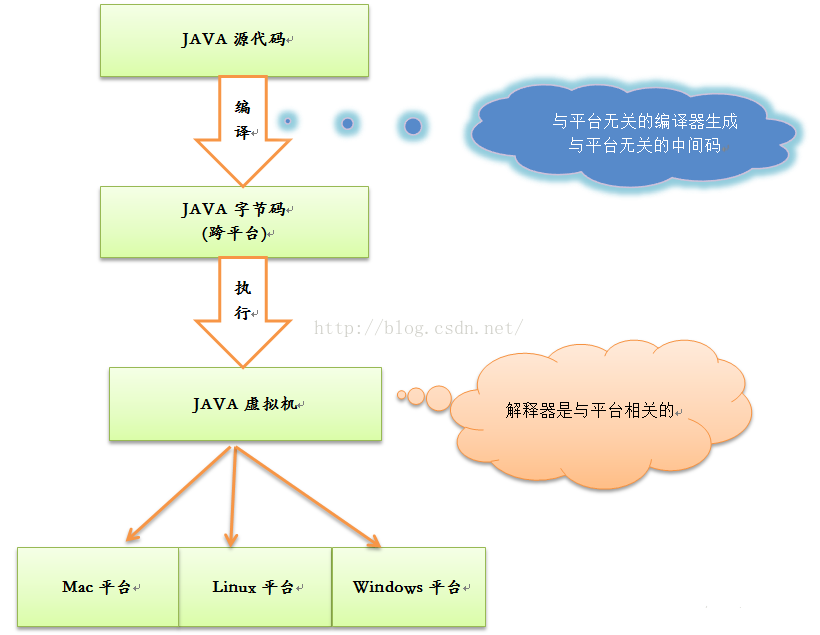
**面试题**

1. **Java**
2. **运行机制**
3. **Java跨平台原理**



Java实现跨平台是JVM(Java虚拟机)起的作用。如果是C/C++的编译方式，一旦换了一个平台，那么就需要重新编译一份对应的可执行代码，但是Java则不同，编译好了一份Java字节码，换到不同的平台上时，并不需要重新编译，前提是这些平台上都安装了相应平台的JVM，JVM不是跨平台的。

1. **垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？**

垃圾回收器通常是作为一个单独的低级别的线程运行，在不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。

垃圾回收器不可以马上回收内存，程序员可以手动执行System.gc()，通知GC运行， 但是Java语言规范并不保证GC一定会执行。

1. **垃圾回收机制**
2. 分代复制垃圾回收

不同的对象的生命周期是不一样的。因此，不同生命周期的对象可以采取不同的收集方式，以便提高回收效率。 在[Java](http://lib.csdn.net/base/java" \t "_blank" \o "Java 知识库)程序运行的过程中，会产生大量的对象，其中有些对象是与业务信息相关，比如Http请求中的 Session对象、线程、Socket连接，这类对象跟业务直接挂钩，因此生命周期比较长。 但是还有一些对象，主要是程序运行过程中生成的临时变量，这些对象生命周期会比较 短，比如：String对象，由于其不变类的特性，系统会产生大量的这些对象，有些对象甚至只用一次即可回收。如果每次垃圾回收都是对整个堆空间进行回收，花费时间相对会长，并且生命周期长的对象依旧存在，因此引入分代回收，把不同生命周期的对象 放在不同代上，不同代上采用最适合它的垃圾回收方式进行回收。

虚拟机中的共划分为三个代：年轻代（Young Generation）、年老点（Old Generation）和持久代（Permanent Generation）。由于对象进行了分代处理，因此垃圾回收区域、时间也不一样。

1. 标记垃圾回收

在使用标记清除算法时,未引用对象并不会被立即回收.取而代之的做法是,垃圾对象将一直累计到内存耗尽为止.当内存耗尽时,程序将会被挂起,垃圾回收开始执行.当所有的未引用对象被清理完毕时,程序才会继续执行。

标记清除算法由两个阶段组成:

