# Java基础：

## 简述java程序执行的过程

由java源文件，通过javac.exe可执行文件，编译为字节码的class文件，然后通过解析器解析执行class文件，解析为计算机识别的二进制编码。

这几年的开发,我基本采用的是1.7版本,当然也使用过1.8版本,在新版的eclipse,必须使用1.8版本,才能运行 lamda

## 各种排序方式哪个效率高一些

测试数据100000条进行排序时，实现效率如下

冒泡排序：13098ms

选择排序：2334ms

插入排序：1264ms

快速排序：23ms

通过测试，几种排序效率差距有些大！快速排序效率最高

## new的线程放在堆里还是栈里?

堆里，对象实例放到堆里边，引用放到栈中

## Overload和Override的区别。Overloaded的方法是否可以改变返回值的类型?

方法的重写Overriding和重载Overloading是Java多态性的不同表现。

重写Overriding是父类与子类之间多态性的一种表现，重载Overloading是一个类中多态性的一种表现。

如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽”了。

如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。

Overloaded的方法是可以改变返回值的类型,它只是与函数名和参数相关,与返回值类型无关。

## HashCode和equals()方法的作用是什么？有什么区别？是否必须存在？请说明理由

 hashCode()方法和equal()方法的作用其实一样，在[Java](http://lib.csdn.net/base/java)里都是用来对比两个对象是否相等一致;

 在实际操作经验中,我发现重写的equal（）里一般比较的比较全面比较复杂，这样效率就比较低，而利用hashCode()进行对比，则只要生成一个hash值进行比较就可以了，效率很高.

然而,hashCode()并不是完全可靠，有时候不同的对象他们生成的hashcode也会一样（当然不排除生成hash值得公式可能存在的问题）

总结起来就是:

1.equal()相等的两个对象他们的hashCode()肯定相等，也就是用equal()对比是绝对可靠的。

2.hashCode()不相等的两个对象他们的equal()肯定不相等

3.equal()相等的两个对象他们的hashCode()也可能相等

4.hashCode()相等的两个对象他们的equal()不一定相等，也就是hashCode()不是绝对可靠的。

所以,重写equals(Object obj)方法，有必要重写hashcode()方法,对于需要大量并且快速的对比的话如果都用equal()去做显然效率太低，所以解决方式是，每当需要对比的时候，首先用hashCode()去对比，如果hashCode()不一样，则表示这两个对象肯定不相等（也就是不必再用equal()去再对比了）,如果hashCode()相同，此时再对比他们的equal()，如果equal()也相同，则表示这两个对象是真的相同了，这样既能大大提高了效率也保证了对比的绝对正确性！

## Java集合

Java里边主要有两种集合，collection接口和map接口，其中collection下又包含list和set两个子接口：

**List子接口**：有序，可以有重复元素。和数组类似，List可以动态增长，查找元素效率高，相对的插入删除元素效率低，因为会引起其他元素位置改变。

**Set子接口**:无序，不允许重复。检索元素效率低下，删除和插入效率高，插入和删除不会引起元素位置改变。set集合中的元素**不按特定方式排序**，只是简单的把对象加入集合中，就像往口袋里放东西。

Map：双列集合，用于存放键值对。键值是唯一的，不可重复

**List接口下有三个实现类ArrayList 、LinkedList和Vector**

**Vector**是一种老的动态数组，是线程同步的，效率很低，一般不赞成使用

**ArrayList**实现一个动态数组，它的规模可变并且能像链表一样被访问。它提供的功能类似Vector类但**不同步**，它是以Array方式实现的List，**允许快速随机存取**。特点是读快改慢；

**ArrayList**是基于数组实现的，是一个动态数组（长度可变的数据结构，允许null当元素，默认长度是10，一次增长0.5倍，其容量为原来的1.5倍。

**LinkedList**实现一个链表，提供最佳顺序存取，适合插入和移除元素。由这个类定义的链

表也可以像栈或队列一样被使用。提供**最佳顺序存取**，**适合插入和移除元素**。特点是改快读慢

**Set接口有HashSet和TreeSet两个实现类**

**HashSet**能够快速定位一个元素，

要注意的是：**存入HashSet中的对象必须实现HashCode()方法**；

HashSet实现了Set接口，它不允许集合中有重复的值，当我们提到HashSet时，第一件事情就是在将对象存储在HashSet之前，要先确保对象重写equals()和hashCode()方法，这样才能比较对象的值是否相等，以确保set中没有储存相等的对象。如果我们没有重写这两个方法，将会使用这个方法的默认实现。

**TreeSet** 将放入其中的元素**按序存放**。它实现的是SortedSet接口，也就是加入了对象比较的方法。通过对集中的对象迭代，我们可以得到一个升序的对象集合。

Map接口的实现类主要有**HashMap** 、**HashTable和TreeMap；**

**当元素的顺序很重要时选用TreeMap，当元素不必以特定的顺序进行存储时，使用HashMap。**HashMap不是同步的，**Hashtable是同步的，但Hashtable不推荐使用，因为HashMap提供了所有类似的功能，并且速度更快。当需要在多线程环境下使用时，HashMap也可以转换为同步的。**HashMap没法保证映射的顺序一直不变，但是作为HashMap的子类LinkedHashMap可以。

HashMap可以通过Map m = Collections.synchronizedMap（hashMap）来达到同步的效果。



## set为什么不可以有重复？

set是一个特殊的Map，value值都是一个object，key值就是set的值，当key一样时，hash算法会计算到同一个节点，所以不能重复

## 数据结构

常用的有四类基本结构：集合、线性结构、树形结构、图状结构

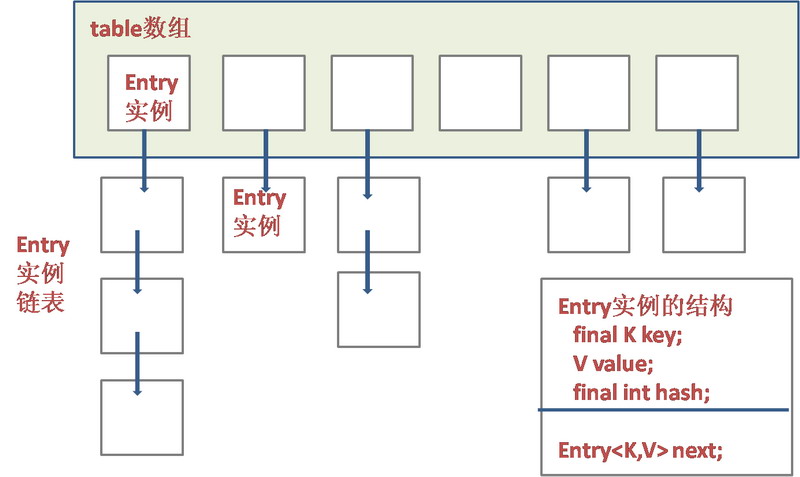
## 说出ArrayList,Vector, LinkedList的存储性能和特性？

ArrayList 和Vector他们底层的实现都是一样的，都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以查询数据快而插入,删改数据慢。  当Vector或ArrayList中的元素超过它的初始大小时,Vector会将它的容量翻倍,而ArrayList只增加50%的大小，这样,ArrayList就有利于节约内存空间

Vector中的方法由于添加了synchronized修饰，因此Vector是线程安全的容器，但性能上较ArrayList差，因此已经是Java中的遗留容器。

LinkedList使用双向链表实现存储（将内存中零散的内存单元通过附加的引用关联起来，形成一个可以按序号索引的线性结构，这种链式存储方式与数组的连续存储方式相比，内存的利用率更高），按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以增删改,查询慢些。

## HashMap底层原理

  
 HashSet是按照哈希算法来存取集合中的对象，具有很好的存取和查找性能，当向集合中加入一个对象时，HashSet会调用对象的hashCode()方法来获取哈希码，然后根据这个哈希吗来计算对象在集合中的存放位置。  
在Object类中定义了hashCode()和equal()，equal()是按照内存地址比较对象是否相同，如果object1.equal(object1)w为true时，则表明了object1和object2变量实际上引用了同一个对象，那么object1和object2的哈希码也是肯定相同。

## 堆和栈的区别？

heap是堆,空间是很大的自由区;stack是栈,空间有限。  
从存储内容和分配空间上看:

栈：基本类型的变量和对象的引用变量都在函数的栈内存中分配, 当在一段代码块定义一个变量时，Java就在栈中为这个变量分配内存空间，当超过变量的作用域后，Java会自动释放掉为该变量所分配的内存空间，该内存空间可以立即被另作他用。

堆：　堆内存用来存放由 new 创建的对象和数组，在堆中分配的内存，由 Java 垃圾回收器来管理。

## Java中有几种方法可以实现一个线程？用什么关键字修饰同步方法？

(1)继承Thread类，重写run函数

(2)实现Runnable接口，重写run函数

(3)实现Callable接口，重写call函数

Callable是类似于Runnable的接口，实现Callable接口的类和实现Runnable的类都 是可被其它线程执行的任务。

  Synchronized

改类的目的。

## sleep() 和 wait() 有什么区别?

① 这两个方法来自不同的类分别是，sleep来自Thread类，和wait来自Object类。

sleep不出让系统资源,依然保持监控状态，到时后会自动恢复；wait是进入线程等待池等待，出让系统资源，其他线程可以占用CPU。一般wait不会加时间限制，要等待其他线程调用notify/notifyAll唤醒等待池中的所有线程，才会进入就绪队列等待OS分配系统资源。sleep(milliseconds)可以用时间指定使它自动唤醒过来，如果时间不到只能调用interrupt()强行打断。

