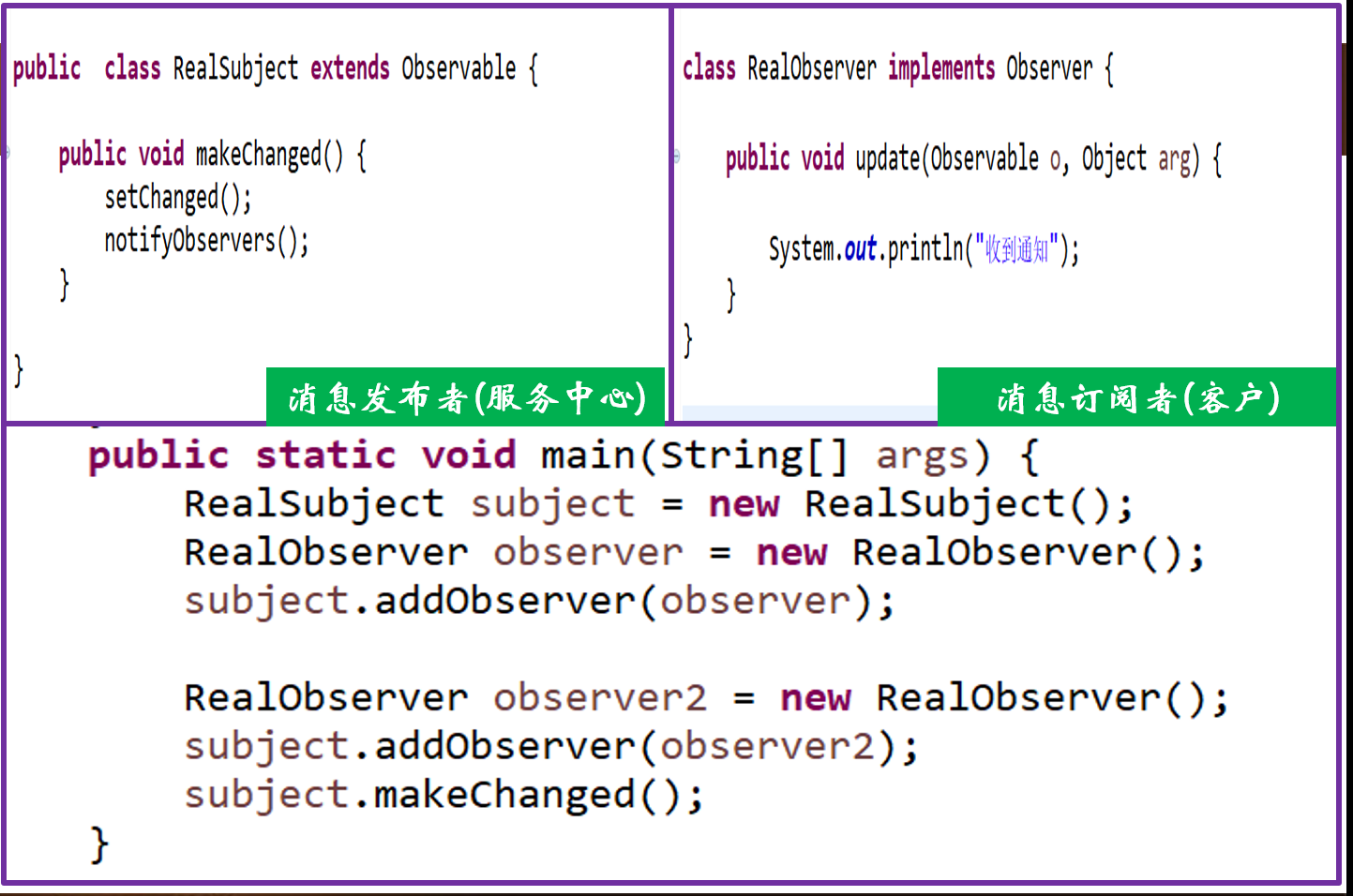
责任链模式是为某个用户的请求创建了一个接收者对象的链条。

责任链模式给予请求的类型，对请求的发送者和接收者进行解耦，通常每个接收者都包含对另一个接收者实例的引用。挡在一个对象不宜或者不能处理用户的请求时，就把把相同的请求传给下一个接收者进行处理，依此类推从而形成一个处理责任链条, 通常在必需获取目标对象时业务链条应尽可能返回请求者所需的资源，而避免链条的无限最大化，如web应用中的请求控制层—>服务层—>数据访问层，就是一种责任链的模式设计。