 标记阶段，标记所有的可访问对象。

 收集阶段，垃圾收集算法扫描堆并回收所有的未标记对象。

1. 增量垃圾回收

简单地说，它的存在是为了解决标记清除的长停顿问题。增量收集器把堆栈分为多个域，每次仅从一个域收集垃圾。这会造成较小的应用程序中断。

1. **JVM加载class文件的原理机制？**

答：JVM中类的装载是由类加载器（ClassLoader）和它的子类来实现的，Java中的类加载器是一个重要的Java运行时系统组件，它负责在运行时查找和装入类文件中的类。

由于Java的跨平台性，经过编译的Java源程序并不是一个可执行程序，而是一个或多个类文件。当Java程序需要使用某个类时，JVM会确保这个类已经被加载、连接（验证、准备和解析）和初始化。

类的加载是指把类的.class文件中的数据读入到内存中，通常是创建一个字节数组读入.class文件，然后产生与所加载类对应的Class对象。加载完成后，Class对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备（为静态变量分配内存并设置默认的初始值）和解析（将符号引用替换为直接引用）三个步骤。最后JVM对类进行初始化，包括：

1)如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；

2)如果类中存在初始化语句，就依次执行这些初始化语句。

类的加载是由类加载器完成的，类加载器包括：

Bootstrap：一般用本地代码实现，负责加载JVM基础核心类库（rt.jar）；

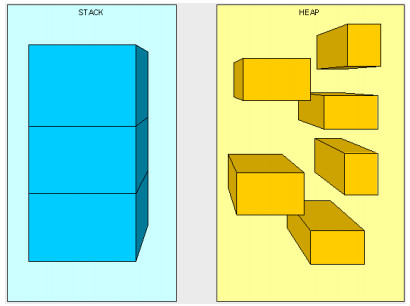
Extension：从java.ext.dirs系统属性所指定的目录中加载类库，它的父加载器是Bootstrap；

System：又叫应用类加载器，其父类是Extension。它是应用最广泛的类加载器。它从环境变量classpath或者系统属性java.class.path所指定的目录中记载类，是用户自定义加载器的默认父加载器。