Thread.sleep(0)的作用是“触发操作系统立刻重新进行一次CPU竞争”。

② 使用范围：wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用。

   synchronized(x){   
      x.notify()   
     //或者wait()

## java 中操作字符串都有哪些类？它们之间有什么区别？

操作字符串的类有：String、StringBuffer、StringBuilder。

String 和 StringBuffer、StringBuilder 的区别在于 String 声明的是不可变的对象，每次操作都会生成新的 String 对象，然后将指针指向新的 String 对象，而 StringBuffer、StringBuilder 可以在原有对象的基础上进行操作，所以在经常改变字符串内容的情况下最好不要使用 String。

StringBuffer 和 StringBuilder 最大的区别在于，StringBuffer 是线程安全的，而 StringBuilder 是非线程安全的，但 StringBuilder 的性能却高于 StringBuffer，所以在单线程环境下推荐使用 StringBuilder，多线程环境下推荐使用 StringBuffer。

## 什么是线程池？

线程池是指在初始化一个多线程应用程序过程中创建一个线程 集合，然后在需要执行新的任务时重用这些线程而不是新建一个线程。线程池中线程的数量通常完全取决于可用内存数量和应用程序的需求。然而，增加可用线程数量是可能的。线程池中的每个线程都有被分配一个任务，一旦任务已经完成了，线程回到池子中并等待下一次分配任务。

## 线程的状态以及状态之间的相互转换

1、新建状态(New)：新创建了一个线程对象。

2、就绪状态(Runnable)：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

3、运行状态(Running)：就绪状态的线程获取了CPU，执行程序代码。

4、阻塞状态(Blocked)：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

①等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。

②同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。

③其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

1. 死亡状态(Dead)：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

## 线程安全的概念？

当多个线程执行一个并发任务时,始终都能得到我们想要的结果,就是线程安全的.

在某一段代码块或方法上使用synchronized关键字,使这段代码成为线程安全的代码.

但并不是使用了synchronized关键字,一定能保证线程安全.因为如果有多个线程,多把锁,两个锁互不影响,这时需要在具有synchronized关键字的方法上加上static,将锁变成类级别的锁(一把锁).

## 终止线程的方法有哪些?

使用退出标志,使线程正常退出,也就是当run方法完成后线程终止. 2.使用stop方法强制终止线程(这个方法不推荐使用,因为stop和suspend、resume一样,也可能发生不可预测的结果). 3.使用interrupt方法中断线程.

## String 类的常用方法都有那些？

**indexOf()：返回指定字符的索引。**

**charAt()：返回指定索引处的字符。**

**replace()：字符串替换。**

**trim()：去除字符串两端空白。**

**split()：分割字符串，返回一个分割后的字符串数组。**

**getBytes()：返回字符串的 byte 类型数组。**

**length()：返回字符串长度。**

**toLowerCase()：将字符串转成小写字母。**

**toUpperCase()：将字符串转成大写字符。**

**substring()：截取字符串。**

**equals()：字符串比较。**

## 普通类和抽象类有哪些区别？

**普通类不能包含抽象方法，抽象类可以包含抽象方法。**

**抽象类不能直接实例化，普通类可以直接实例化。**

## 接口和抽象类有什么区别？

**实现：抽象类的子类使用 extends 来继承；接口必须使用 implements 来实现接口。**

**构造函数：抽象类可以有构造函数；接口不能有。**

**main 方法：抽象类可以有 main 方法，并且我们能运行它；接口不能有 main 方法。**

**实现数量：类可以实现很多个接口；但是只能继承一个抽象类。**

**访问修饰符：接口中的方法默认使用 public 修饰；抽象类中的方法可以是任意访问修饰符。**

## java 中 IO 流分为几种？

**按功能来分：输入流（input）、输出流（output）。**

**按类型来分：字节流和字符流。**

**字节流和字符流的区别是：字节流按 8 位传输以字节为单位输入输出数据，字符流按 16 位传输以字符为单位输入输出数据。**

## BIO、NIO、AIO 有什么区别？

**BIO：Block IO 同步阻塞式 IO，就是我们平常使用的传统 IO，它的特点是模式简单使用方便，并发处理能力低。**

**NIO：New IO 同步非阻塞 IO，是传统 IO 的升级，客户端和服务器端通过 Channel（通道）通讯，实现了多路复用。**

**AIO：Asynchronous IO 是 NIO 的升级，也叫 NIO2，实现了异步非堵塞 IO ，异步 IO 的操作基于事件和回调机制。**

## Files的常用方法都有哪些？

**Files.exists()：检测文件路径是否存在。**

**Files.createFile()：创建文件。**

**Files.createDirectory()：创建文件夹。**

**Files.delete()：删除一个文件或目录。**

**Files.copy()：复制文件。**

**Files.move()：移动文件。**

**Files.size()：查看文件个数。**

**Files.read()：读取文件。**

**Files.write()：写入文件。**

## 多线程用途

1、常见的浏览器、Web服务(现在写的web是中间件帮你完成了线程的控制)，web处理请求，各种专用服务器(如游戏服务器)

2、servlet多线程

3、FTP下载，多线程操作文件

4、[数据库](http://lib.csdn.net/base/mysql)用到的多线程

6、tomcat，tomcat内部采用多线程，上百个客户端访问同一个WEB应用，tomcat接入后就是把后续的处理扔给一个新的线程来处理，这个新的线程最后调用我们的servlet程序，比如doGet或者dpPost方法

7、后台任务：如定时向大量(100W以上)的用户发送邮件；定期更新配置文件、任务调度(如quartz)，一些监控用于定期信息采集

8、自动作业处理：比如定期备份日志、定期备份数据库

9、异步处理：如发微博、记录日志

10、页面异步处理：比如大批量数据的核对工作(有10万个手机号码，核对哪些是已有用户)  
线程间的通信方式

**①同步**

这里讲的同步是指多个线程通过synchronized关键字这种方式来实现线程间的通信。

由于线程A和线程B持有同一个MyObject类的对象object，尽管这两个线程需要调用不同的方法，但是它们是同步执行的，比如：线程B需要等待线程A执行完了methodA()方法之后，它才能执行methodB()方法。这样，线程A和线程B就实现了 通信。

这种方式，本质上就是“共享内存”式的通信。多个线程需要访问同一个共享变量，谁拿到了锁（获得了访问权限），谁就可以执行。

**②while轮询的方式**

在这种方式下，线程A不断地改变条件，线程ThreadB不停地通过while语句检测这个条件(list.size()==5)是否成立 ，从而实现了线程间的通信。但是这种方式会浪费CPU资源。之所以说它浪费资源，是因为JVM调度器将CPU交给线程B执行时，它没做啥“有用”的工作，只是在不断地测试 某个条件是否成立。就类似于现实生活中，某个人一直看着手机屏幕是否有电话来了，而不是： 在干别的事情，当有电话来时，响铃通知TA电话来了。关于线程的轮询的影响，可参考：JAVA多线程之当一个线程在执行死循环时会影响另外一个线程吗？

这种方式还存在另外一个问题：

轮询的条件的可见性问题，关于内存可见性问题，可参考：JAVA多线程之volatile 与 synchronized 的比较中的第一点“一，volatile关键字的可见性”

线程都是先把变量读取到本地线程栈空间，然后再去再去修改的本地变量。因此，如果线程B每次都在取本地的 条件变量，那么尽管另外一个线程已经改变了轮询的条件，它也察觉不到，这样也会造成死循环。

**③wait/notify机制**

代码如下：

线程A要等待某个条件满足时(list.size()==5)，才执行操作。线程B则向list中添加元素，改变list 的size。

A,B之间如何通信的呢？也就是说，线程A如何知道 list.size() 已经为5了呢？

这里用到了Object类的 wait() 和 notify() 方法。

当条件未满足时(list.size() !=5)，线程A调用wait() 放弃CPU，并进入阻塞状态。---不像②while轮询那样占用CPU

当条件满足时，线程B调用 notify()通知 线程A，所谓通知线程A，就是唤醒线程A，并让它进入可运行状态。

这种方式的一个好处就是CPU的利用率提高了。

但是也有一些缺点：比如，线程B先执行，一下子添加了5个元素并调用了notify()发送了通知，而此时线程A还执行；当线程A执行并调用wait()时，那它永远就不可能被唤醒了。因为，线程B已经发了通知了，以后不再发通知了。这说明：通知过早，会打乱程序的执行逻辑

**④**管道通信就是使用java.io.PipedInputStream 和 java.io.PipedOutputStream进行通信

具体就不介绍了。分布式系统中说的两种通信机制：共享内存机制和消息通信机制。感觉前面的①中的synchronized关键字和②中的while轮询 “属于” 共享内存机制，由于是轮询的条件使用了volatile关键字修饰时，这就表示它们通过判断这个“共享的条件变量“是否改变了，来实现进程间的交流。而管道通信，更像消息传递机制，也就是说：通过管道，将一个线程中的消息发送给另一个。

