**一、JAVA基础概念**

**1.面向对象的特征有哪些？**

1）抽象：抽象就是忽略一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的方面。抽象并不打算了解全部问题，而只是选择其中的一部分，暂时不用部分细节。抽象包括两个方面，一是过程抽象，二是数据抽象。

2）封装：封装是把过程和数据包围起来，对数据的访问只能通过已定义的界面。面向对象计算始于这个基本概念，即现实世界可以被描绘成一系列完全自治、封装的对象，这些对象通过一个受保护的接口访问其他对象。

3）继承：继承是一种联结类的层次模型，并且允许和鼓励类的重用，它提供了一种明确表述共性的方法。对象的一个新类可以从现有的类中派生，这个过程称为类继承。新类继承了原始类的特性，新类称为原始类的派生类（子类），而原始类称为新类的基类（父类）。派生类可以从它的基类那里继承方法和实例变量，并且类可以修改或增加新的方法使之更适合特殊的需要。   
 4）多态性：多态性是指允许不同类的对象对同一消息作出响应。多态性包括参数化多态性和包含多态性。多态性语言具有灵活、抽象、行为共享、代码共享的优势，很好的解决了应用程序函数同名问题。

**2.char型变量中能不能存贮一个中文汉字?为什么?**

１）能存一个汉字

２）一个char是UNICODE编码，是２个字节，１６个二进制位

３）char 变量值，无论怎样运算，其值总是　大于或等于０，即：是一个无符号的16个二进制位的整数。

**3. java内存会泄漏吗？如果会，那么在什么情况下发生？**

在Java中，内存泄漏就是存在一些被分配的对象，这些对象有下面两个特点：

首先，这些对象是可达的，即在有向图中，存在通路可以与其相连；

其次，这些对象是无用的，即程序以后不会再使用这些对象。如果对象满足这两个条件，这些对象就可以判定为Java中的内存泄漏，这些对象不会被GC所回收，然而它却占用内存。

**4.加载.CLASS的顺序）（实例化过程）**

答：

1）类只有在使用New创建的时候才会被JAVA类装载器装入

2）JAVA类首次装入时，会对静态成员变量或方法进行一次初始化,但方法不被调用是不会执行的，静态成员变量和静态初始化块级别相同，非静态成员变量和非静态初始化块级别相同。

先初始化父类的静态代码--->初始化子类的静态代码-->

初始化父类的非静态代码--->初始化父类构造函数--->

初始化子类非静态代码--->初始化子类构造函数

3）创建类实例时，首先按照父子继承关系进行初始化

4）类实例创建时候，首先初始化块部分先执行，然后是构造方法；然后从

本类继承的子类的初始化块执行，最后是子类的构造方法

**5.什么是架构模式，分析模式与设计模式？**

架构是在组件彼此间和与环境间的关系，引导设计发展原则中体现的系统的基本结构。

**6.JAVA 有哪些架构模式，哪些设计模式和哪些分析模式？**

企业架构，系统架构，组织架构，信息架构，硬件架构，应用架构，基础设施架构

**7.面向对象设计的5大原则**

1.开放---封闭原则

模块的行为必须是开放的，可扩展的，扩展的时候不应该影响系统中已由模块的其他部分功能的实现代码。也就是说，对扩展是开放的，对修改是封闭的。面向接口编程或者模板方法模式都是常用的方法。