15 观察者设计模式主要适合什么样的使用场景？

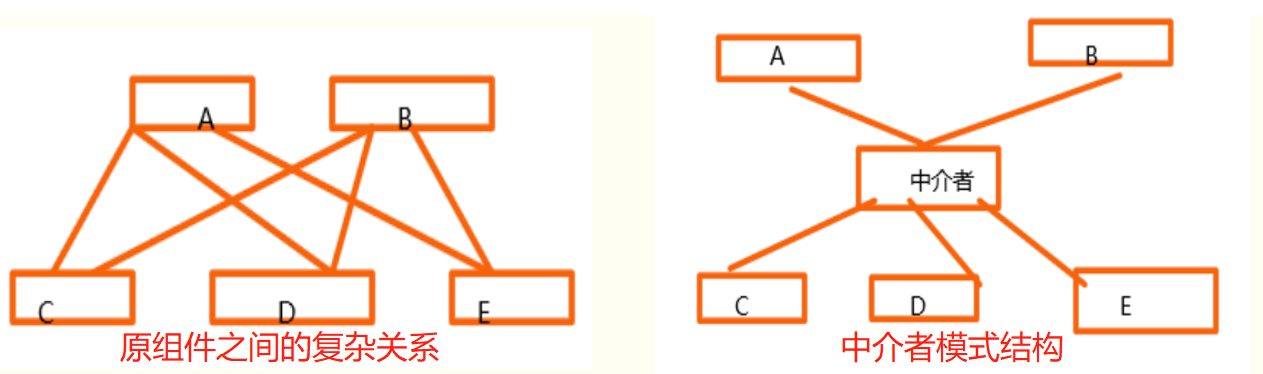
观察者模式是比较特殊的一种行为模式，这种模式主要适用于消息通知业务场景下，通常就是用户(订阅者)在消息服务中心(消息发布者)进行订阅服务，在注册成功以后便可以得到消息中心的适时服务通知；没有注册者或被取消注册者就无法得到服务通知。

通常观察者模式主要包含消息发布接口及实现类，订阅者接口和实现类。Java语言SE平台中提供了观察者模式实现的API，java.util.Observable 服务端消息发布接口；java.util.Observer 客户端订阅者接口。以下是一个简单实现示例



16 介绍一下中介者模式的使用和优缺点。

中介者模式是为了能够简化应用中N多个对象之间的非常复杂关联应用关系而设置一个中介类，这中介类负责N个对象之间复杂关联关系的协调工作。这种情景很类似与一个生活中极其复杂的事情的办理，涉及到很多部门结构的共同协调工作，而这种工作的最终完成可以委托给一个中介公司复杂办理同时协调所有部门结构从而达到最终的目的，可以画一个图形描述一下



中介者模式对于N个对象之间的复杂关系起到了集中管理的优势，分解了N个对象之间的复杂关系定

义，但是由于实现中介者模式需要为中介者以及托管协调对象设计接口及实现类，在系统中如存在多种类似的复杂关联则会造成大量类的定义，对于系统的维护也存在不小的负担。

* 关于webSocket

**1 说明一下你理解的webSocket**

webSocket 是基于HTML5规范提出的一种在TCP协议上能够在客户端与服务端进行全双工即时通信的一种协议。webSocket 并不对HTTP做任何修改，而是对HTTP协议起到了一种补充作用，通常在进行webSocket全双工通信时，需要使用HTTP协议来完成连接的建立，然后客户端和服务端确认后升级为webSocket通信方式从而实现双向即时通信，因为webSocket只需要一次连接成功后在未断开时可以同时从客户端和服务端发送消息，这种方式填补了HTTP每次请求都需要重新确认建立新连接的开销，在类似于消息推送的业务场景下webSocket明显比传统的HTTP更具有优势。

**2 webSocket 和java 中的Socket套接字实现的即时通信有什么不同吗？**

webSocket 是一种基于TCP的长连接协议，只需一次有效连接建立便可以实现双向通信，而且webSocket通常需要在浏览器支持HTML5的场景下进行应用，webSocket可以将任何可确认支持webSocket的其他客户端作为连接对象而没有所谓的单一服务端，这种客户端无需独立设计开发应用而只需要浏览器的支持，webSocket协议实现的通信有自己一套标准的API。

Java 套接字是基于Java 语言实现TCP协议通信的抽象接口组件定义，这种即时通信系统和浏览器没有什么关系，往往需要开发客户端应用和对应程序一个服务端应用程序，多个客户端通常在信息交互时需要通过一个共有建立连接的服务器实现通信业务。当然除java外，其他语言也可以实现基于TCP的套接字应用。

**3 怎么理解全双工通信**

全双工通信泛指可以支持客户端和服务端同时允许向对方发送消息，并且在多次重复发送中完全不必建立新的连接，当前webSocket可以说是实现全双工通信的高效方式，相比在webSocket出现前实现即时通信的ajax轮询以及一次阻塞连接等待消息返回再次发送请求的重复操作是很消耗网络资源且效率低下，webSocket很好的解决了全双工通信的问题。

**4 webSocket 大概如何实现建立确认一个长连接从而实现通信（webSocket怎样利用HTTP完成连接建立）？**

webSocket真正通信过程中虽然不是以HTTP形式发送请求但是webSocket并不能完全独立于HTTP之外，在进行客户端和服务端建立连接时webSocket则必需使用HTTP方式，通常客户端向

服务端发送一段报文，这段报大体和HTTP报文类似，其中Upgrade(升级)参数为websocket，告知服务端将升级为websocket；Connection参数为Upgrade，告知服务端以此升级进行建立连接；报文中还包含Sec-WebSocket-Key的64位加密信息；还有参数Sec-WebSocket-Protocol为chat, superchat；参数Sec-WebSocket-Version: 13 版本信息。