## 反射机制主要功能

在运行时判断任意一个对象所属的类； 

在运行时构造任意一个类的对象； 

在运行时判断任意一个类所具有的成员变量和方法； 

在运行时调用任意一个对象的方法； 

生成动态代理。

## 说明: jvm原理图JVM虚拟机

## 说明: jvm原理图堆和栈

**①栈的空间比较小，**存放函数的参数值，局部变量的值等**。**

**②堆的空间比较大，用于存放new的对象。**

**③**与C++不同，Java自动管理栈和堆，程序员不能直接地设置栈或堆。

**④**堆（数据结构）：堆可以被看成是一棵树，如：二叉树。

**⑤**栈（数据结构）：一种先进后出的数据结构。

## gc是什么，几种方法?

gc是垃圾回收机制，垃圾回收器要负责三项任务：分配内存、确保被引用对象的内存不被错误地回收以及回收不再被引用的对象的内存空间。垃圾回收器的存在一方面把开发人员从释放内存的复杂工作中解脱出来，提高了开发人员的生产效率；另一方面，对开发人员屏蔽了释放内存的方法，可以避免因开发人员的错误地操作内存而导致应用程序的崩溃，保证了程序的稳定性。但是，垃圾回收也带来的问题，为了实现垃圾回收，垃圾回收器必须跟踪内存情况，释放没用的对象，在完成内存释放后还需要处理堆中的碎片，这些操作增加了JVM的负担，从而降低程序的执行效率。

  System.gc()方法,

  finalize()方法

# Javaweb基础：

## 存储过程

存储过程（procedure）是为了完成特定功能的SQL 语句集，存储在数据库中，经过第一次编译后再次调用不需要再次编译。存储过程里边可以定义变量、写if判断、写循环。他不能return返回，但是可以通过参数返回，有三种参数（in输入、our输出 还有 inout 即可输入又可输出。因为存储过程只在创造时进行编译，以后每次执行存储过程都不需再重新编译，而一般SQL语句每执行一次就编译一次,所以使用存储过程可提高数据库执行速度。当对数据库进行复杂操作时(如对多个表进行Update,Insert,Query,Delete时），可将此复杂操作用存储过程封装起来与数据库提供的事务处理结合一起使用。.存储过程可以重复使用,可减少数据库开发人员的工作量。安全性高,可设定只有某此用户才具有对指定存储过程的使用权。

当然他也有缺点，最大的缺点就是可移植性查，另外学习起来也有一定的难度。还有就是采用存储过程后也就意味着你的系统有一些业务逻辑不是在应用程序里处理，这种架构会增加一些系统维护和调试成本。所以，个人认为，普通业务逻辑尽量不要使用存储过程，定时性的ETL任务或报表统计函数可以根据团队资源情况采用存储过程处理。存储过程可以快速解决问题，但是移植性、维护性、扩展性不好，它有时会约束软件的架构，约速程序员的思维，在你的系统没有性能问题时不建议用存储过程。如果你要完成的功能只是一次或有限次的工作，如数据订正、数据迁移等等，存储过程也可以拿上用场。如果你的系统很小，并且有50%的开发人员熟练掌握PL/SQL，人员结构稳定，那存储过程可以减少很多代码量，并且性能不错。当系统变复杂了，开发人员多了，存储过程的弊端就会呈现。mysql存储过程用call调用，Oracle用excecute调用；mysql写存储过程，一般先要用delimiter $$，设置 $$为命令终止符号，代替分号 ,因为分号在存储过程里边要用；创建好以后， 在用delimiter ；恢复；

我之前写过一些存储过程，但最近接触的比较多的还是电商类的互联网项目，这类项目中，存储过程并不建议使用。最近一次存储过程记得是，为测试人员写的一个创建测试数据的存储过程。当时测试人员为了进行报表统计功能的测试，需要经常重复性的生成大量的数据，还设计到了很多张表。如果手动一条条加数据特别麻烦，于是找我想个办法，我就帮他写了一个存储过程；好久没写了，手比较生，当时也是一边找资料，一边写的。很简单的东西，写了近2个小时；

函数（function）与存储过程的结构类似，但是存储过程中不能用return返回值，函数必须有一个return子句，用于返回函数值；当然函数也可以用类似存储过程的参数返回；函数除了直接调用以外，可以直接放在select语句中使用，存储过程不行，存储过程定义关键字用procedure，函数定义用function。一般情况下如果要求有且有一个返回值，用函数，否则，用存储过程。

触发器（trigger）一般是执行sql语句的增删改查时使用，比如修改A表的一条数据时，让他自动触发一个操作，在B表里边新增一条记录；触发器因为需要绑定在表上，所以有数量限制，他里边的语法和存储过程、函数类似。

视图(View)实际上是多张表之间进行关联查询，把查询结果虚拟成一张表，就叫视图。创建视图也非常简单，create view viewname as + 查询语句，就可以了。视图的使用和表的使用完全一样，但不要在视图上进行增删改的操作，因为他本身就是一个查询结果的集合。利用视图，可以使我们把复杂的查询简单化，可以增强数据的安全性，屏蔽用户对基表的操作；他一般无法提高查询效率。

## 存储过程与函数的区别

不同点：

1、标识符不同。函数的标识符为FUNCTION，过程为：PROCEDURE。

2、函数中有返回值，且必须返回，而过程可以没有返回值。

3、过程无返回值类型，不能将结果直接赋值给变量；函数有返回值类型，调用时，除在select中，必须将返回值赋给变量。

4、函数可以在select语句中直接使用实现对字段进行计算，而存储过程不能，

例如：假设已有函数fun\_getAVG() 返回number类型绝对值。

那么select fun\_getAVG(col\_a) from table 这样是可以的。

5、函数一般处理简单的逻辑方便，存储过程一般处理复杂的逻辑，

相同点：

1、二者都可以有输出

2、二者写法逻辑上很相似

## 数据库优化应该从哪些方法考虑

1. **数据库选型：**首先应该从数据库的选型来看，首先是关系型数据库，目前最流行的是mysql和Oracle，mysql更适合于中小型项目，安装使用比较简单，免费开源。Oracle适合于大型项目，安装包比较大，使用比较耗内存，收费，但是效率高、可靠性好，使用行级锁，比mysql的表级锁粒度更细，对并发性的支持要好的多。NOSQL数据库（比如：mongdb），事务性比较差，但效率高，主要适用于数据量比较大，但关系型要求不是很强的数据，比如评论数据。缓存数据库（比如：redis），由于保存在内存中，所以速度特别快，但是不适应持久化保存。所以缓存数据库一般作为应用和数据库的中间层来使用。
2. **设计规范：**其次应该从数据库设计规范来考虑。好的规范，可以是数据库运行效率更高，优化更方便。以mysql为例；

* 由于mysql对大小写在windows和linux上敏感型不一致，为了避免麻烦，我们一般建议数据库名、表名、字段名都用小写+下划线命名；
* 数据库名最后于项目名称一致。
* 表名最好采用：“数据库名\_表名”的方式；
* 字段名最后见名知意，不要太长；
* 字段选用合适的字段类型，比如姓名可以用varchar（10），而不要用默认的长度255，日期最好用date或datetime或时间戳。金额最好用decimal，而不是double、float等。
* 关联表以外的表最好都有主键id、创建时间、创建人、最后修改时间、最后修改人字段。
* 数据库设计时，以三大范式为参考，但是必要时，可以使用冗余；
* 对于预期数据量特别大的表，可能没有都会有很大增长的表，要考虑水平分表，比如系统日志表，可以考虑表名中加入月份，每月生成新表。
* 对于字段比较多的表，比如企业信息表，可以考虑纵向分表，把表分成两张1对1的表。把经常查询的字段放在一张表里，不经常用到的和大文本字段放在另外一张表里。
* 开发阶段，介意每个表只有主键，不应该包含其他外键。因为开发过程中，系统不稳定，可能出现很多垃圾数据，有了外键，数据的清理会给我们开发照成很多不便。要求开发人员用代码维护表之间的关系，这样更能增加代码的健壮性。

1. **索引：**开发测试完成以后，一定要记得给数据库添加索引，增加sql的执行效率。添加索引也要注意，索引应该添加到经常作为查询条件或者排序字段、或者作为外键的列上。而那些不经常查询的字段、大文本字段、可选值比较少的字段（比如性别）等不需要创建索引。另外，sql中的不等于、is null、函数、like两边同时使用%等，会使索引失效，应该尽量避免使用；
2. **配置：**数据库的配置优化（my.ini)，比如：最大连接数、最大缓存数、最大线程数、物理内存等；
3. **读写分离**、主从复制优化。因为一般查询多、修改少，索引可以增加查询效率，但会影响修改效率。所以做了读写分离以后，可以把索引只建立在读的数据库上。并且查询不需要事务，所以可以用非事务行引擎；

**数据库集群**，数据库压力过大时，用集群来分担压力是个很不错的方案；

## 索引

索引可以理解为数据的查询目录，建索引的目的就是提高对表的查询速度；没有索引时，查询时全表检索，有了索引就可以根据索引快速查找需要的数据；但是索引也不能乱建，因为索引需要维护，会导致增删改的效率降低。会使数据的维护变的复杂，影响开发的效率，索引也会占用数据库的物理空间；所以我们一般在项目的开发阶段、测试阶段、试运行阶段都很少去创建索引，因为有了索引，系统bug造成垃圾数据特别不好删除。只有在项目正式上线后才去增加索引，以提高项目运行速度。索引我们一般都创建在经常作为查询条件的字段、排序的字段和作为关联关系的字段上边。尽量避免在大文本字段、数据量比较小的字段（比如性别），增删改性能大于检索性能的字段上边；另外，有些情况，即使增加了索引，索引也不会生效，比如：索引字段使用了不等于（！=或者<>）符合，用了函数、进行了运算，使用了is null或is not null，和不匹配的数据类型进行比较、like查询时两边都用了%等；还有一个要注意的地方是，如果在多个字段上建立联合索引，那么组合索引的第一个列被where子句引用时，索引才会起作用。因为想要使用索引增加查询效率，必然要牺牲增删改效率，为了解决这个问题，我们经常对数据库做主从复制，读写分离。同时创建两个数据库，一主一从，两个数据库数据完全一致，主的数据库用来进行写的操作，操作后数据库会自动把数据同步到从的数据库，从的数据库用来执行读的操作。这样我们建立索引时，就可以只在读的数据库创建就可以了。这样就索引即能增加查询效率，有不影响增删改效率。这样做了之后，我们还可以对他进一步优化，比如数据库引擎的优化，主数据库因为执行增删改操作，所以用事务型引擎Innodb，读的数据库，不需要事务，就可以用效率更高的MyIASM引擎。同时根据实际情况，也可以配置一主多从或者多主多从。索引的创建常用的有2中方式：CREATE 【UNIQUE】INDEX index\_name ON table\_name (column\_list)；或者ALTER TABLE table\_name ADD INDEX index\_name (id,name)；修改用：ALTER TABLE table\_name REBUILD INDEX index\_name (column\_list)；删除用：DROP INDEX index\_name ON talbe\_name或者：ALTER TABLE table\_name DROP INDEX index\_name查看用：select \* from all\_indexes where table\_name='student';Conlum\_list中多个字段用”，”号分割。