2.单一职责原则

对于某个具体的类，应该仅有一个引起它变化的原因。

3.Liskov替换原则

和开放--封闭原则关系密切，正式由于子类的可替换性，才使得使用父类的其他功能模块无需修改就可以实现扩充。

4.接口隔离

是单一职责原则应用于接口设计的自然结果：一个类对另外一个类的依赖关系应该是建立在最小接口上；使用多个专门的接口比使用单一的复合总接口要优越。

5.依赖倒置

高层模块不依赖于底层模块，两者都依赖于抽象；抽象不应该依赖于细节实现，实现细节应该依赖于抽象。通常是在两个，模块之间定义出一个抽象接口。

**8.transient 关键字用途**

transient 关键字表示在实现Serializable 接口的时候不保存该值。瞬时的

**9.JAVA内存溢出，引起发生内存溢出的原因有？**

内存溢出是指应用系统中存在无法回收的内存或使用的内存过多，最终使得程序运行要用到的内存大于虚拟机能提供的最大内存。

常见如下几种原因：

1）内存中加载的数据量过于庞大，如一次从数据库取出过多数据；

2）集合类中有对对象的引用，使用完后未清空，使得JVM不能回收；

3） 代码中存在死循环或循环产生过多重复的对象实体；

4） 使用的第三方软件中的BUG；

**10.静态变量与实例变量的区别**

类的成员变量有两种：一种是被static关键字修饰的变量，叫类变量或静态变量，一种是叫做实例变量，区别：

1. java虚拟机在加载类的过程中为静态变量分配内存，静态变量位于方法区，被类的所有实例共享。

静态变量可以通过类名直接访问，它的生命周期取决于类的生命周期，当类被加载的时候，静态变量被创建并分配内存空间，当类被卸载时，静态变量被摧毁，并释放所占有的内存。

2）类的每一个实例都有相应的实例变量，每创建一个类的实例，java虚拟机为实例变量分配一次内存，实例变量位于堆区中，实例变量的生命周期取决于实例的生命周期，当创建实例时，为实例变量背创建，并分配内存，当实例被销毁时，实例 变量被销毁，并释放所占有的内存空间。

**11. JVM加载class文件的原理机制**

在Java中，类装载器把一个类装入Java虚拟机中，要经过三个步骤来完成：装载、链接和初始化，其中链接又可以分成校验、准备、解析

装载：查找和导入类或接口的二进制数据；

链接：执行下面的校验、准备和解析步骤，其中解析步骤是可以选择的；

校验：检查导入类或接口的二进制数据的正确性；

准备：给类的静态变量分配并初始化存储空间；

解析：将符号引用转成直接引用；

初始化：激活类的静态变量,初始化Java代码和静态Java代码块

**12.JAVA对象池技术的原理和优点**

1）原理：对象池技术基本原理的核心有两点：缓存和共享，即对于那些被频繁使用的对象，在使用完后，不立即将它们释放，而是将它们缓存起来，以供后续的应用程序重复使用，从而减少创建对象和释放对象的次数，进而改善应用程序的性能。

2）优点：

a、通过对象池技术，实现对象的共享从而有效地减少了应用程序内存上的开销。

b、当需要一个对象时，就可以从池中取一个出来（如果池中没有则创建一个），则在需要重复重复创建相等变量时节省了很多时间，从而提高应用程序的性能。

c、通用程度高

**13.JAVA位运算符**

1）移位运算符：“>> 右移”；“<< 左移”；“>>> 无符号右移”

2）位逻辑运算符：& 与；| 或；~ 非（也叫做求反）；^ 异或

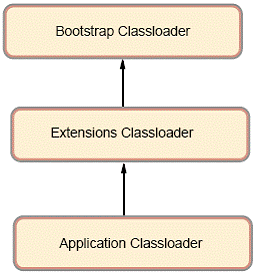
**14.类加载器**

每个JAVA程序至少拥有三个类加载器：

1）引导类加载器（Bootstrap Classloader）：它负责加载系统类（在<JAVA\_HOME>/jre/lib目录下的jar中进行加载）。是虚拟机整体中的一部分，通常是用C语言实现的。引导类加载器没有相应的ClassLoader对象。eg：String.class.getClassLoader() 返回NULL。

2）扩展类加载器(Extensions Classloader)：它负责加载扩展路径下的代码，一般位于<JAVA\_HOME>/jre/lib/ext 。

3）应用程序类加载器（Application Classloder）：它在CLASSPATH环境变量或者-classpath命令行选项设置类路径的目录里或者是JAR/ZIP文件里查找这些类



**15.SOA**

　　面向服务的体系结构（Service-Oriented Architecture，SOA）

     是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以一种统一和通用的方式进行交互。

　　这种具有中立的接口定义（没有强制绑定到特定的实现上）的特征称为服务之间的松耦合。

**16.SOAP**

简单对象访问协议 （SOAP：Simple Object Access Protocol）简单对象访问协议（SOAP）是一种轻量的、简单的、基于 XML 的协议，它被设计成在 WEB 上交换结构化的和固化的信息。 SOAP 可以和现存的许多因特网协议和格式结合使用，包括超文本传输协议（ HTTP），简单邮件传输协议（SMTP），多用途网际邮件扩充协议（MIME）。它还支持从消息系统到远程过程调用（RPC）等大量的应用程序。

**二、JAVA概念的区别**

**1.String 和StringBuffer(StringBuilder)的区别**

1）String 对象是不可变的字符序列，但可以共享。String 类可用于检查序列的单个字符、比较字符串、搜索字符串、提取子字符串、创建字符串副本并将所有字符全部转换为大写或小写。大小写映射基于 Character 类指定的 Unicode 标准版。