1. **Java内存**

**堆（heap）与栈（stack）**

在java中，Main函数就是栈的起始点，也是程序的起始点。程序要运行总是有一个起点的（程序执行的入口）。



**概括:**

1**.栈是运行时的单位 , 而堆是存储的单元**。

 2.**栈解决程序的运行问题，即程序如何执行，或者说如何处理数据，**

**堆解决的是数据存储的问题，即数据怎么放，放在哪儿**。

在java中一个线程就会相应有一个线程栈与之对应，这点很容易理解，因为不同的线程执行逻辑有所不同，因此需要一个独立的线程栈。

而堆则是所有线程共享的。

**疑问一：为什么要把堆和栈区分出来呢？栈中不是也可以存储数据吗？**

     1. 从软件设计的角度看，栈代表了处理逻辑，而堆代表了数据。这样分开，使得处理逻辑更为清晰。分而治之的思想。

    这种隔离、模块化的思想在软件设计的方方面面都有体现。

     2.堆与栈的分离，使得堆中的内容可以被多个栈共享（也可以理解为多个线程访问同一个对象）。

        好处:  a.提供了一种有效的数据交互方式（如：共享内存）

                 b.堆中的共享常量和缓存可以被所有栈访问，节省了空间。

     3. 栈因为运行时的需要，比如保存系统运行的上下文，需要进行地址段的划分。

        由于栈只能向上增长，因此就会限制住栈存储内容的能力，

        而堆不同，堆中的对象是可以根据需要动态增长的，

        因此栈和堆的拆分使得动态增长成为可能，相应栈中只需记录堆中的一个地址即可。

     4. 面向对象就是堆和栈的完美结合。

    其实，面向对象方式的程序与以前结构化的程序在执行上没有任何区别。

    但是，面向对象的引入，使得对待问题的思考方式发生了改变，而更接近于自然方式的思考。

    当我们把对象拆开，你会发现，对象的属性其实就是数据，存放在堆中；

        而对象的行为（方法），就是运行逻辑，放在栈中。

        我们在编写对象的时候，其实就是编写了数据结构，也编写了处理数据的逻辑。不得不承认，面向对象的设计，确实很美。

**疑问二：**  **堆中存什么？栈中存什么？**

     1. 栈存储的信息都是跟当前线程（或程序）相关的信息。(局部变量、程序运行状态、方法、方法返回值)等，

         栈中存的是基本数据类型和堆中对象的引用。一个对象的大小是不可估计的，或者说是可以动态变化的，但是

         在栈中，一个对象只对应了一个4byte的引用（堆栈分离的好处）。

     2. 堆只负责存储对象信息。

**疑问三：  为什么不把基本类型放堆中呢？**

     1. 其占用的空间一般是1~8个字节---需要空间比较少，

     2.而且因为是基本类型，所以不会出现动态增长的情况---长度固定，因此栈中存储就够了，如果把它存在堆中是没有什么意义的（还会浪费空间）。

**疑问四：**  **java中的参数传递是传值呢？还是传引用？**

     对象传递是引用值传递，原始类型数据传递是值传递

     实际上这个传入函数的值是对象引用的拷贝，即传递的是引用的地址值，所以还是按值传递

      堆和栈中，栈是程序运行最根本的东西。程序运行可以没有堆，但是不能没有栈。

      而堆是为栈进行数据存储服务的，堆就是一块共享的内存。

    不过，正是因为堆和栈的分离的思想，才使得java的垃圾回收成为可能。

      java中，栈的大小通过-Xss来设置，当栈中存储的数据比较多时，需要适当调大这个值，否则会出现 **java**.lang.StackOverflowError异常。

1. Dd
2. D
3. **基础语法**
4. **一个".java"源文件中是否可以包括多个类（不是内部类）？有什么限制？**

可以有多个类，但只能有一个public的类，并且public的类名必须与文件名相一致。

一个文件中可以只有非public类，如果只有一个非public类，此类可以跟文件名不同

1. **&和&&的区别。**

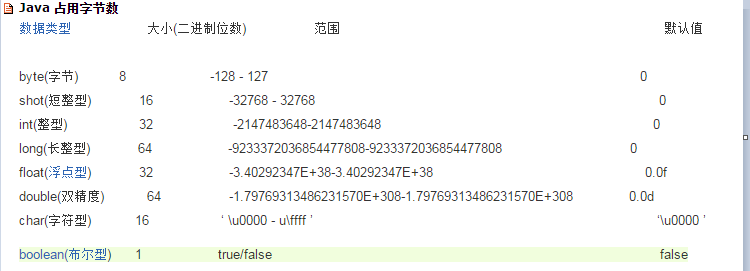
&和&&都可以用作逻辑与的运算符，&&为短路与，&不是短路与。

另外&可以做为整数的位运算符

例1：对于if(str != null&& !str.equals(“”))表达式，当str为null时，后面的表达式不会执行，所以不会出现NullPointerException如果将&&改为&，则会抛出NullPointerException异常。

例2：If(x==33 &++y>0) y会增长，If(x==33 && ++y>0)不会增长

1. **java中int数据占几个字节**



1. **Integer与int的区别**

int是java提供的8种原始数据类型之一，另外Java为每个原始类型提供了封装类，Integer是java为int提供的封装类。int的默认值为0，而Integer的默认值为null，即Integer可以区分出未赋值和值为0的区别，int则无法表达出未赋值的情况。

如在业务中表示year，如果用int,在没有赋值时，int类型会给一个默认值0，这样就不知道这个year=0,是手工还是默认给的值，而定义成Integer则在没有赋值时是null,推荐使用封装类型。