## 索引的作用是什么？

1.通过创建[唯一索引](https://www.baidu.com/s?wd=%E5%94%AF%E4%B8%80%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y1rAf4nvNWrHfkmhNWnWn40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPH6dPW04PWc3)，可以保证数据记录的唯一性。

2.可以大大加快[数据检索](https://www.baidu.com/s?wd=%E6%95%B0%E6%8D%AE%E6%A3%80%E7%B4%A2&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y1rAf4nvNWrHfkmhNWnWn40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPH6dPW04PWc3)速度。

3.可以加速表与表之间的连接，这一点在实现数据的参照完整性方面有特别的意义。

4.在使用ORDER BY和GROUP BY子句中进行检索数据时，可以显著减少查询中分组和排序的时间。

5.使用索引可以在检索数据的过程中使用优化隐藏器，提高系统性能。

## 索引的结构类型主要有哪些？分别描述不同的数据结构特点。

根据数据库的功能，可以在数据库设计器中创建四种索引：[唯一索引](https://www.baidu.com/s?wd=%E5%94%AF%E4%B8%80%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmHRLnW9BnHT3n1DdmHck0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWmkPHDkPWRL)、非[唯一索引](https://www.baidu.com/s?wd=%E5%94%AF%E4%B8%80%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmHRLnW9BnHT3n1DdmHck0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWmkPHDkPWRL)、主键索引和聚集索引。 尽管[唯一索引](https://www.baidu.com/s?wd=%E5%94%AF%E4%B8%80%E7%B4%A2%E5%BC%95&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmHRLnW9BnHT3n1DdmHck0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWmkPHDkPWRL)有助于定位信息，但为获得最佳性能结果，建议改用主键或唯一约束。   
 唯一索引：   
 唯一索引是不允许其中任何两行具有相同索引值的索引。 当现有数据中存在重复的键值时，大多数数据库不允许将新创建的唯一索引与表一起保存。数据库还可能防止添加将在表中创建重复键值的[新数据](https://www.baidu.com/s?wd=%E6%96%B0%E6%95%B0%E6%8D%AE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmHRLnW9BnHT3n1DdmHck0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWmkPHDkPWRL)。例如，如果在 employee 表中职员的姓 (lname) 上创建了唯一索引，则任何两个员工都不能同姓。   
 非唯一索引：   
 非唯一索引是相对唯一索引，允许其中任何两行具有相同索引值的索引。 当现有数据中存在重复的键值时，数据库是允许将新创建的索引与表一起保存。这时数据库不能防止添加将在表中创建重复键值的[新数据](https://www.baidu.com/s?wd=%E6%96%B0%E6%95%B0%E6%8D%AE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmHRLnW9BnHT3n1DdmHck0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWmkPHDkPWRL)。   
 主键索引：   
 数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。 在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。   
 聚集索引（也叫聚簇索引）：  
 在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。 如果某索引不是聚集索引，则表中行的物理顺序与键值的[逻辑顺序](https://www.baidu.com/s?wd=%E9%80%BB%E8%BE%91%E9%A1%BA%E5%BA%8F&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLmHRLnW9BnHT3n1DdmHck0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnWmkPHDkPWRL)不匹配。与非聚集索引相比，聚集索引通常提供更快的数据访问速度。

## 种检索策略是什么，分别适用于哪种场合

立即检索——

优点：对应用程序完全透明，缺点：select语句数目多。适用：类级别。

延迟检索——

优点： 由应用程序决定加载哪些对象，可以避免执行多余的select语句以及避免加载不需要访问的对象，节省内存空间，提高检索效率。

缺点： 应用程序如果要访问游离态的代理类实例，必须保证它在持久化时已经被初始化。

适用： 一对多或多对多关联。应用程序不需要立即访问或者根本不会访问的对象。

左外连接检索：

优点：对应用程序完全透明，不管对象处于持久化状态还是游离状态，应用程序都可以方便的从一个对象导航到另一个与它相关联的对象。使用了外连接，select语句数目少。

缺点：可能会加载程序不许要访问的对象。复杂的数据库表连接影响检索性能。

适用：一对一或多对一关联。应用程序需要立即访问的对象。数据库系统具有良好的表连接性能。

## 建立索引常用的规则如下

1、表的主键、外键必须有索引；

2、数据量超过300的表应该有索引；

3、经常与其他表进行连接的表，在连接字段上应该建立索引；

4、经常出现在Where子句中的字段，特别是大表的字段，应该建立索引；

5、索引应该建在选择性高的字段上；

6、索引应该建在小字段上，对于大的文本字段甚至超长字段，不要建索引；

7、复合索引的建立需要进行仔细分析；尽量考虑用单字段索引代替：

8、频繁进行数据操作的表，不要建立太多的索引；

9、删除无用的索引，避免对执行计划造成负面影响；

13.union和union all的区别

union在进行表求并集后会去掉重复的元素，所以会对所产生的结果集进行排序运算，删除重复的记录再返回结果。

union all 则只是简单地将两个结果集合并后就返回结果。因此，如果返回的两个结果集中有重复数据，那么返回的结果集就会包含重复的数据。

在执行查询操作时，union all 要比union 快很多，所以，如果可以确认合并的两个结果集中不包含重复的数据，那么最好使用union all

## 主键与索引的区别

主键与索引的区别如下：

主键是一种约束，唯一索引是一种索引，两者在本质上是不同的。

主键创建后一定包含一个唯一性索引，唯一性索引并不一定就是主键。

唯一性索引列允许空值，而主键列不允许为空值。

主键列在创建时，已经默认不为空值 + 唯一索引了。

主键可以被其他表引用为外键，而唯一索引不能。

一个表最多只能创建一个主键，但可以创建多个唯一索引。

## 锁机制：

①sql锁：

**悲观锁(Pessimistic Lock), 顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会block直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。**

**乐观锁(Optimistic Lock), 顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库如果提供类似于write\_condition机制的其实都是提供的乐观锁。**

**两种锁各有优缺点，不可认为一种好于另一种，像乐观锁适用于写比较少的情况下，即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。但如果经常产生冲突，上层应用会不断的进行retry，这样反倒是降低了性能，所以这种情况下用悲观锁就比较合适。**

**悲观锁**：就是考虑问题很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁；

实现：sql语句后边加上for update

例子：Select id,nam from biao for update

**乐观锁**:就是考虑问题很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁

实现：在表里边加一个vesion

例子：

Select max(nub) ,version from biao

Update biao set nub=nub+1,version=vsersion+1 where id=id and version =version

②方法锁：

方法锁主要包括：synchronized(c愧 ruai 死他)锁和lock锁

区别：

总结来说，Lock和synchronized有以下几点不同：

1. Lock是一个接口，而synchronized是Java中的关键字，synchronized是内置的语言实现；
2. synchronized在发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生；而Lock在发生异常时，如果没有主动通过unLock()去释放锁，则很可能造成死锁现象，因此使用Lock时需要在finally块中释放锁；
3. Lock可以让等待锁的线程响应中断，而synchronized却不行，使用synchronized时，等待的线程会一直等待下去，不能够响应中断；
4. 通过Lock可以知道有没有成功获取锁，而synchronized却无法办到。
5. Lock可以提高多个线程进行读操作的效率。

1）Lock是一个接口，而synchronized是Java中的关键字;

2）**synchronized**当一个线程获取了对应的锁，并执行该代码块时，其他线程便只能一直等待，等待获取锁的线程释放锁,不能够响应中断,（释放：1执行完线程自动释放2发生异常jvm让线程释放）（（比如调用sleep方法）），这样的好处是不会导致死锁现象发生。

**Lock**锁，可以不让等待的线程一直无期限地等待下去，比如只等待一定的时间或者响应中断。

但Lock在发生异常时，如果没有主动通过unLock()去释放锁，则很可能造成死锁现象，因此使用Lock时必须在try{}catch{}块中进行，需要在finally块中释放锁；

3）**通过Lock可以知道有没有成功获取锁，而synchronized却无法办到**。

**在性能上来说，如果竞争资源不激烈，两者的性能是差不多的，而当竞争资源非常激烈时（即有大量线程同时竞争），此时Lock的性能要远远优于synchronized。所以说，在具体使用时要根据适当情况选择。**

**在并发量比较小的情况下，使用synchronized是个不错的选择，但是在并发量比较高的情况下，其性能下降很严重，此时ReentrantLock（**可重入锁，唯一实现了Lock接口的类**）是个不错的方案。**

## 代码中事务是如何控制的？

java中的事务主要有两种，JDBC事务（本地事物）和JTA(Java Transaction API)事务（分布式事物）；事务有四大特性ACID原子性、一致性、隔离性和持久性。框架中，我们一般把事物交给spring来管理。spring配置事务的方式一般有两种，一个是声明式事务，一个是注解式事务。注解事务，比较简单灵活，在spring配置文件中配置一个<tx:annotation-driven> 饿里忒甚的注解，然后在需要的方法上加@Transactional注解就可以了。声明式事务，切点一般是扫描service层实现类，通过方法名匹配配置传播特性，决定哪些方法上加事务，哪些不需要事物。事务主要有五大隔离级别和7种传播特性；五大隔离级别由低到高：主要控制是否出现脏读，不可重复读和幻觉读；7种传播特性主要决定是新建事务，还是取当前事务；