2）StringBuffer是线程安全的可变字符序列，可将字符串缓冲区安全地用于多个线程。

3）StringBuilder一个可变的字符序列不是线程安全的(不是同步的)，StringBuilder被设计用作StringBuffer的替代类，效率比StringBuffer快。

**2.HashMap和Hashtable的区别**。

1）HashMap是Hashtable的轻量级实现（非线程安全的实现），它们都完成了Map接口，主要区别在于HashMap允许空（null）键值（key）,由于非线程安全，效率上可能高于Hashtable。   
HashMap允许将null作为一个entry的key或者value，而Hashtable不允许。   
2）HashMap把Hashtable的contains方法去掉了，改成containsvalue和containsKey。因为contains方法容易让人引起误解。 Hashtable继承自Dictionary类，而HashMap是Java1.2引进的Map interface的一个实现。   
3）Hashtable的方法是Synchronize的，而HashMap不是，在多个线程访问Hashtable时，不需要自己为它的方法实现同步，而HashMap 就必须为之提供外同步。

3迭代器与枚举有两点不同Itreator

**4.谈谈final,finally,finalize的区别？**

**final**：java中的关键字，修饰符。

1.用于声明类，表示类不可继承，即该类不能作为父类被继承。

2.如果将变量或者方法声明为final，可以保证它们在使用中不被改变.

4.被声明final的方法只能使用，不能重写。

5.修饰基本类型，就表示这个变量被赋予的值是不可变的，即它是个常量；修饰对象，就表示这个变量被赋予的引用是不可变的。以后的引用中只能读取，不可修改。

**finally：**java的一种异常处理机制。

finally 结构使代码总会执行，而不管有无异常发生。使用 finally 可以维护对象的内部状态，并可以清理非内存资源。

**finalize**：Object中的基础方法

它是在Object类中定义的，因此所有的类都继承了它。子类覆盖finalize()方法以整理系统资源或者执行其他清理工作。finalize()方法是在垃圾收集器删除对象之前对这个对象调用的。

**5.谈谈Collection与Collections的区别**

1) Collection 是集合类的上级接口，继承与它的有Set 和List

2) Collections 是针对集合类的一个帮助类，它提供了一系列的静态方法实现对各种集合的搜索、排序、线程安全化等操作。

**6.error 和 exception有什么区别**

Error：1) 表示总是不可控制的(unchecked),经常用来表示系统错误或底层资源错误；

2) 如果可能的话，应该在系统级被捕获

Exception：1) 表示通常是程序员在设计或实现上出现的问题，可以是可控制或不可控制的；2) 可以捕获得到，应该在应用程序级别被处理。

**7.抽象类（abstract class）和接口（Interface）的区别**

abstract class和interface在Java语言中都是用来对问题领域进行分析、设计中得出的抽象概念，是不能够实例化的。区别：

1）抽象类可以定义自己的成员变量和非抽象方法；接口中，所有方法都是抽象的。

2）抽象类在JAVA语言中表示的是单一继承的关系，一个类只能继承一个抽象类。

但是一个类可以实现多个接口，属于多继承。

3）抽象类中可以定义方法的默认行为，接口是不能的。

**12、Static Nested Class 和 Inner Class的不同。**

1）Static Nested Class是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。

2）而通常的内部类需要在外部类实例化后才能实例化。

**13.sleep() 和 wait() 有什么区别?**

    1）sleep是线程类（Thread）的方法，调用sleep()导致当前线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时间后会自动恢复。调用sleep不会释放对象锁。  
 2）wait是Object类的方法，调用wait()导致当前线程等待，直到其他线程通过调用notify()或notifyAll()通知在该对象的监视器上等待的线程醒来。然后该线程重新获得监视权，才能继续执行。

**三、JAVA线程题**

**1．试述何谓线程和进程**

1）是某种程度上相互隔离的、独立的运行程序；是CPU调度资源的基本单位。

2）线程(thread),有时被称为轻量级进程(Lightweight Process，LWP)，是程序执行流的最小单元。一个标准的线程由线程ID，当前指令指针(PC)，寄存器集合和堆栈组成。另外，线程是进程中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有系统资源，只拥有一点在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。一个线程可以创建和撤消另一个线程，同一进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。