表示钱： BigDecimal或Integer（表示分）

1. **请设计一个一百亿的计算器**

如果只是大整数运算，使用BigInteger就可以

如果有浮点数据参与去处，需要使用BigDecimal进行运算

Java中基本类型的浮点数运算是不精确的，需要使用BigDecimal运算，尤其是金融、会计方向的软件

1. **char型变量中能不能存贮一个中文汉字?**

char型变量是用来存储Unicode编码的字符的，unicode编码字符集中包含了汉字，所以，char型变量中可以存储汉字。

1. **short s1 = 1; s1 = s1 + 1;有什么错? short s1 = 1; s1 += 1;有什么错?**

对于short s1 = 1; s1 = s1 + 1;由于s1+1运算时会自动提升表达式的类型，所以结果是int型，再赋值给short类型s1时，编译器会提示错误，需要强制转换类型。

对于short s1 = 1; s1 += 1;由于+=是java语言规定的运算符，Java编译器会对它进行特殊处理，因此可以正确编译。

1. **"=="和equals方法有什么区别？**

==操作符专门用来比较两个变量的值是否相等，也就是用于比较变量所对应的内存中所存储的数值是否相同，要比较两个基本类型的数据或两个引用变量是否相等，只能用==操作符。

如果一个变量指向的数据是对象类型的，那么这时候涉及了两块内存，对象本身占用一块内存（堆内存），变量也占用一块内存，例如Objet obj = new Object();变量obj是一个内存，new Object()是另一个内存，此时，变量obj所对应的内存中存储的数值就是对象占用的那块内存的首地址。对于指向对象类型的变量，如果要比较两个变量是否指向同一个对象，即要看这两个变量所对应的内存中的数值是否相等，这时候就需要用==操作符进行比较。

equals方法是用于比较两个独立对象的内容是否相同，就好比去比较两个人的长相是否相同，它比较的两个对象是独立的。例如，对于下面的代码：

String a=new String("foo");

String b=new String("foo");

两条new语句创建了两个对象，然后用a,b这两个变量分别指向了其中一个对象，这是两个不同的对象，它们的首地址是不同的，即a和b中存储的数值是不相同的，所以，表达式a==b将返回false，而这两个对象中的内容是相同的，所以，表达式a.equals(b)将返回true。

在实际开发中，我们经常要比较传递进行来的字符串内容是否等，例如，String input =…;input.equals(“quit”)，如果一个类没有自己定义equals方法，那么它将继承Object类的equals方法，Object类的equals方法的实现代码如下：

boolean equals(Object o){

return this==o;

}

这说明，如果一个类没有自己定义equals方法，它默认的equals方法（从Object类继承的）就是使用==操作符，也是在比较两个变量指向的对象是否是同一对象，这时候使用equals和使用==会得到同样的结果，如果比较的是两个独立的对象则总返回false。如果你编写的类希望能够比较该类创建的两个实例对象的内容是否相同，那么你必须覆盖equals方法，由你自己写代码来决定在什么情况即可认为两个对象的内容是相同的。

当此方法被重写时，通常有必要重写 hashCode 方法，以维护 hashCode 方法的常规协定，该协定声明相等对象必须具有相等的哈希码。

hashcode这个方法是用来鉴定2个对象是否相等的。hashcode方法一般用户不会去调用，比如在hashmap中，由于key是不可以重复的，他在判断key是不是重复的时候就判断了hashcode这个方法，而且也用到了equals方法。这里不可以重复是说equals和hashcode只要有一个不等就可以了。所以简单来讲，hashcode相当于是一个对象的编码。我们一般在覆盖equals的同时也要覆盖hashcode，让他们的逻辑一致。

1. **作用域public，private，protected，以及不写时的区别?**

修饰符 当前类 同 包 子 类 其他包

public √ √ √ √

protected √ √ √ ×

default √ √ × ×

private √ × × ×

类的成员不写访问修饰时默认为default。默认对于同一个包中的其他类相当于公开（public），对于不是同一个包中的其他类相当于私有（private）。

受保护（protected）对子类相当于公开，对不是同一包中的没有父子关系的类相当于私有。

Java中，外部类的修饰符只能是public或默认，类的成员（包括内部类）的修饰符可以是以上四种。

1. **Java 常用包**

java.lang--语言包：Java语言的基础类，包括Object类、Thread类、String、Math、System、Runtime、Class、Exception、Process等，是Java的核心类库