脏读:

指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。因为这个数据是还没有提交的数据， 那么另外一个事务读到的这个数据是脏数据，依据脏数据所做的操作可能是不正确的。

**不可重复读:**

指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。

**幻觉读:**

指当事务不是独立执行时发生的一种现象，例如第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户。

事物主要是为了保证一次操作里边涉及到多次增删改时，保证他们可以同步提交或回滚。他有四大特性ACID（原子性、一致性、隔离性、持久性）

1. 原子性（Atomicity）:事务是数据库逻辑工作单元，事务中包含的操作要么都执行成功，要么都执行失败。
2. 一致性（Consistency）：事务执行的结果必须是使数据库数据从一个一致性状态变到另外一种一致性状态。当事务执行成功后就说数据库处于一致性状态。如果在执行过程中发生错误，这些未完成事务对数据库所做的修改有一部分已写入物理数据库，这是数据库就处于不一致状态。
3. 隔离性（Isolation）：一个事务的执行过程中不能影响到其他事务的执行，即一个事务内部的操作及使用的数据对其他事务是隔离的，并发执行各个事务之间无不干扰。
4. 持续性（Durability）：即一个事务执一旦提交，它对数据库数据的改变是永久性的。之后的其它操作不应该对其执行结果有任何影响。

我们框架中一般都通过Spring的AOP去管理事物；常用两种有方式：声明式事物和注解式事物；

声明式事物就是通过在配置里边配置他的事物传播特性和切入点的方式实现；切入点我们一般扫描所有的service层的方法做为切入点；事物传播特性一般通过方法名称判断去使用只读事物还是可增删改事物；

注解式事物：在Spring配置文件里边开启以后，在需要事物增删改事物的方法上加上@Transactional注解就可以了；

**事务的隔离级别**也分为四种，四个级别可以逐个解决脏读、不可重复读、幻读这几类问题

**7种事务传播行为**就是多个事务方法相互调用时，事务如何在这些方法间传播

## 四个事物隔离级别

数据库事务的隔离级别有4种，由低到高分别为Read uncommitted 、Read committed 、Repeatable read 、Serializable 。而且，在事务的并发操作中可能会出现脏读，不可重复读，幻读。

Read uncommitted

读未提交，顾名思义，就是一个事务可以读取另一个未提交事务的数据。

Read committed

读提交，顾名思义，就是一个事务要等另一个事务提交后才能读取数据。读提交，能解决脏读问题。

Repeatable read

重复读，就是在开始读取数据（事务开启）时，不再允许修改操作。重复读可以解决不可重复读问题

Serializable 序列化

Serializable 是最高的事务隔离级别，在该级别下，事务串行化顺序执行，可以避免脏读、不可重复读与幻读。但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用。

大多数数据库默认的事务隔离级别是Read committed，比如Sql Server , Oracle。Mysql的默认隔离级别是Repeatable read。

## 怎样进行代码优化

代码结构层次的优化(目的:更加方便代码的维护--可维护性，可读性)

1. 代码注释(代码规范)
2. 工具类的封装(方便代码的维护，使代码结构更加清晰不臃肿，保证团队里代码 质量一致性)
3. 公共部分的提取代码性能的优化(目的:使程序的性能最优化)

①使用一些性能比较高的类(bufferInputStream)

②缓冲区块的大小(4k或者8k)

③公共部分的提取

④通常要用stringbuffer替代string加号拼接

## Tomcat怎么优化

1、去掉对web.xml的监视，把jsp提前编辑成Servlet。有富余物理内存的情况，加大tomcat使用的jvm的内存

2、利用缓存和压缩

3、采用集群

　 4. 优化tomcat参数

这里以tomcat7的参数配置为例，需要修改conf/server.xml文件，主要是优化连接配置，关闭客户端dns查询

## servlet的生命周期

Servlet 的生命周期始于将它装入Web服务器的内存时，并在终止或重新装入Servlet时结束。包括加载和实例化、初始化、处理请求以及服务结束。这个生存期由javax.servlet.Servlet接口的init,service和destroy方法表达。

描述servlet的生命周期，谁来控制servlet的生命周期，servlet生命周期在J2EE规范中是如何来定义的服务器实例化一个servlet对象，容器运行其init方法，请求到达时运行其service方法，service方法根据请求的类型调用（doGet，doPost）方法，当服务器决定将实例销毁时调用其destory方法

## servlet线程安全描述

servlet是单列的,对于所有请求都使用一个实例,所以如果有全局变量被多线程使用的时候,就会出现线程安全问题.

解决这个问题有三种方案:

1.实现singleThreadModel接口,这样对于每次请求都会创建一个新的servlet实列,这样就会消耗服务端内存,降低性能,但是这个接口已经过时,不推荐使用.

2.可以通过加锁(synchronized关键字)来避免线程安全问题.这个时候虽然还是单列,但是对于多线程的访问,每次只能有一个请求进行方法体内执行,只有执行完毕后,其他线程才允许访问,降低吞吐量.

3.避免使用全局变量,使用局部变量可以避免线程安全问题,强烈推荐使用此方法来解决

## 拦截器和过滤器有什么区别？

Filter也称之为过滤器，它是Servlet技术中最实用的技术，Web开发人员通过Filter技术，对web服务器管理的所有web资源：例如Jsp, Servlet, 静态图片文件或静态 html 文件等进行拦截，从而实现一些特殊的功能。例如实现URL级别的权限访问控制、过滤敏感词汇、压缩响应信息等一些高级功能。它主要用于对用户请求进行预处理，也可以对HttpServletResponse进行后处理。

java里的拦截器是动态拦截Action调用的对象。它提供了一种机制可以使开发者可以定义在一个action执行的前后执行的代码，也可以在一个action执行前阻止其执行，同时也提供了一种可以提取action中可重用部分的方式。在AOP（Aspect-Oriented Programming）中拦截器用于在某个方法或字段被访问之前，进行拦截然后在之前或之后加入某些操作。

拦截器与过滤器的区别：

1、拦截器是基于java的反射机制的，而过滤器是基于函数回调。

2、拦截器不依赖与servlet容器，过滤器依赖与servlet容器。

3、拦截器只能对action请求起作用，而过滤器则可以对几乎所有的请求起作用。

4、拦截器可以访问action上下文、值栈里的对象，而过滤器不能访问。

5、在action的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次

## Session的save()、update()、saveOrUpdate()方法分别是做什么的？有什么区别

saveorupdate()如果传入的对象在数据库中有就做update操作，如果没有就做save操作。

save()在数据库中生成一条记录，如果数据库中有，会报错说有重复的记录。

update()就是更新数据库中的记录

## Ajax有什么作用，可以做什么?

话术:ajax其实就是异步请求，需要页面无刷新（局部刷新）异步交换数据时，各种操作都可以通过ajax来做，比如延迟查询某些数据，验证用户名，手机号是否被占用等等...比如比较典型的我们做过的项目中，商品详情界面中，商品基本信息是通过页面跳转获取到的，为了提高用户体验，商品描述和规格就是通过ajax进行请求获取到的

# Javaweb框架

## springmvc工作原理：

tomcat启动时，也是先加载web.xml，找到spring mvc的前端总控制器DispatcherServlet,并且通过DispatcherServlet来加载相关的配置文件信息。

当浏览器发出一个请求之后，首先找到DispatcherServlet，通过这个控制器和代码中的requestMapping注解找到对应的controller中的方法，参数直接通过方法参数接收就可以，然后调用service、dao操作数据库，返回数据。如果页面跳转，controller的方法可以返回ModelAndView和string类型，再通过spring-mvc配置文件中的视图解析器找到对应的页面。

数据可以用request或ModelAndView返回到页面。如果是ajax请求，可以直接返回List或对象，加上responseBody注解，进行返回。

## Spring MVC 的运行原理

Spring MVC的执行流程就是一次请求和响应的过程：

1. 首先，用户向服务器发送请求，请求被前端控制器 DispatherServlet 截获，DispatherServlet 对 url 进行解析；
2. 并获得Controller中的 配置信息，根据请求的地址和Controller中的配置信息找到对应的Controller中的方法；
3. 调用方法前，会自动将参数填充进来，如果需要的话也会做必要和格式转换，例如将json转换成对象，将表单键值对转换成对象；
4. 接着开始执行控制层的对应的方法；
5. 控制层的方法执行完成后，Controller 向 DispatcherServlet 返回一个 ModelAndView 对象。ModelAndView中包含视图名称和模型数据；
6. 然后SpringMVC 会根据配置文件中的视图解析器找到相应的视图，并结合Model进行渲染；