**2.线程的状态**

线程有四种状态：

1）New（新生）：当你用 new 操作符创建一个线程时，eg：new Thread(r),线程还没运行，此时线程处于新生状态。

2）Runnable(可运行)：一旦调用了start()方法，该线程就是可运行的了。注意：在任何给定的时刻，一个可运行的线程可能正在运行，也可能不是。（这就是为什么状态为“Runnable”而不是“Running”）

3）Blocked(被阻塞)

当发生以下任一情况时，线程就进入被阻塞的状态：

* 线程通过调用sleep()方法进入睡眠状态
* 线程调用一个在I/0上被阻塞的操作，即该操作在输入输出操作完成前不会返回到它的调用者；
* 线程试图得到一个锁，而该锁正被其他线程持有。
* 线程在等待某个触发条件
* 有人调用了线程的suspend方法。注意：该法已经过时

通过以下途径的一种，线程可以从被阻塞状态回到可运行状态：

* 线程被置于睡眠状态，且已经经过指定的毫秒数。
* 线程在等待I/O操作，且该操作已经完成
* 线程正在等待另一个线程持有的锁，且另一个线程已经释放对锁的所有权（也有可能等待超时，当超时发生时，线程解除阻塞）
* 线程正在等待触发条件，且另外一个线程发出了信号表明条件已经发生了变化。
* 线程已被挂起，且有人调用了它的resume方法。注意：该法已经过时

4）Dead（死亡）

有两个原因会导致线程死亡：

* 因为run()方法正常退出而自然死亡
* 因为一个未捕获的异常终止了run()方法而使得现场猝死。

**3.使用线程的好处**

1）使UI响应更快

2）利用多处理器系统，使用多个线程可以提供程序的吞吐量和性能

3）简化建模，在某些情况下使用线程使程序编写和维护起来更加简单

4）执行异步操作或后台处理

**4.线程类Thread 常用的方法**

1）void interrupt()：发送一个中断请求给一个线程，这个线程的中断状态将被设成TRUE。如果这个线程被sleep()调用阻塞，那么将抛出一个InterruptException异常。

2）static boolean interrupted()：检查当前线程（正在执行该指令的线程）是否已经被中断。注意：对于它的调用会产生副作用，它会将当前的线程中断状态设为false。

3）boolean isInterrupted()：检查一个线程是否已经被终止了。与static boolean interrupted()不同，这个调用不会改变当前线程的状态。

4）static Thread currentThread()：返回当前执行的Thread对象。

5）boolean isAlive() :判断线程是否活着，处于Runnable 和 Blocked状态返回 true；处于New 和 Dead返回false。

注意：无法确定一个活着的线程是可运行还是被阻塞的，也无法确定一个可运行的线程是否正在运行。

6）void join()：等待直到指定的线程死亡

7）void join(long millis)：等待直到指定的线程或经过指定的毫秒数

8）void setPriority(int newPriority)：设置线程的优先级。优先级必须在Thread.MIN\_PRIORITY 和 Thread.MAX\_PRIORITY(即 1 到 10 之间)。一般情况下，使用Thread.NORM\_PRIORITY优先级。

9）static void yield()：导致当前执行线程进入让步状态。如果有优先级不低于它的其他可运行线程存在，那么这些线程接下来就会被调度。

10）void setDaemon(Boolean isDaemon)：将线程标记为用户线程或守护线程。这个方法必须在线程开始执行之前调用。注意：一个守护线程唯一的作用就是为其他线程提供服务

5.线程池

如果你的程序创建了大量生存期很短的线程，那就应该使用线程池。

1）一个线程池包含大量准备运行的空闲线程，你将一个Runnable对象给线程池，线程池中的一个线程就会调用run()方法，当run()退出时线程不会死亡，而是在池中准备为下一个请求提供服务。

2）减少并发线程的数量

**四、JAVA网络编程题**

**五、Servlet 笔试题**

**2.试描述Servlet 的生命周期**

Servlet被服务器实例化后，容器运行其init方法，请求到达时运行其service方法，service方法自动派遣运行与请求对应的doXXX方法（doGet，doPost）等，当服务器决定将实例销毁的时候调用其destroy方法。

**六、JSP 笔试题**

**七、JAVA IO笔试题**

**八、J2EE 系列服务**

**九、JAVA开源框架（Spring篇）**

**1.Spring概述**

Spring是基于J2EE平台的轻量级的框架。整个框架架构在Core核心模块之上，是整个框架的基础；在该模块中包括IoC的实现和框架内部使用的各种工具类。沿着Spring树往上左序遍历，就是AOP模块。