如果服务端支持websocket则响应返回类似的一段报文，其中包含Upgrade参数为websocket；

参数 Connection为Upgrade；参数Sec-WebSocket-Accept是64位加密信息；Sec-WebSocket-Protocol参数值为chat。这样就通过一次HTTP握手实现了升级为websocket协议通信方式，便可以实现双向通信，而在未断开连接中不必再重新请求建立任何新连接。

**5 单工、半双工、全双工通信的区别**

单工就是指A只能发信号，而B只能接收信号，通信是单向发起的，就好像是大海上的灯塔和航船的关系；灯塔只能发出光信号，而航船只能接收灯塔所发出的信号而无法向灯塔发出信号，类似于基于HTTP的web客户端请求服务器就是单工形式，只能由客户端请求服务器而服务器不能想客户端发送请求；

半双工就是指A能发信号给B，B也能发信号给A，但是在此过程中AB双方不能够同时进行信息发送，只能是一方发送完信息后等待令一方发送，或一方接连发送而令一方一直接收，最典型的生活中现象就是传统的警用对讲机；

全双工就是A能够发送给B消息时的同时B也能够同时给A发送消息，而不必等待，生活中就如同打电话一样，双方可以你说我也说，而不必等待对方说完自己再说，也就是收发信息可同时进行。

websocket协议就是实现全双工的通信的一种应用层标准。

**6 websocket、sockJs和STOMP有什么关联和区别?**

简单来说websocket是基于HTML5的实现全双工通信的一个底层协议；sockJs是基于websocket的客户端备选方案它也是底层协议；STOMP（Simple Text Oriented Message Protocol）面向消息的简单文本协议，它基于websocket或 sockJs实现近乎于套接字模式的消息交互，主要用来定义消息语义，在多数情况下全双工通信可以选择STOMP进行实现。

**7 说说webSocket客户端使用标准API实现即时通信的过程**

通常在确定客户端Browser支持webSocket的前提下，使用JS建立WebSocket客户端对象并设置请求服务器的URL；然后在对象成功建立并连接成功后会自动调动使用WebSocket对象执行其onopen方法 ，如果连接失败以及发送和接收数据失败会自动触发调用onerror方法。在成功接收服务器信息时会自动执行调用onmessage方法并通过参数获取服务器的消息。当客户端接收到服务端关闭连接请求时将自动调用onclose方法从而断开连接。总体而言就是webSocket在成功连接后通过一系列回调方法完成即时通信业务直到最后结束。

**8 说说哪些java Web 应用服务器实现了对webSocket的支持**

首先在JEE JSR356 标准规范中定义了webSocket API支持。常见的诸如Oracle公司的webLogic应用服务器版本12c支持webSocket实现，10g和11g也可以通过http publish支持长连接.

Apache社区的Tomcat服务器7.0.5及以上版本支持标准的JSR356标准；还有Resin服务器4.0.5+通过自定义API可以实现对webSocket的支持；还有jetty服务器的7.0以上版本也支持webSocket的标准实现，JSR356 webSocket的标准实现通常都包含在javax.websocket组件包和javax.websocket.server包下。

**9 描述一下java实现服务端websocket的配置和执行过程(Java websocket API)**

在确定使用支持JEE JSR356 标准规范中定义了webSocket服务器后，在项目中库目录下加入java websocket(**websocket-api.jar**)的支持，我们可以自定义类(POJO)并使用注释@ServerEndpoint 进行标注此类是一个WebSocket 服务器的处理端点。此类中通常定义与HTML5 websocket API规范的标准onOpen(Session session)、onMessage(String message)、onError(Throwable t)、onClose(Session session, CloseReason reason) 方法以及receiveMessage(String s)方法。依次使用 @OnOpen、@OnMessage、@OnError、@OnClose和@Message注解进行标注。

类注解@ServerEndpoint必需提供一个url用来表示将 WebSocket 服务端运行在 ws://[Server 端 IP 或域名]:[Server 端口]/websockets/echo 的访问端点，客户端浏览器向这个url发出websocket请求，通常要求自定义类必需具备公共无参的构造器；@OnMessage注释标准的方法用来接收用户发送的消息,消息参数通常可以是普通文本和二级制数据；@OnOpen注解标准的方法在新连接建立时被调用, 参数Session代表客户端对象它提供了客户端连接的诸多细节，一般可以将此Session理解为类似 HTTPSession 会话对象；@OnClose注解标注的方法在连接关闭时被执行，第二个参数CloseReason封装了关闭原因等细节信息；@OnError注解标注的方法在连接出现错误时别调用，参数Throwable是java SE 错误类型顶级父类包含一切可能的错信息；@Message注解标注的方法可以理解为高级的附加交互功能，通常对客户端消息进行过滤，如提供其属性maxMessageSiz设置为一个整数限制消息大小，

注: @Message注解 并非javax. websocket包下标准注解，最好面试中不涉及此内容

**10 基于HTML5的websocket哪些主流浏览器有能够实现支持(websocket对浏览器版本的限制)**

通常PC或移动设备主流浏览器都可以实现对基于HTML5的websocket的支持。

如Google的Chrome4x以上版本；Microsoft 的IE10+版本；火狐Firefox5x以上版本；苹果公司的Safari 5以上版本；Google移动端Android Browser4.5以上版本等都很好实现支持HTML5 websocket。