7、将视图渲染的结果返回给客户端。

## SpringMVC中都有哪些配置

视图解析器、放行静态资源、扫描controller、文件上传下载、json类型转换器、自定义拦截器、全局异常处理

## Spring面试题

内容比较多，答案自行补充一下

Spring原理：

Spring的IOC：

Spring的AOP：

Spring的事务传播特性、隔离级别

Spring生命周期

## spring中常用的设计模式（重点）

第一种：简单工厂 又叫做静态工厂方法（StaticFactory Method）模式，但不属于23种GOF设计模式之一。 简单工厂模式的实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类。 spring中的BeanFactory就是简单工厂模式的体现，根据传入一个唯一的标识来获得bean对象，但是否是在传入参数后创建还是传入参数前创建这个要根据具体情况来定。如下配置，就是在 HelloItxxz 类中创建一个 itxxzBean。 Hello! 这是singletonBean!**value**>Hello! 这是itxxzBean! value>

第二种：工厂方法（Factory Method） 通常由应用程序直接使用new创建新的对象，为了将对象的创建和使用相分离，采用工厂模式,即应用程序将对象的创建及初始化职责交给工厂对象。 一般情况下,应用程序有自己的工厂对象来创建bean.如果将应用程序自己的工厂对象交给Spring管理,那么Spring管理的就不是普通的bean,而是工厂Bean。

第三种：单例模式（Singleton） 保证一个类仅有一个实例，，即提供了全局的访问点BeanFactory。

第四种：适配器（Adapter） 在Spring的Aop中，使用的Advice（通知）来增强被代理类的功能。Spring实现这一AOP功能的原理就使用代理模式（1、JDK动态代理。2、CGLib字节码生成技术代理。）对类进行方法级别的切面增强，即，生成被代理类的代理类， 并在代理类的方法前，设置拦截器，通过执行拦截器重的内容增强了代理方法的功能，实现的面向切面编程。 Adapter类接口：。 第六种：代理（Proxy） 为其他对象提供一种代理以控制对这个对象的访问。 从结构上来看和Decorator模式类似，但Proxy是控制，更像是一种对功能的限制，而Decorator是增加职责。 spring的Proxy模式在aop中有体现，比如JdkDynamicAopProxy和Cglib2AopProxy。

第七种：观察者（Observer） 定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。 spring中Observer模式常用的地方是listener的实现。如ApplicationListener。

## spring的事务怎么进行管理的

管理事务的方式有5中,我们一般就是通过AOP来管理事务的,就是把代码看成是一个纵向的有序的,然后通过Aop去管理事务,就好比增删该的时候需要开启一个事务我们给他配置一个required,required就是有事务就执行事务,没有就给他创建一个事务,然后查的时候不需要开启事务,就给他配置一个supports ,supports就是说它有事务就执行事务,没有的话就不执行。

## spring框架的特点

答：非侵入式、容器、IOC、依赖注入、AOP

## Spring说一下？AOP具体是什么？

AOP面向切面编程将程序中的交叉业务逻辑（比如安全，日志，事务），封装成一个切面，然后注入到目标业务逻辑中去。比如：很多方法都可能会抛异常，你要记录这个异常到日志中去，可以写个拦截器，在这个类中记录日志，在spring.xml中配置一个记录这些日志的方法的拦截器，在这个方法执行后调用这个拦截器，记录日志。这样就不用每次抛异常都要手动记录日志。spring的事务管理用到的就是AOP这样也可以提高程序的内聚性。

## 解释下Spring里IOC和AOP?

IoC,叫做控制反转,所有对象实例化的过程转交给Spring IoC组件,不用程序员手动new对象.目前主流编程语言都是面向对象语言,经常处理对象和对象之间的关系.引入Spring IoC(我们也可以叫DI)可以把我们的关注点从对象之间关系释放出来,我们只关注业务逻辑.IoC实际就**解析XML**或注解后通过**反射机制**加载对应的类,然后进行实例化或**注入**(**面试题:有几种注入方式**).里面常用的设计模式包含:单例,**工厂**等.

AOP是面向切面编程.把传统程序执行过程看成纵向执行过程,每个方法当作一个点.基于这些点可以进行增强处理.形成了横向的切面,包含了原有方法和增强方法.不改变原有代码结构,添加额外功能.常用AOP功能有**事务管理([面试题:Spring如何进行事务管理,什么是事务]),权限处理(面试题:常见权限解决方案)**,**日志(面试题:项目部署后如果监听项目执行状态.)**打印等.其实Spring的AOP就是对**代理设计模式的**一个实现.

## 介绍一下Mybatis框架

Mybatis的前生是ibatis，最后升级版本后名称叫mybatis。mybatis是以纯sql操作数据，是一个持久层框架，比Hibernate灵活，但移植性差。配置文件转换为流后给交SqlSessionFactoryBuilder就会产生SqlSessionFactory,但实际上底层是这样实现的. 根据打断点进源码观察到实际上把配置文件流交由XMLConfigBuilder进行解析,解析后把所有配置信息封装到了Configuration中,在把Configuration传递给DefaultSqlSessionFactory,并实例化这个类. 在源码可以看出DefaultSqlSessionFactory是SqlSessionFactory的实现类,所以这个时候就产生了SqlSessionFactory接口的实例. 剩下的就是平时在开发中我们总写的代码. 需要openSession,产生SqlSession示例,根据SqlSession实例执行事务.最后要提交事务和关闭SqlSession 通常都不关闭SqlSessionFactory,在大点的项目中,都需要用到二级缓存,所以不关闭，配置pojo的别名、配置映射文件的位置、配置 sqlSessionFactory、在Mapper扫描器中配置dao接口。

## mybatis缓存？

mybatis一级缓存是SqlSession级别的缓存，默认支持一级缓存，不需要在配置文件去配置。

mybaits的二级缓存是mapper范围级别，，除了在SqlMapConfig.xml设置二级缓存的总开关<settingname='cacheEnabled'value='true'/>，还要在具体的mapper.xml中开启二级缓存：

<mappernamespace='cn.hpu.mybatis.mapper.UserMapper'>

<!-- 开启本mappernamespace下的二级缓存 -->

<cache></cache>

useCache配置禁用二级缓存（默认情况是true，即该sql使用二级缓存。）

<selectid='findOrderListResultMap' resultMap='ordersUserMap' useCache='false'>

mybatis刷新缓存（就是清空缓存flushCache='true' 属性，默认情况下为true即刷新缓存，如果改成false则不会刷新）

<insertid='insertUser' parameterType='cn.itcast.mybatis.po.User' flushCache='true'>

## Spring MVC 和 Struts2的区别、优缺点？

1. Spring MVC 开发效率和性能要优于Struts2。
2. Spring MVC继承了 Ajax，使用 @ResponseBody 可以获取json数据，使用@RequestBody可以在请求中发送json数据。
3. Struts中使用Ajax 比较麻烦。
4. Spring MVC 通过前端控制器 DispatcherServlet 完成请求的分发。入口是一个Servlet
5. Struts2 通过 FilterDispatcher（2.1.3以前）或 StrutsPrepareAndExecuteFilter（2.1.3和以后） 完成请求的分发。入口是一个 Filter。
6. Spring MVC 是基于方法级别传递数据。单实例情况下不存在线程安全的问题。因为Spring MVC的参数是在控制器的方法中传递的。
7. Struts2是基于类级别传输数据。Struts2单实例情况下有线程并发问题，多实例情况下没有线程并发问题，但是效率又很低。也可以在单实例情况下使用ActionContext类传递数据，但是比较麻烦。

## 简述orm 工作原理

ORM的全称是Object Relational Mapping，即对象关系映射。它的实现思想就是将关系数据库中表的数据映射成为对象，以对象的形式展现，这样开发人员就可以把对数据库的操作转化为对这些对象的操作。因此它的目的是为了方便开发人员以面向对象的思想来实现对数据库的操作。

## 说一下orm与jdbc的区别

jdbc只是一个java操作数据库的规范接口而已 orm不过是一种思想，对象关系映射。 ORM：是对象关系模型，如hibernate,让你以面向对象的方式去编程。封装了JDBC. JDBC：是从底层访问数据库服务器。一般银行，金融行业为了安全起见，直接用JDBC访问

## spring\_boot和spring\_cloud

|  |
| --- |
| spring boot 我理解就是把 spring，spring-mvc，spring-data-jpa 等等的一些常用的常用的基础框架组合起来，提供默认的可插拔的设计配置。spring boot对spring的配置进行简化，几乎零配置。同时对spring 需要的jar 也进行了整合，解决jar冲突的问题。可以理解为springMvc的升级版。提供了spring-boot-starter-parent的整合工具包。集成了他以后，项目中大部分jar包就不需要再引用了。内置了Tomcat插件，可以直接用main方法运行；  spring\_cloud是基于spring\_boot的微服务分布式架构，他主要包括：   1. 服务注册中心：Eureka Server 2. 服务注册者客户端：Eureka Client 3. 服务消费端：负载均衡客户端：ribbon和feign 4. 断路器Hystrix 5. 路由网关zuul（Zuul的主要功能是路由转发和过滤器，zuul默认和Ribbon结合实现了负载均衡的功能。） |