继续上行，在Core核心模块和AOP模块的基础上，Spring提供了数据访问和事务管理的抽象和集成服务。为了简化各种J2EE 服务的使用，Spring框架提供了针对这些J2EE服务的集成服务，最后要提到Web模块，该模块中Spring提供了一套自己的Web MVC框架；Spring框架为其他的WEB框架提供了集成支持（如支持Struts、WebWork）。如下图所示：

Core

IoC容器

Framework工具类

AOP

Spring AOP

集成AspectJ

DAO

Spring JDBC

事物管理

Framework工具类

ORM

Hibernate、JPA

iBATIS、Toplink

WEB

Spring MVC

其他Web框架的集成（eg：Struts）

J2EE服务集成

JMX、JMS、JCA

JavaMail、EJB

Remoting

**图 9-1 Spring框架总体结构**

**2.轻量级和重量级区分**

主要区分于容器提供的服务多少，服务多自然性能就会下降。

1）Spring：提供了很多服务，但这些服务都不是默认打开的，应用需要某种服务，需要指明打开，如果服务少那么就属于轻量级，如果服务开多了，就属于重量级。

2）EJB： 它都是将功能默认打开的，属于重量级的

**3.什么是IoC（控制反转）和依赖注入(DI)**

1）控制反转（IOC）:应用本身不负责对象的创建与维护，依赖对象的创建及维护是由外部容器负责的。这样控制权就由应用转移到了外部容器，控制权的转移就是反转。

2）依赖注入：在运行期，由外部容器动态地将依赖对象注入到组件中。

**4.IoC 三种依赖注入的方式**

1）构造方法注入（constructor injection）

顾名思义，就是被注入对象可以通过在其构造方法中声明依赖对象的参数列表，让IoC容器知道它需要那些依赖对象。

2）setter 方法注入（setter injection）

当前对象只要为其依赖对象所对应的属性添加setter方法，就可以通过setter方法将相应的对象为其依赖对象设置到被注入对象中。

3）接口注入（interface injection）

通过接口注入。

**5.IoC容器的职责**

1) 业务对象的构建管理：IoC需要将业务对象的构建逻辑从客户端对象那里分离出来，从而达到各层之间解耦的效果。

2）业务对象的依赖注入：IoC结合之前构建和管理的业务对象，以及各个业务对象间的依赖关系，将这些对象所依赖的对象注入绑定，使业务对象在使用的时候出于就绪状态。

3）对业务对象生命周期的管理、企业级服务的集成、AOP的支持等

**6为何使用spring，好处：**

1）降低组件之间的耦合度，实现软件各层的解耦。

2）使用容器提供的众多服务（eg：事务管理、消息服务、JMS服务）

3）容器提供单例模式

4）提供AOP技术（eg：权限拦截、运行期监控）

5）容器提供众多辅助类，可以加快开发效率）（eg：HibernateTemplate、JdbcTemplate）

6）Spring对于主流的应用框架提供了集成支持（eg：Struts、Hibernate、JPA）

**十、JAVA开源框架（Struts篇）**

**1.什么是MVC模式，原理是什么？**

MVC是一种程序开发设计模式,它实现了显示模块与功能模块的分离。提高了程序的可维护性、可移植性、可扩展性与可重用性，降低了程序的开发难度。它主要分模型、视图、控制器三层。

原理：

1）模型（Model）：模型是应用程序的主体部分。模型表示业务数据，或者业务逻辑.

2）视图（View）：视图是应用程序中用户界面相关的部分，是用户看到并与之交互的界面。

3）控制器（controller)：控制器工作就是根据用户的输入，控制用户界面数据显示和更新model对象状态。

**2.MVC的优缺点**

优点：

1）低耦合性。因为模型与控制器和视图相分离，很容易改变应用程序的视图层、数据层和业务规则。

2）高重用性和可适用性。MVC模式允许你使用各种不同样式的视图来访问同一个服务器端的代码。它包括任何WEB（HTTP）浏览器或者无线浏览器（wap）；由于模型返回的数据没有进行格式化，所以同样的构件能被不同的界面使用。例如，很多数据可能用HTML来表示，但是也有可能用WAP来表示，而这些表示所需要的仅令是改变视图层的实现方式，而控制层和模型层无需做任何改变。