**11 查阅相关资料完成一个基于java实现的websocket应用示例**

达到

* 关于SOA

1 你是如何理解SOA的？

我的理解是SOA在信息领域是一种软件系统进行架构设计的思想和理念, 通常被称为” 面向服务架构”。由于近些年信息技术的高速发展导致数据信息以成近乎于百倍速度增长,这同时导致了数据处理软件系统的越来越复杂性而不断缩短系统的升级和维护周期，在较传统的系统架构方式下使得系统越来越复杂，耦合性，关联性也随着不断升级而维护越来越难，并且可能因为其紧密的粘合度可能会在一个功能模块出现问题的情况下使得整个系统奔溃，在这种背景下也就需要一种新的架构来满足新的要求，SOA在某些情况下就能够有效解决此类问题。因为SOA最核心的理念就是对复杂系统进行拆分，从而能够让整个系统以多个子系统单独存在并在某种规范下和谐组装，这些子系统可以根据自身情况使用不同语言及部署环境开研发维护升级和独立部署，从而能够解决上述的一些传统系统架构存在的问题。但是所以事物都有它的两面性，有利就有弊，SOA虽然有自身比较独特的优势但同时在使用SOA构架系统时也使得设计、开发、运维成本成倍增长，依赖关系比较复杂，数据一致性和可靠性也会有不小的潜在威胁等等问题，所以个人认为在考虑使用SOA架构时应首选考虑这些弊端并深入分析是否真的有必要来使用SOA进行构建，不能因为新理念就盲目追求使用而忽略未来可能造成的不可控制的后果。

2 你个人认为微服务有什么特点和优势？

首选微服务能够更高层次的解耦, 因为每个微服务系统子系统服务最大程度是分离。因此更独立,开发设计及部署基本都可以独立完成；由于其独立性所以组件化在微服务上体现的更加完美，复用性更强；也因为极强的独立性从而也使得微服务系统更具有超强的业务执行能力，和更快的独立开发速度和单一维护的简单性, 同时这种独立的特性也使得可以继续频繁发布软件并作为自己团队单一管理一种产品，当然在这种特性下也就允许这种架构的子系统可以使用不同的技术模式，也就是不像一个复杂的集成严密统一系统那样基本需要使用一种语言来进行研发了，这些都是微服务架构的一些特点和优势。

3 你认为微服务架构的系统通常是怎么协调运行的？

我认为微服务的运作核心也是最为复杂是服务的发现处理机制，它协调着是否能够及时准确发现服务并注册最终将其功能向网络发布，这是一个承上启下的关键环节，在此之前通常是不同类型客户端发出服务请求并需要通过安全验证机制并授予通行许可，也就是所谓的令牌，接下来在此基础上应有安全网关快速处理用户的请求并且很好处理静态内容和检验每个节点上错误故障，最终能够被发现是一个可用的服务模块系统，在此环节后或利用代理服务器和数据中心环境向分布式网络进行发布最终服务被客户端所应用，这就是一个服务器的基本运行过程，当然也有很多细节，这里就不一一说明了。

4 说说你对DDD(Domain Driven Design) 领域驱动设计的了解。

据我了解领域驱动设计是美国人艾瑞克·伊文思首先提出的软件系统设计构建的一种理论方法被业界称之为DDD模型。

过去在进行系统分析架构设计和实现的过程中基本上系统的分析和系统的设计都是分离开的，这样的方式最终造成了需求分析的结果很大程度上无法很好地直接应用到设计和编程实现，更有甚者能够被最终被运行的代码却将需求分析不同程度上扭曲，最后导致客户应用软件系统后发现很多功能不是自己预期，而且软件系统基本无法快速随业务需求变化而更新。

比如采用用户界面(UI)和数据库(DataBase)的两层架构的面向数据库的架构灵活性极差，对于业务模块基本就是毫无独立性可言；再如采用UI和服务(Service)和数据库的多层SOA架构方式，通常最终随着时间轴的延续会使整个系统服务变得囊肿不堪，难于维护，伸缩性能比较差，再看 DDD融合SOA微服务的事件驱动的命令和查询职责分离(CQRS)架构模式更能够最高层次应付庞大、并发性的复杂业务逻辑，它用聚合模型方法代替数据表模型方式，用并发的事件驱动方法代替串联的消息驱动方法从而能够真正意义上实现以业务实体为核心的灵活拓展，这也是当今乃至未来很长时期内以业务设计为核心的先进理念。这也体现了DDD模型能够准确反映出业务语言(业务模型)，对比传统的JEE以及Spring IOC加ROM等模型只单纯关注数据而不关注业务导致业务增加时很难无支撑特殊业务的实现，其实也就是扩展理念基本没有业务模块的设计思想。而DDD架构最大的特点和优势在于对软件系统需求首先考虑的就是领域模型，也就是所谓的以业务为核心而不是将系统切割成数据和行为，所以能够看出 DDD架构让分析设计者应首先考虑业务而不是数据的统一认识从而需求分析设计与最终实现才能更好体现一致性，当然值得一提的是DDD也是从面向对象为核心理念出发的，这就是我对DDD构建的一些理解。

5 说说你知道的微服务实践框架集?