java.util--实用工具包：Scanner、Date、Calendar、LinkedList、Hashtable、Stack、TreeSet等；

java.NET--网络功能包：URL、Socket、ServerSocket等；

java.sql--数据库连接包：实现JDBC的类库；

java.io--输入输出包：提供与流相关的各种包；

1. **Java常用接口**

 Comparable ,Collection,Set, List, Map, Runnable Iterable Iterator 等

1. D
2. Dd
3. d
4. **面向对象**
5. **你对面向对象思想的理解？**

面向对象编程（Object-Oriented Programming）简称OOP技术，是开发计算机应用程序的一种新方法、新思想。过去的面向过程编程中常常会导致所有的代码都包含在几个模块中，使程序难以阅读和维护，在做一些修改时常常牵一动百，使以后的开发和维护难以为继。而使用OOP技术，使用许多代码模块，每个模块都只提供特定的功能，它们是彼此独立的，可以增加代码重用的几率，更加有利于软件的开发、维护和升级。另外OOP的三大核心特性：继承、封装、多态的特性，使得在面对象编上能够设计出高内聚、低耦合的系统结构，使得系统更灵活、更容易扩展，而且成本较低，所以这一编程思想是目前一种应用最为普遍的软件设计思想。

**封装**

封装是保证软件部件具有优良的模块性的基础，封装的目标就是要实现软件部件的“高内聚、低耦合”，防止程序相互依赖性而带来的变动影响。面向对象的封装就是把描述一个对象的属性和行为的代码封装在一个“模块”中，也就是一个类中，属性用变量定义，行为用方法进行定义，方法可以直接访问同一个对象中的属性。

**继承**

在定义和实现一个类的时候，可以在一个已经存在的类的基础之上来进行，把这个已经存在的类所定义的内容作为自己的内容，并可以加入若干新的内容，或修改原来的方法使之更适合特殊的需要，这就是继承。继承是子类自动共享父类数据和方法的机制，这是类之间的一种关系，提高了软件的可重用性和可扩展性。

**多态**

多态指的同一个对象，在程序不同时刻的多种运行状态。举例：水(气态，液态，固态)。

所谓多态，指的就是父类引用指向子类对象，调用方法时会调用子类的实现而不是父类的实现。多态的实现的关键在于“动态绑定”。

为使多态能运行，存在着继承或者实现关系

可以将子类的对象赋给父类/接口，以多态的形式来传递参数，增强了参数类型的灵活性。即父类(接口)引用指向子类(实现)对象

多态的好处和弊端：

好处：多态的存在提高了程序的扩展性和后期可维护性

弊端：虽然可以预先使用，但是只能访问父类中已有的功能，运行的是后期子类的功能内容。不能预先使用子类中定义的特有功能。

多态应用场景：当接口已经确定，但同一个接口在不同环境需要不同实现的时候。如：工厂模式

工厂方法模式（Factory Mehtod Pattern）：定义一个用于创建对象的接口，让子类决定哪一个类实例化。工厂方法模式让一个类的实例化延迟到其子类。工厂方法模式又称为工厂模式，又称为虚拟构造器模式或多态工厂模式。工厂方法模式是一种类创建型模式。

1. **object中定义了哪些方法？**

protected Object   clone()创建并返回此对象的一个副本。     
boolean   equals(Object obj)指示其他某个对象是否与此对象“相等”。     
protected void   finalize()当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。     
Class<?>   getClass()返回此 Object 的运行时类。     
int   hashCode()返回该对象的哈希码值。     
void   notify()唤醒在此对象监视器上等待的单个线程。     
void   notifyAll()唤醒在此对象监视器上等待的所有线程。     
String   toString()返回该对象的字符串表示。     
void   wait()在其他线程调用此对象的 notify() 方法或 notifyAll() 方法前，导致当前线程等待。     
void   wait(long timeout)在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量前，导致当前线程等待。     
void   wait(long timeout, int nanos)在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量前，导致当前线程等待。