3）较低的生命周期成本。MVC使降低开发和维护用户接品的技术含量成为可能。

4）快速的部署。使用MVC模式使开发时间得到相当大的缩减，它使程序员（Java开发人员）集中精力于业务逻辑，界面程序员（HTML和JSP开发人员）集中业务于表现形式上。

5）可维护性。视图层和业务逻辑层也使得WEB应用更易于维护和修改。

6）有利于软件工程化管理。由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化管理程序代码。

缺点：

（1）增加了系统结构和实现的复杂性。对于简单的界面，严格遵循MVC，使模型、视图与控制器分离，会增加结构的复杂性，并可能产生过多的更新操作，降低运行效率。

（2）视图与控制器间的过于紧密的连接。视图与控制器是相互分离，但又紧密联系的部件，如何视图没有控制器的存在，那它的应用是很有限的，反之亦然，这样就妨碍了它们的独立重用。

（3）视图对模型数据的低效率访问。依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。

（4）目前，一般高级的界面工具或构造器不支持MVC架构。改造这些工具以适应MVC需要和建立分离的部件的代价是很高的，从而造成使用MVC的困难。

**十一JAVA开源框架（Hibernate篇）**

**十二DB 笔试题**

**1.一个数据表达到500万条数据记录后，访问性能下降，如何优化？**

**2.Oracle数据库中有一表tab**

no name

1 a

2 b

3 c

4 d

你写一条sql语句显示以下结果

ab ac ad bc bd cd

答：

select t1.name||t2.name

from tab t1,tab t2

where t1.no<t2.no

order by t1.name,t2.name;

**3.oracle中minus的用法**

minus运算的主要功能是： A-B (A与B的差集);

在两个查询结果的时中，返回的第一个查询结果中与第二个查询结果记录的差集

**4.Rowid和Rownum的区别？**

rowid和rownum都是虚列，但含义完全不同。

1. rowid是物理地址，用于定位oracle中具体数据的物理存储位置
2. rownum是对结果集加的一个伪列，即先查到结果集之后再加上去的一个列 (强调：先要有结果集)。

简单的说 rownum 是对符合条件结果的序列号。它总是从1开始排起的。所以你选出的结果不可能没有1，而有其他大于1的值。

注意：rownum不能以任何基表的名称作为前缀

## 5. 写一条sql语句让你变为这样的表

教师号　　星期号　是否有课

　１　　　　２　　　有

　１　　　　３　　　有

　２　　　　１　　　有

　３　　　　２　　　有`

　１　　　　２　　　有

select tid , sum(case wid when 1 then 1 else 0 end) 星期一

, sum(case wid when 2 then 1 else 0 end) 星期二

, sum(case wid when 3 then 1 else 0 end) 星期三

from tb\_test group by tid

**十三JAVA程序题**

**1．阅读下列程序，写出打印结果**

**public** **class** Test{

**public** **static** **void** main(String args[]){

**int** a=0; **int** b=1; **int** c=2;**int** d=3;

System.*out*.println(a+++b+c+++d);//切忌粗心

}

}

分析: 根据程序可以知道，这题考的是算术符号的优先级的问题。++ 优先级高于 + ，a++ 与 ++a 优先级是同等的。

所以可以转化为: System.*out*.println((a++) + b +(c++) +d)//先 a+b 后a++

打印结果是：6

在java中，a++ 和 ++a的相同点都是给a+1，不同点是a++是先参加程序的运行再+1，而++a则是先+1再参加程序的运行。

**2. 阅读下列程序，写出打印结果**

Integer a = **new** Integer(1234);

Integer b = **new** Integer(1234);

System.*out*.println(a==b);

System.*out*.println(a.equals(b));

分析：入口得根据 == 与 equeals() 的作用与区别着手

结果：false

True

**十四JAVA设计模式**

1.装饰模式

1）装饰者和被装饰对象具有相同的超类型，你可以用一个或多个装饰者去包装一个对象。

2）因为装饰者和被装饰者具有相同的超类型，所以在需要原始对象（被包装的）的场合，可以用修饰过的对象代替它。

3）装饰者可以在所委托被装饰者行为之前或之后加上自己的行为，以达到特定的目的。

4）对象可以在任何时候被修饰，可以在运行时动态地、不限量地用你喜欢的装饰者来装饰对象。

2.设计原则

1）把会变化的部分取出并封装起来，以便以后可以轻易地改动和扩充此部分，而不影响不需要变化的部分

2）针对接口编程，而不是针对实现编程

2.观察者模式（observer）

1）意图：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自动更新。

2）别名：依赖（Dependents）、发布—订阅（Publish—Subscribe）

3）适用性——一以下任一情况可使用观察者模式：

* 当一个抽象模型有两个方面，其中一个方面依赖另一个方面。将这两者封装在独立的对象中以使它们可以各自独立地改变和复用。
* 当一个对象的改变同时需要改变其他对象，而不知道具体有多少个对象有待改变
* 当一个对象必须通知其它对象，而它又不能假定对象是谁。换言之，你不希望这些对象是紧密耦合的。