据我了解现在比较流行的有Spring Boot。它的设计目的是为了能够简化传统Spring应用初始搭建以及开发过程的复杂繁琐性，是比较受欢迎的微服务开发框架之一，利用Spring Boot开发的便捷行简化了分布式系统的开发和部署;

在spring boot 基础上就是Spring Cloud，它基于HTTP或HTTPS的RETS风格构建服务体系，Spring Cloud能够帮助大多数架构师构建相对完整的微服务架构技术生态链条；再有像国内阿里巴巴开源的Dubbo也是由分布式服务化治理框架工具集，通过RPC请求进行访问，它是在其自身的电商平台中逐渐所形成的，所以更适合大型电子商城系统平台，Dubbo的出现还早于Spring Cloud的。国内很多电商平台都使用Dubbo; 还有Akka。这是一个用Scala语言编写实现的，它可以用在有简化编写容错、极强的可伸缩性，集群更加方便。

* 关于spring框架

1 说说你对IOC的理解

IOC 在信息软件行业被称为控制反转，IOC是一种软件系统实现的设计理念也可以说是一种面向对象的设计原则。我记得最早好像是马丁.福勒所提出，主要核心思想是在面向对象程序设计中对象的使用往往只存在一种依赖与被依赖的应用场景，例如在A对象依赖使用B对象情形下，B对象就是被依赖的资源而A对象就是依赖者，通常在早期面向对象设计实现中，会在A对象需要B对象时，在A对象的类中创建被依赖的B对象，当然也可以以弱依赖的方式作为参数传入，无论哪种情况被依赖的对象都需要在应用程序核心源代码中被创建并且维护它的生命周期，这种情况就产生了对象的创建使用极其分散而不能够集中，理论上也不便于管理，也分散了开发人员应专注于业务实现的精力，IOC原则的宗旨是将应用中需要使用的对象以及这种依赖关系和对象生命周期的维护集中进行管理，当然可以使用工程模式进行改进上述问题，但是工厂组件类仍然在应用程序中，也需要维护，因此IOC最终的实现是使得对象工厂能够脱离应用程序而单独存在于应用之外并可以在多种面向对象设计场景下实现重用，像使用非常广泛的spring框架就是一个IOC容器，也就是一个外部对象工厂，从而我们在设计使用面向对象语言，如java设计是就可以将对象的创建从应用代码中托管给spring这样的IOC容器从而实现统一创建和管理并能够让开发人员集中精力专注于软件系统业务，当然成本是开发设计人员要了解学习掌握类似于spring这样的IOC容器。

2 spring框架中AOP都包括哪些主要部分(spring AOP核心实现是怎样的)

spring AOP 核心包括

Aspec称为方面，它是被关注点的核心模块，关注点实可以通过插拔式配置实现对多个业务进行拦截比如事务管理；

Joinpoint称为连接点，连接点是程序执行过程中明确的功能点，常见的像方法的执行调用某些情况下抛出异常等等；

Advice称之为通知，通知是spring AOP实现中特定的连接点，也就是AOP部分要执行的动作，Advice在spring中可以分为BeforeAdvice 前置通知, AfterAdvice 后置通知, ThrowAdvice错误通知以及DynamicIntroductionAdvice动态通知；

Pointcut 称之为切入点， 切入点时当一个通知将被引发时的一个入口描述集合。这个入口必需是用户在应用配置中显示设置从而告诉通知在这里进入，在实际开发中最常使用的就是表达式；

Weaving 称之织入，织入是在spring aop实现过程中进行组装切面并创建一个被通知对象，spring默认使用自身的运行时完成织入动作，当然如果在spring中集成了AspectJ AOP框架可以在编译阶段完成织入。最后只得一提的是spring aop 设计中采用了代理机制实现，通常spring可使用标准的JDK动态代理也可以使用选择CJLIB工具实现代理，这个可以有开发人员来配置决定。

3 spring aop实现中使用JDK动态代理和CGLIB如何选择(二者区别)

对于JDK代理我们知道此实现是主要特点是对接口进行代理实现，从而支持动态代理，而CGLIB是不支持动态代理的，也就是通常不支持接口代理。spring 的aop实现通过组件AopProxyFactory AOP代理工程依据AdvisedSupport通知支持组件对象的配置来决定的。在spring AOP中默认处理中当目标类是接口时候就会使用JDK动态代理实现，相反情况则使用CGLIB来处理并生成代理。

4 spring 应用中在某个bean托管spring容器时autowriter属性不同取值都是什么含义

bean 标签autowire属性可以有以下六个取值。

no不启用自动装配，这是spring的默认设置；

byName：通过属性的名称的方式查找目标bean其实是通过set方法实现；

byType通过属性的类型查找目标bean，通常这种方式应确保上下文中只有一个这样类型的bean对象，否则会出现错误；

constructor使用构造方法注；

autodetect 设置在byType和constructor之间自动的选择注入方式，这种方式使用是非常少的；

defaul方式由上级根标签<beans>的default-autowire属性确定，使用也是很少的情景。

5 Spring AOP实现和AspectJ 框架AOP实现有什么区别

Spring AOP实现是最终是在运行时增强 ，而AspectJ框架实现的AOP机制是编译时增强，这是二者的最大不同点。Spring AOP实现中应用了代理机制，而AspectJ框架AOP是基于字节码操作。Spring 框架AOP已经可以把AspectJ 进行了集成处理，因此在spring中可以选择使用AspectJ的AOP功能，Spring AOP相对比较简单一些，而AspectJ AOP与Spring AOP相比略显复杂学习成本也比较高，但是它的功能也更加强大。在Java领域AspectJ应该是Java生态系统环境中非常完善的AOP框架了，在实际应用如果我们系统设计的切面比较少，那么使用二者时性能差不多，如果切面比较多的情境下，选择AspectJ 是比较合适的，因为AspectJ比Spring AOP执行效率更高一些。