## maven怎么添加jar包，如果失败怎么办

可以在本地仓库添加，在pom.xml文件中写上需要引入的jar包，或者手动添加 install

## 数据库比较

|  |
| --- |
| 1. 首先主要是关系型数据库、nosql数据库和缓存数据库的区别；我觉的：   **关系型数据库（mysql、Oracle）**就想当于一个储物柜，里边分了各种小格子，把各种数据有序的保存在里边，查询时非常方便；有比较完善的sql语句；   * mysql免费开源，自动提交事务，是基于表级别的一个事务，而Oracle不自动提交，是基于行级别的事务；mysql安装包小，使用简单，Oracle安装包较大，还会收费，更适合于大数据的处理、并发量高； * mysql用limit进行分页，第一个参数表示启始数，第二个参数表示要查询的条数。 * oracle用伪列rownum来进行分页，因为伪列只能用小于，不能用大于，所以需要一个两层的嵌套查询。 * mysql主键可以自增。Oracle必须用序列（sequence）；   字段类型有些区别；   * mysql一般都是先有用户，然后创建库、创建表。而Oracle一般是，先创建数据库，然后创建临时表空间、物理表空间，再创建用户，创建表。   **nosql数据库（mongdb）**就相当于一个仓库，比如把衣服放在一堆，书本放在一堆，这样他的存储效率肯定很高，但是查找时，不方便精确查找。所以nosql更适合于数据量比较大，但是数据之间关系不是特别强，或者不是特别重要的数据；比如：电商里边的评论数据；  他的collection相当于数据库，document相当于表，document中数据主要以json格式存储；  **缓存数据库（redis、memcached）**：缓存数据库由于是保存在内存中的，所以他的存储效率非常高；但是他不适合持久化存储数据，尽管redis支持持久化，但是他也仅仅是在一定程度上保证了数据的安全性，还是有很多情况会造成数据丢失的，比如redis穿透、雪崩等。  所以他仅仅适合做为数据库和应用程序直接的中间层使用，不能作为真正的存储数据库使用。即使使用，也只能是一些不是特别重要的数据，比如购物车，在B2C的项目中，用户购物车一般都是临时存储，如果放入购物车长时间不提交，那么用户再购买的可能性很小了，所以即使丢了，也不会造成太多的损失。以这个，来换取项目的运行效率非常划算； |

## Restful

Restful（**Representational State Transfer**表现层状态转化），是一种软件架构设计的设计风格而不是标准，只是提供了一组设计原则和约束条件。它结构清晰、符合标准、易于理解、扩展方便，所以正得到越来越多网站的采用。他的理念是把每一个URI代表一种资源，浏览器可以通过http的四个关键词对这种资源进行操作。形成四种不同的请求（get、post、put、delete）；但java里边一般只支持get和post，如果想要支持put和delete，需要在web.xml中加一个过滤器，还需要前台传一个\_method的参数，告诉后台是put还是delete请求。个人感觉也不是很方便。我们用restful分格，一种是写rest分格的api，一种是写rest风格的url。rest风格的url就是用路径传参，代替传统的？号传参。用注解@PathVariable来接收参数。rest风格的api，就是把同一个资源的请求，用一个rui表示，用四个关键字来区分不同的请求。使得接口调用人员不会对请求的资源地址产生混淆和大量的检查方法名的麻烦，形成一个统一的接口。另外，Rest风格的接口方法，一定是直接响应客户数据的，而不是跳转页面的，在springmvc中，一般我们把所有需要用到ReponseBody注解响应数据的方法，成为rest方法。他还提供了一个@RestController的注解，用来代替@Controller+@ReponseBody;

## 根据项目选择合适的web服务？

java中的web服务器，主要是与JSP/Servlet兼容的Web服务器，比较常用的有Tomcat、jetty、JBoss、WebSphere 和 WebLogic 等；目前最流行的还是tomcat，适合于中小型项目，大型项目也可以借助其他其他工具用集群方式部署；jetty是相对于tomcat的一个更轻量级的框架，在小型项目或者云台托管的项目更高效简单。而JBoss、WebSphere、WebLogic是主要运用于大型的企业级web服务，这三个里边只有Jboss是开源免费的，WebSphere和WebLogic都是商业web服务，一套都有上万元以上。他们功能强大，性能稳定，支持分布式部署，有很好的售后支持。WebLogic应该是商业版本里边目前市场占有率最高的服务器，配置简单功能强大。WebSphere是IBM的产品，功能强大，而且有IBM的开发工具相配套，开发Web程序十分方便

## Java操作文件

**Java中，一般用poi来操作Excel，用itext来操作word或PDF；**

导出Excel需要注意的地方主要有：

他的2003版本和2007版本用的类是不一样的。

2003 HSSFWorkbook

2007 XSSFWorkbook

并且poi3.8版本后，提供了一个超大数据量导出的类；SXSSFWorkbook

导出的步骤主要就是：

1. 创建工作簿
2. 创建sheet
3. 创建行
4. 创建列
5. 把工作簿写入到输出流

如果是导出到固定地方，那么直接new一个输出流，如果让用户自己选择保存路径，就通过response获取输出流；

# Javaweb拓展

## Reids

Redis是一个继memcached后的又一个第三方缓存数据库，他比memcached强大很多，支持更多的数据类它作为数据库和应用程序直接的中间层来使用，已减轻数据库压力，提高运行效率。

我们项目中很多地方用到了redis；比如商品的三级分类、省市县、关于我们、联系我们、友情链接，常见问题等经常查询但是不经常改变的数据.redis还可以在tomcat集群里边实现session的共享。由于他的单线程的，所以在电商平台里边也经常用他做“防止超卖”, 生成规则的商品编号等。还有就是购物车也用到了redis

代码中，我们一般都通过spring整合redis官方提供的jedis工具包来操作redis。可以是单机版，也可以是集群。Redis本身就支持集群操作redis\_cluster，另外redis还支持主从复杂，还有他独特的哨兵模式，在主服务器宕机时，从服务器可以自动转换为主服务器。另外，他也有他的分片机制，就像mysql的水平分表设计，数据量大时，可以把数据存储到不同的库中，来减轻redis负担。

Redis的持久化方式主要有2种，RDB和AOF，RDB是一种快照方式，默认每隔5分钟创建一个快照副本，这种方式占用空间大，而且会丢失间隔时间5分钟之内的数据，但是他适合做备份，恢复时，可以根据需要恢复任意间隔时间点的数据。AOF是一种日志的持久化记录方式，每秒钟，都把redis中新增的数据记录到日志文件中，这种方式只有一个文件，占用空间少，最多丢失1秒内的数据。相对比较好，但是如果想要恢复5分钟或10分钟前某个时间点的数据，就不行了。所以实际项目中，我们一般会两种方式同时使用。如果搭建集群的话，还可以通过集群互相备份数据，只要集群不同时挂掉，单个redis就可以从集群中的其他服务器获取到最新数据。

还有就是，由于redis不是很稳定，有时候会发生“穿透”和“雪崩”； redis，都是按照key去缓存查询，如果不存在对应的value，就应该去数据库查找。如果key对应的value是一定不存在的，并且对该key并发请求量很大，就会对后端系统造成很大的压力。这就叫做缓存穿透。当缓存服务器重启或者大量缓存集中在某一个时间段失效，这样在失效的时候，也会给后端系统(比如DB)带来很大压力，这就叫“缓存雪崩”。解决办法是，对查询结果为空的情况也进行缓存，并且给缓存设置不同的有效期。当然redis容灾的最有效的方法还是搭建集群。在缓存失效后，通过加锁或者队列来控制读数据库写缓存的线程数量。比如对某个key只允许一个线程查询数据和写缓存，其他线程等待。：不同的key，设置不同的过期时间，让缓存失效的时间点尽量均匀。

快照（RDB）方式，默认方式，文件以二进制方式保存到RDB文件。

文件追加（AOF）方式，文件以协议文本的方式write到AOF文件。

作用，重启后的数据恢复。当两种方式都启用时，redis会优先载入AOF文件，因为在通常情况下AOF文件保存的数据集要比RDB文件保存的数据集要完整。注意优先是指载入AOF文件就不再加载RDB了，不管AOF是不是空文件。

## Redis 分片

分片(partitioning)就是将你的数据拆分到多个 Redis 实例的过程，这样每个实例将只包含所有键的子集；

Redis 的分片承担着两个主要目标：

允许使用很多电脑的内存总和来支持更大的数据库。没有分片，你就被局限于单机能支持的内存容量。

允许伸缩计算能力到多核或多服务器，伸缩网络带宽到多服务器或多网络适配器。

## redis集群里存数据是怎么存的?

Hash槽0~16383存数据的时候,redis会根据存进来的key值,计算一个二进制码,根据这个二进制码存放到对应hash槽里.

取的时候,也是根据你的key值,计算一个二进制码,根据二进制码去对应的hash槽里取数据.

存数据: set age 100

取数据 : get age

Key是一样的,那key的二进制码也是一样的.所以就能找到对应的hash槽.