3. OO的设计原则

1）封装变化

2）多用组合，少用继承

3）针对接口编程，不针对实现编程

4）为交互对象之间的松耦合设计而努力

**十五JAVA与算法**

**十六JAVA 集合**

1.队列（Queue）

1）队列通常有两种实现方式：一种是循环数组，另一种是链表（LinkedList实现了Queue接口）。循环数组总是比链表更加高效，但同时使用循环数组是有局限性，它的容量是有限的。

2）为了实现自己的队列类，扩展自AbstractQueue类比Queue接口容易得多。

2.数组与链表

1）数组查找快速，但是增加、删除一个元素的时候比较费事。链表可以解决数组中存在的问题，但是列表的查找是费事的。

2）数据是将对象的引用放在连续的存储位置中，而链表是将每个对象存放在独立的结点中。

**十七.JPA**

**1.实体（Entity）**

1）实体：它是一个轻量的持久化的域对象（domain object）

2）实体类需用注解或XML将它声明为实体。必须有一个无参数的构造方法（需是public、protected），实体类不能用final修饰（包括方法和实例变量）

注意：实体类必须一个顶层类，接口和枚举都不能作为实体

**2.实体关系**

实体间的关系有：一对一、一对多、多对一、多对多，关系是多态的。关系可以是单向或双向的。双向关系同时具有持有端（owning side）和反向端（inverse side），单向关系只有持有端。关系的持有端决定数据库的更新。

以下规则应用于双向关系：

1）双向关系中的反向端必须通过OneToOne，OneToMany或ManyToMany注解的MappedBy元素指向持有端。

2）一对多或多对一的双向关系中的“多”端必须是持有端（owning side），因此不能在ManyToOne注解中使用MappedBy元素。

3）对于一对一的双向关系，包含对应的外键（foreign key）的那一端是持有端。

4）对于多对多的双向关系，任何一端都可以作为持有端。

**3.实体实例的生命周期**

在EntityManager中用于管理实体实例的生命周期，分为：new、managed、detached、removed四种状态：

1）new：一个新建的实体实例没有持久化标识，并且没有与持久化上下文相关联。

2）managed：一个受管的实体实例有一个正与持久化上下文相关联的持久化标识。

3）detached：一个脱管的实体实例有持久化标识，但这个标识没有（或已经不再）与一个下文向关联

4）removed：被删除的实体实例具有持久化标识，并且与持久化上下文相关联，但是被计划从数据库中删除。

**4.持久化的entity实例**

对于一个新的实体实例调用persist方法使它成为受管的、持久化的实例，通过persist的级联传播也可以获得同样的效果。对实体X调用persist操作，其意：

1）如果X是一个新的实体，将变成受托管的。事务提交时或提交前，或作为flush操作的结果，实体X将进入到数据库中。

2）如果X是一个已有的受管对象，持久化会忽略它。但是有从X到其它实体的关联被标注为cascade=persist 或 cascade = ALL 或XML部署描述符中有等价的配置。Persist操作将被传递到X引用的实体上去。

3）如果X是已删除的实体，它重新成为受管实体。

4）如果X是脱管对象，在调用persist在操作时可能抛出EntityExistsException，也可能是在flush或commit的时候抛出EntityExistsException或PersistenceException

**3.持久化上下文的生命周期**

1）事务范围：容器管理的持久化上下文的生命周期要么限定在一个事务的范围内（PersistenceContextType.TRANSACTION）,缺省的情况下，容器的持久化上下文的生命周期对应于事务的类型是：PersistenceContextType.TRANSACTION。

2）扩展范围：要么扩展到超出单个事务范围（PersistenceContextType.EXTENDED），它的存在始于EntityManager实例被创建，终止于EntityManager被关闭。

两种事务类型的实体状态：

* 事务提交：事务提交后，事务范围的持久化上下文中的受管实体成为脱管实体；扩展的持久化上下文中受管实体仍然是受管实体。
* 事务回滚：都会导致所有已存在的受管实例和已删除的实例都变成脱管，这些实例保持它们在事务回滚的那一刻的状态。