6 在spring 上下文当中被管理的bean对象的作用域都有哪些

spring 上下文中的bean对象可以有5个类型的作用域分别是

singleton单例bean实例，Spring中的bean对象设置的默认形式，也就是整个上下文中只有一个这样的bean对象；

prototype 多例bean实例，创建机制是每次请求都会有容器创建完整的新bean实例。

request 请求域bean对象，每一次HTTP请求都会产生一个新的bean对象，这样的bean只在当前HTTP request内有效，随着请求结束这样的bean将被释放，此种配置只适合web应用；

session会话级别的bean实例，与request级别bean不同的是此对象将被存储与当前用户HTTP session会话域当中；

global-session全局session作用域，这种类型的bean只在Portlet的Web应用中会被使用，在Spring升级为V 5的版本中被取消。Portlet是一个比较小巧的Java Web插件，可以像Servlet一样进行处理基于HTTP web请求，由于应用不是非常广泛，鲜为人知，这可能是spring5取消这个bean作用域的原因吧。

7 说说spring中bean的生命周期过程

在应用spring的应用中可通过xml文件或通过源文件注解对bean进行配置从而在spring加载上下文配置文件或对注解进行扫描后来创建目标bean对象，如果涉及一些属性值设置则会通过目标set方法执行操作，注释方式可以没有set方法。.

如果Bean实现了BeanNameAware接口则会调用setBeanName方法，传入Bean的名字。如果Bean实现了BeanClassLoaderAware接口，调用setBeanClassLoader方法，传入ClassLoader对象的实例。如果Bean实现了BeanFactoryAware接口，调用setBeanClassFacotory方法，传入ClassLoader对象的实例。

如果Bean实现了InitializingBean接口，执行afeterPropertiesSet 方法；如果Bean在配置文件中配置init-method属性，则执行指定的初始化自定义方法；当要销毁一个Bean的时，如果此Bean实现了DisposableBean接口，就将执行调用destroy方法来执行资源释放；在执行销毁Bean的时候，如果这个Bean在配置中的定义包含destroy-method属性，就会执行指定的自定义销毁方法。大体上bean的生命周期就是这样。

8 请描述spring MVC的执行过程(工作原理)（核心构成）

9 说说spring中都支持什么样的事务处理，哪种更提倡使用，配置过程是怎样的

10 spring上下文中的被管理bean singleton和prototype类型的生命周期有什么不同(它们的被管理区别)

11 spring 框架中你了解哪些上下文接口和上下文类

12 据你了解spring中都使用了哪些设计模式

13 在使用spring管理应用程序的bean对象时，是怎么实现线程安全同步处理的

14 你认为springframework和spring boot 有什么不同

15 spring 是否适合管理所有java语言实现的应用程序，说说你自己的看法

* 关于ORM框架

1 说说你对JPA的了解(扩展内容)

三方面，

1. 对象与关系数据库映射元数据，主要是persistence，实现方式XML和注解映射
2. JPA 持久化组件集
3. JPQL 查询语言

2 使用JPA和使用Hibernate有什么不同

3 你认为mybatis和hibernate 在使用上有什么不同，各有什么优势

4 你如何理解持久化单元

5 说说JPA中持久化Bean对象都有哪些状态

6 JPA 中有哪些主要API，规范哪些内容

7 你是怎样理解持久化上下文的

8 JPA 中是否支持多数据源配置，如果支持如何实现

9 完成spring整合JPA应用实践

* 关于Tomcat容器

1 说说Tomcat 设计的最主要目的是什么？

Tomcat 通常在行业内被叫做Java Web服务器或叫容器，它最初的设计目的在于能够支持Java EE 标准组件Servelt和JSP实现的java应用程序容器上运行，从而处理客户端的请求。当然JSP虽然做为java web 应用的动态视图组件，但是仍然会被处理成Servlet的形式，所以也可以称Tomcat是标准的Servlet服务器。它同时也能够支持html静态页面，但是容器不会对html做任何处理，Tomcat和标准的企业级EJB服务器不同，它是Apache软件社区轻量级开源免费的java web容器，在领域内使用频率应该是最高的，对于访问量并发量不是很大的应用，很多时候它都是首选。