## Dubbo

dubbo是一个分布式架构的服务框架，一般结合maven的模块试开发使用。传统的单架构项目，不方便维护和升级；通过maven的模块式开发，就可以把一个单架构的工程，拆封成一个一个的小模块，包括（jar和war）；jar包可以被war包直接应用。war包可以分布式部署，但是这样不同的war之间的交互就成为了问题。我们之前的做法是，两个war之间进行交互时，一般采用webservice或httpclient。但是在这种分布式架构中，多个war之间的交互会特别频繁，如果用webservice或httpclient架构师很多时候自己都搞不清楚他们之间的依赖关系了。而dubbo解决了这个问题，有了dubbo架构以后，多个war之间不在直接进行交互，而是统一和dubbo的注册中心zookeeper进行交互。无论是发布服务还是调用服务，都通过dubbo实现，并且提供了web页面，来监控和管理各个接口直接的调用。

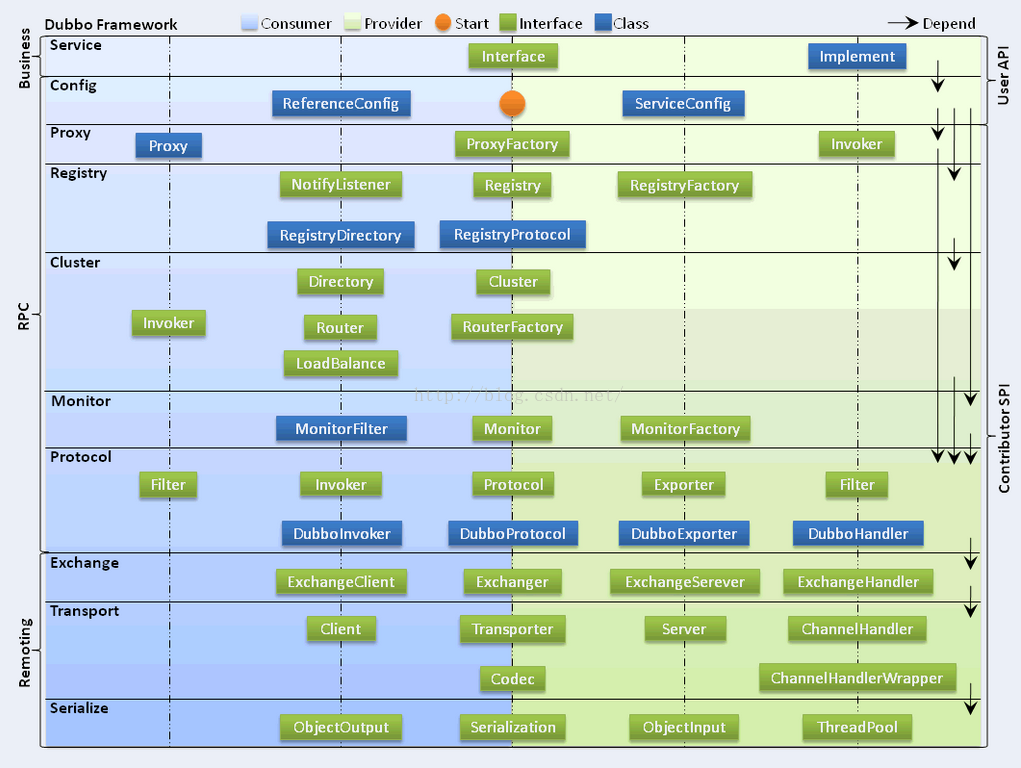
使用时首先搭建dubbo的注册中心zookeeper，然后下载dubbo的服务治理工程dubbo\_admin.war，解压修改他的配置文件dubbo.properties，设置用户名、密码和注册中心zookeeper的ip、端口。部署到Tomcat的webapp中，启动Tomcat，浏览器用ip+端口访问，就可以看到dubbo\_admin的web页面了。接着在我们要发布或者调用服务的项目中，导入dubbo提供的相关jar包，加载配置文件，把要发布的服务注册到zookeeper里边。或者从zookeeper里边调用服务就可以了。dubbo\_admin的web页面中，可以监控和管理所有的服务提供者和调用者。。

在dubbo架构中，service接口和实体对象时各个工程共享的，service实现类是发布服务的工程独有的。只需要发布服务的工程把他的接口注册到zookeeper中，其他工程就可以从zookeeper中进行调用了。

## Dubbo的好处

把项目拆分成各个独立的小工程，通过接口调用方式，互相交互数据，可以单独进行部署和升级，这样就减少了他们直接的耦合性和代码的复用性。还可以针对不同模块采用不同的部署策略，比如订单模块并发比较高，所以可以把订单模块这个war包，单独部署多套，都注册到同一个zookeeper中。当客户端调用时，zookeeper会帮忙进行负载处理。

## Dubbo框架设计一共划分了10个层



Dubbo的架构设计共分为了10层。最上面的Service层是留给实际想要使用Dubbo开发分布式服务的开发者实现业务逻辑的接口层。图中左边淡蓝色背景为服务消费方使用的接口，右边淡绿色背景为服务提供方使用的接口，位于中轴线的为双方都要用到的接口。结合Dubbo官网，总结如下：

1.服务接口层：该层与实际业务逻辑有关，根据服务消费方和服务提供方的业务设计，实现对应的接口。

2.配置层：对外配置接口，以ServiceConfig和ReferenceConfig为中心，可以直接new配置类，也可以根据spring解析配置生成配置类。

3.服务注册层：封装服务地址的注册和发现，以服务URL为中心，扩展接口为RegistryFactory、Registry、RegistryService，可能没有服务注册中心，此时服务提供方直接暴露服务。

4.服务代理层：服务接口通明代理，生成服务的客户端Stub和服务端Skeleton，以ServiceProxy为中心，扩展接口ProxyFactory。

5.集群层：封装多个提供者的路由和负载均衡，并桥接注册中心，以Invoker为中心，扩展接口为Cluster、Directory、Router和LoadBalance，将多个服务提供方组合为 一个服务提供方，实现对服务消费通明。只需要与一个服务提供方进行交互。

6.监控层：RPC调用时间和次数监控，以Statistics为中心，扩展接口MonitorFactory、Monitor和MonitorService。

7.远程调用层：封装RPC调用，以Invocation和Result为中心，扩展接口为Protocol、Invoker和Exporter。Protocol是服务域，它是Invoker暴露和引用的主功能入口，它负责Invoker的生命周期管理。Invoker是实体域，它是Dubbo的核心模型，其他模型都是向它靠拢，或转换成它，它代表一个可执行体，可向它发起Invoker调用，它有可能是一个本地实现，也有可能是一个远程实现，也有可能是一个集群实现。

8.信息交换层：封装请求响应模式，同步转异步，以Request和Response为中心，扩展接口为Exchanger和ExchangeChannel,ExchangeClient和ExchangeServer。

9.网络传输层：抽象和mina和netty为统一接口，以Message为中心，扩展接口为Channel、Transporter、Client、Server和Codec。

10.数据序列化层：可复用的一些工具，扩展接口为Serialization、ObjectInput,ObejctOutput和ThreadPool。

## Zookeeper

Zookeeper主要功能是为分布式系统提供一致性协调(Coordination)服务，主要有两大功能：统一配置管理和集群负载均衡，

**统一配置管理**：分布式系统都有好多机器，这些机器上边的配置是一致的，正常情况下需要修改好一个以后，然后scp到其他服务器，每次修改，就要把所有的配置都修改一遍，Zookeeper提供了这样的一种服务：一种集中管理配置的方法，我们在这个集中的地方修改了配置，所有对这个配置感兴趣的都可以获得变更。这样就省去手动拷贝配置了，还保证了可靠和一致性。

**集群负载均衡**：在分布式的集群中，经常会由于各种原因，比如硬件故障，软件故障，网络问题，有些节点会进进出出。有新的节点加入进来，也有老的节点退出集群。这个时候，集群中有些机器（比如Master节点）需要感知到这种变化，然后根据这种变化做出对应的决策。Zookeeper会对通过选举机制，选举一个主节点作为管理者。选举一般都是奇数台，否则会选举失败。并且集群中，有半数以上宕机，则会认为整个集群挂掉。

**我们项目中用到**Zookeeper**的地方主要有**：solr集群、redis集群和dubbo注册中心。

## Freemarker

freemarker是一个模板化引擎语言，传统项目中，后台页面基本上都是list列表，页面都是类似的，只是展示的数据不一样。这样我们就可以把他的样式做成一个freemarker模板，然后传入数据就可以展示不同的页面。他的模板一般是以.ftl结尾的。如果使用这个呢，可以使我们开发人员不需要太关注前台。但是对于很多门户或商城类项目，每个页面都是不一样，所以用他也不是很方便。另外，由于可以把freemarker模板直接转换成html、jsp、java、xml、word等各种文档。所以我们经常使用他做代码生成、word生成或者首页静态化等。我这里就用到了首页静态化的功能。由于任何用户访问时，首先会访问到我们的首页，所以很多东西都希望能在首页展示，但是放的东西多了，就会加载很慢。给用户的体验度很不好。所以我们在项目启动时，直接把首页需要的数据查询出来，通过freemarker模板生成静态的html页面，之后用户访问时，都去访问这个静态页面，这样就不需要频繁访问数据库，减轻了数据库压力，提高了用户体验度。但是缺点是，数据库数据数据变换了以后，数据无法实时更新。我们一般通过定时器的方式，每天凌晨重新生成。

还有就是热销商品的商品详情页面也做了静态化处理，首页我们是通过定时器每天凌晨一点，去重新生成；商品详情我们是在商品信息修改以后，给定时器表（id,业务id、type，remark）中推送一条信息，到第二天凌晨一点时，定时任务扫描表，发现有需要重新生成的页面，就调用freemarker模板，重新生成html页面，以商品id作为html页面名称，然后删除任务表中的数据。为了预防大量静态页面给服务器造成压力，我们把html页面直接生成到Nginx的静态页面服务器上。访问时，不用经过Tomcat，直接通过Nginx访问。

和他类似的技术，我还知道一种volicity，但是这个技术现在基本上用的很少了，主要是他对jsp的兼容不是很好。

freemarker主要语法用${}输出变量，$#等要用原样输出符${r"#"}，空值会抛异常，<#list 做循环，循环下标用变量\_index

${book.name?if\_exists } //用于判断如果存在,就输出这个值   
${book.name?default(‘xxx’)}//默认值xxx   
${book.name!"xxx"}//默认值xxx   
${book.date?string('yyyy-MM-dd')} //日期格式   
${book?string.number} 20 //三种不同的数字格式   
${book?string.currency}--<#-- $20.00 -->   
${book?string.percent}—<#-- 20% -->

用lt、lte、gt和gte来替代<、<=、>和>=

如果存在特殊字符,可以使用${r"..."}进行过滤

# App接口开发

1. 由于移动端app一般时不能直接访问数据库的，所以需要我们Java后台开发接口，供移动端去调用；