**十八.SOA那点东东**

**1. WSDL**

1）概念

WSDL 指网络服务描述语言 (Web Services Description Language)。

WSDL 是一种使用 XML 编写的文档。这种文档可描述某个 Web service。它可规定服务的位置，以及此服务提供的操作（或方法）。

2）WSDL 文档结构

WSDL 文档是利用这些主要的元素来描述某个 web service 的：

|  |  |
| --- | --- |
| **元素** | **定义** |
| <portType> | web service 执行的操作 |
| <message> | web service 使用的消息 |
| <types> | web service 使用的数据类型 |
| <binding> | web service 使用的通信协议 |

**WSDL 端口**

**<portType>** 元素是最重要的 WSDL 元素。

它可描述一个 web service、可被执行的操作，以及相关的消息。

可以把 <portType> 元素比作传统编程语言中的一个函数库（或一个模块、或一个类）。

**WSDL 消息**

**<message>** 元素定义一个操作的数据元素。

每个消息均由一个或多个部件组成。可以把这些部件比作传统编程语言中一个函数调用的参数。

**WSDL types**

**<types>** 元素定义 web service 使用的数据类型。

为了最大程度的平台中立性，WSDL 使用 XML Schema 语法来定义数据类型。

**WSDL Bindings**

**<binding>** 元素为每个端口定义消息格式和协议细节。

3）操作类型

请求-响应是最普通的操作类型，不过 WSDL 定义了四种类型：

|  |  |
| --- | --- |
| **类型** | **定义** |
| One-way | 此操作可接受消息，但不会返回响应。 |
| Request-response | 此操作可接受一个请求并会返回一个响应 |
| Solicit-response | 此操作可发送一个请求，并会等待一个响应。 |
| Notification | 此操作可发送一条消息，但不会等待响应。 |

**2.UDDI**

**十九 JavaScript**

**1.JavaScript的实现**

一个完整的JavaScript实现由三个部分组成：

1）核心（ECMAScript）：ECMAScript是一个描述，定义了脚本语言的所有属性、方法和对象。其他语言可以实现ECMAScript来作为功能的基础，JavaScript就是这样（当然ActionScript也是）。

2）文档对象模型（DOM）：是HTML和XML的应用程序接口（API）。DOM将整个页面规划成由节点层级构成的文档。

3）浏览器对象模型（BOM）：BOM主要处理浏览器的窗口和框架，特定浏览器的扩展都是BOM的一部分：

* 对Cookie的支持
* 弹出新的窗口、移动、关闭、改变浏览器窗口大小
* 提供用户屏幕分辨率详细信息的屏幕对象

等等.....

**2. ECMAScript基础概念**

1）区分大小写，eg：变量test 与 Test不一样

2）变量是弱类型的，变量无特定的类型

3）每行的分号可有可无

4）注释与JAVA一样

5）括号是代码块，表示一系列应该按照顺序执行的语句。{}

**3.变量名的规范**

1）第一个字符必须是字母、下划线（ \_ ）或美元符号（$）

2）余下的字符可以是字母、下划线（ \_ ）、美元符号（$）和数字

3）首字母必须小写，接下来的单词首字母须大写

规范的，eg：var test，$test，\_test

**4.原始值和引用值**

1）原始值：是存储在栈中简单数据段，值直接存储在变量访问的位置。有String、Number、Null、Boolean、Undefined 5种原始类型

2）引用值：是存储在堆中的对象，即是存储的一个指针。

**5.typeof 运算符**

1）“undefined”,变量是Undefined

2) “boolean”,变量是Boolean

3）“number”,变量是Number

4）“string”,变量是String

5）“object”,变量是Null或者是一种引用类型

**6.一元运算符**

1）delete：delete运算符删除开发者定义的对象属性或方法的引用

2）void：void运算符对任何值都返回undefined，通常用于避免输出不该输出的值

3）= = =：全等号由三个等号组成，只有在无需类型转换的情况下运算数相等，才返回True

eg: var snum = “55”; var nnum= 55;

alert( snum = = nnum); // true

alert( snum = = = nnum); // false

4）!= = ：非全等号在无需类型转换的情况下运算数不相等，才返回 True

7.闭包

所谓闭包，是指词法表示包括不必计算的变量的函数，函数能使用函数外的变量。