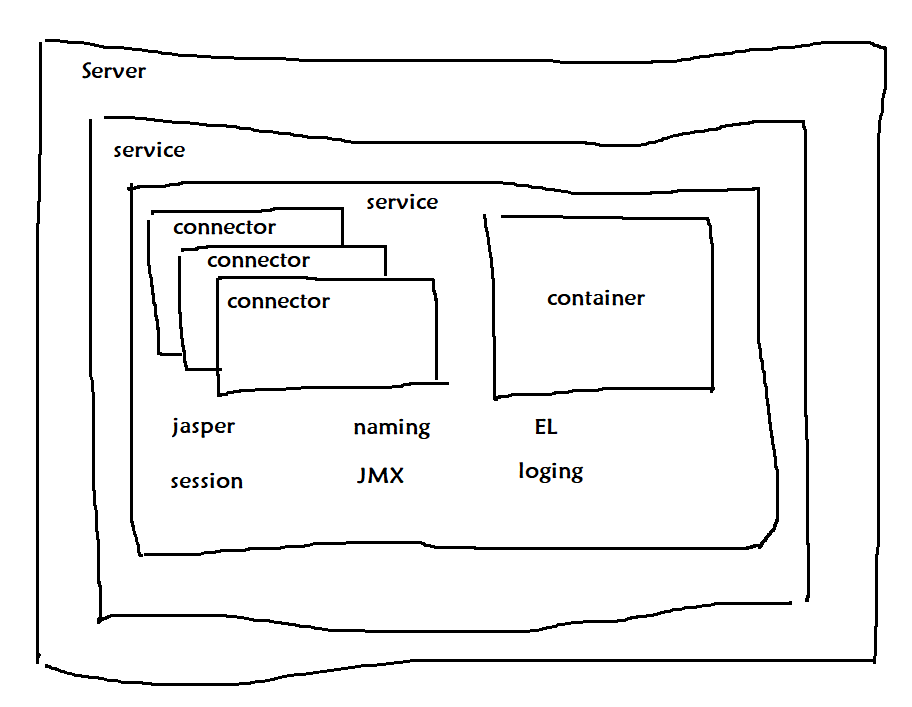
2 能否说一下Tomcat服务器的整体架构

因为也没有打算做Tomcat服务器的专业管理人员，所以可能了解的不够深，我大概说一下我了解的吧。

Tomcat 的最顶层是作为一个Server，也就是一个服务器，它的下面直接子组件是service，代表提供的服务，Tomcat中通常有多个Service ，具有不同的功能；Service下面最主要是由Connector和Container组成，分别代表连接和容器，这两部分也是Tomcat的核心构成。

Container用于处理相关连接请求工作，如提供套接字处理，请求和响应的关系转化等处理；Container用于封装和管理Servlet以及JSP，这也是Tomcat被称作Servlet容器的最主要原因。

当然Tomcat除了这些核心构成还有一些必要的辅助组件，如JSP引擎，资源命名查找，日志系统，会话管理以及JMX等，就说这么多吧，如果需要我可以给您画个简单的构成图。



3 Tomcat 怎样和Apache HTTP服务器配合使用(在Tomcat处理html能力有限的情况如何结合Apache进行配置调优)

这二者的整合无非就是发挥各自的优势，也就是说如果是静态页面则交付Apache服务器处理，如果是动态程序如jsp 就给tomcat进行处理。

概括性来说配置基本是这样的，在准备工作就绪后，通常先配置tomcat，基本有主要的两点，第一，配置connector 的http 端口号，需要注意这个端口号不能和apache相冲突；第二配置connector 的AJP 端口号，这是实现和apache协调工作的关键地方，因为此端口号是apache和tomat通信的入口应保持一致。配置完Tomcat接下来配置Apache服务器，首先配置好监听端口号，

然后将下载好的tomcat-connectors包中的mod.so文件拷贝到apache modules目录下，在http.conf配置文件中添加模块路径等信息,这里就不细说了，接下来在conf目录下添加工作属性文件，也就是和tomcat整合的信息配置最主要的就是AJP端口要一致，最后调整httpd-vhosts.conf 文件修改添加虚拟主及配置信息并告知有有哪些内容交给Tomcat进行处理，这就是最基本的tomcat和apache整合工作的大体配置过程。

4 Tomcat 有哪些部署web应用的不同方式

5 Tomcat 集群是否有了解，能大概说一下吗

6 你自己都了解哪些Tomcat的性能调优

7 Tomcat中有哪些方式可以配置JNDI资源查找

* 关于Quartz

1 使用过Quartz 框架吗，说说你对这个框架的理解

Quartz 是一个用java语言编写的开源免费任务调度管理框架，在Java普通桌面以及基于web的应用中用来解决定时任务的调度管理，java中其实多线程API中有任务执行组件，但是与Quartz框架相比没有Quartz框架功能强大，而且Quartz还支持集群，在一些java应用中当需要执行某些定时任务执行时可以选择Quartz框架从而满足系统需求。

2 能描述一下Quartz框架的主要体系结构吗

3 如果需要对Quartz中的某个Job的执行进行监控记录其状态，运行中，暂停中，等待中应该怎么实现

4 能说一下Quartz中的线程工作情况吗

5 请描述一下Quartz中Job的数据存储是怎样的

6 请简单说说Quartz的集群和存在的问题

7 查阅相关资料完成一个Quartz应用的示例程序

* 关于算法

算法面试中最主要的不是要在面试官给出算法题后，要求面试者给出正确的答案。而是通过算法题来考核面试者能够在短暂思考后，综合平时的算法学习积累给出一个比较正确的解决问题的思考路径或方向从而能够体现自己的分析，思考问题的方式，因为一个问题的答案都是相对的而没有绝对的，算法问题的回答应看做是和面试官一次协作完成的话题探究。

* 关于高并发应用

综合问题：***请谈谈你对当今信息量极其庞大而又存在大量用户并发请求访问应用资源的一些自己看法(诸如此类)***。

是的，随着互联网飞速发展随之而来的是庞大的数据量提升和大量用户并发访问网络的资源棘手问题是近几年被讨论的热议话题，毕竟我涉世不深，对这方面还是学习研究的不是很透彻，不过我还是对此方面有比较大的兴趣，就谈谈自己的看法吧，也希望您多多指教。

综合各类相关方面的资料和实际应用问题，我个人认为必需尽可能解决最基本的三个共性问题。

第一是高并发中请求任务量的合理进行拆分的处理问题。这在很多互联网、书籍、以及在线博客的资源中被叫做分工，我习惯将这个问题称为”任务指派”更容易理解。因为用户请求的并发量非常大，这就像现实生活中一个超市某天来了要把超市挤塌的好多顾客购买物品一样，一个两个人忙不过来，这自然会想到需要雇多个服务人员来解决问题，从而不同的服务人员负责相应的工作而为顾客提供高效优质的服务，这和应用产生大量并发请求处理几乎是没有区别的，其实也就是我个人一直坚持的现实什么样，软件系统就是什么样，这也体现了软件系统就是现实生活的抽象表现。当然从系统应用来讲这些高并发的请求理所当然也就需要分配给适量的线程进行处理，不过这里也要注意一个问题，还是从生活角度来说吧。超市要能够基本评估大约有多少顾客，需要雇多少个有什么样能力的服务员提供服务，因为少了用户不满意，多了超市开销也是不小，反而不美。话说回来就是系统中应该衡量使用多少个线程能够很好解决高并发更加合适，毕竟线程多了对系统的负担也是很大的，可能适得其反。再就是每个线程具备什么样的能力，做它应该做的事情，这和雇超市服务员是相同的，他们也需要进行很好的管理，否则可能会出乱子，当然对于具体分工的API javaSE平台也提供了像Executor 、Future等接口和实现类作为可选的工具，这个就不说了。

第二就是线程间的协调同步问题，对于刚刚的超市现象就是服务人员的合作工作的问题，因为每个人的工作不会是完全独立的，好比王姐买苹果，李哥是负责给王姐的货架上货，李姐在买掉多少苹果会通知李哥需要工作了，应该为王姐的货架上苹果了，否则李哥就需要等待李姐的命令，当然现实中李哥也是有可能为张姐，白姐也提供这样的支持，我想应用中线程也是如此，需要这种同步协调度才能发挥最高的效率您说对吧，当然其实就是一个线程任务执行完毕通知另外一个线程进行任务，当然也存在及其特殊情况某个任务以外终止或有条件中断而让另外一个线程开始工作的可能。对于线程的同步处理当然java SE平台也提供了CountDownLatch倒计时辅助锁、 CyclicBarrier迭代屏障、Lock接口以及语法糖synchronized 重量级同步锁供给我们选择，哦，值得一提的是synchronized这种语法糖对于解决高并发的性能而言肯定是下下之策，通常不会使用。

对了，刚刚我上面两点的看法都是从性能出发，解决的是效率问题，下面这第三点我个人认为会对前面两点有着极其重要的影响，您可能已经想到了，就是如何保证并发下的数据安全问题，很多专业可能会称为数据一致性，业务原子性，或者叫得更多的互斥保证，我为了便于理解记忆，我自己用一句俗话进行记忆，那就是“安全第一” ，因为您也知道，无论什么样的系统，或大或小，最终的目的无非就是一个为了数据的共享，也就是说数据是核心，这也就应该必需保证数据的正确性，就如同财务的问题不允许出现任何错误，会计在拢账的时候，出纳在着急用钱也必需等待会计把账目弄完，否则就会岔子，这也就是我刚刚说过的如果不能保证数据的安全正确性，恐怕前面两点做的再好都是没有什么太大意义的。所以我个人认为高并发最直接的貌似解决性能，但是最终的核心仍然是数据的安全访问保障。我们回到应用中当然就是线程间的安全问题，其实也就是数据安全问题，也就是我们绕不开的话题线程同享变量。在解决互斥而达到安全的目的我们当然可以根据情况选择使用synchronized、Lock、ThreadLocal本地变量、复制写、以及final终极模式，这里我还是要强调一下，我前面说过如果只考虑性能问题我们通常不会使用synchronized，但是特殊情况在单系统中为了保证绝对安全可能真的必需使用synchronized以确保数据的安全性，比如财务问题。

最后我提出一个自己的问题，因为这个问题我似乎还没有看到哪里能给出一个参考答案，也就是

在保证数据安全的前提下，高并发对于应用级的最佳实现是什么？对于设计架构级的最佳实现是什么？

以上就是我的一些不成熟的对于高并发的理解和自己的看法，还请您斧正。