1. **Java对象初始化顺序**

分为两种，一种是本类的初始化，一种是含有父类的初始化顺序。

本类的初始化顺序是：静态变量、静态初始化块、变量、初始化块、构造函数

继承类的初始化顺序是：父类静态变量、父类静态初始化块、子类静态变量、子类静态初始块、父类变量、父类初始化块、父类构造函数、子类变量、子类初始化块、子类构造函数。

【分析】

static{

System.out.println("静态块");

}

{

System.out.println("初始化模块"); }

public ClassName() {

System.out.println("构造方法");

}

说明：

原则上回答全面的话，应该是完整的说出带有继承的这种类的初始化过程，下面有个步骤可以参考：

1.装载程序的时候首先找到的是它的基（父）类,如果有多层基（父）类则会一级一级的往上找最后找到根基（父）类。

2.执行根基础（父）类中的static初始化,再执行下一个衍生类中的static,依此类推,一直保持这个顺序。

3.此时类已经装载完毕,开始创建对象,所有的基本数据类型都会设成它们的默认值,对象句柄设为null

4.调用基础（父）类的构造方法,基础（父）类的构建采用与衍生类构造方法完全相同的处理过程。

5.构造方法初始完之后,进行变量的初始化。

6.执行构造方法中剩余的部分。

1. **方法的覆盖/重写Overload和Override的区别？Overload的方法是否 可以改变返回值的类型?**

Overload是重载，Override是覆盖，也就是重写。

重载Overload表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同（即参数个数或类型不同）。

重写Override表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。子类覆盖父类的方法时，只能比父类抛出更少的异常，或者是抛出父类抛出的异常的子异常，因为子类可以解决父类的一些问题，不能比父类有更多的问题。子类方法的访问权限只能比父类的更大，不能更小。

返回值范围更小。是否可以改变返回值类型，在重载的定义中，与方法是什么类型返回值无关

1. **abstract class（抽象类）和interface（接口）有什么区别?**

含有abstract修饰符的class即为抽象类，abstract类不能创建实例对象。含有abstract方法的类必须定义为abstract class，abstract class类中的方法不必是抽象的。abstract class类中定义抽象方法必须在具体子类中实现，所以，不能有抽象构造方法或抽象静态方法。如果子类没有实现抽象父类中的所有抽象方法，那么子类也必须定义为abstract类型。

接口（interface）可以说成是抽象类的一种特例，接口中的所有方法都必须是抽象的。接口中的方法定义默认为public abstract类型，接口中的成员变量类型默认为public static final。

接口是一种特殊的抽象类，这种抽象类中只包含常量和方法的定义(抽象方法)，而没有变量和方法的实现。

抽象类：有抽象方法和非抽象方法

接口：所有方法都是抽象的

一个接口可以实现多个父接口，将实现的各个接口用，隔开

接口的特点

A:是对外暴露的规则

B:是功能的扩展

C:接口的出现降低耦合性。

耦合(类与类之间的关系)

内聚(类完成功能的能力)

编程规范：高内聚低耦合

D:接口可以多实现。如：CPU和主板、笔记本的USB插口、插座

1. **abstract的method是否可同时是static,是否可同时是native，是否可同时是synchronized?**

abstract的method不可以是static的，因为抽象的方法是要被子类实现的，而static与子类扯不上关系！

native方法表示该方法要用另外一种依赖平台的编程语言实现的，不存在着被子类实现的问题，所以，它也不能是抽象的，不能与abstract混用。

synchronized和abstract合用的问题不能共用，abstract方法只能存在于抽象类或接口中，它不能直接产生对象，而默认synchronized方法对当前对象加锁，没有对象是不能加锁。

另外synchronized不能被继承，子类继承时，需要另加修改符。

1. **static关键字**

静态初始化块:static{ } : 使用static来修饰的自由块只在类加载的时候执行一次，通常用于初始化静态变量。

A:它只执行一次，它比main还先执行。

B:执行顺序是静态代码块--构造方法

static关键字声明属性(类属性):可以被该类的所有实例对象所共享

static关键字声明方法:可以在不创建对象的情况下,直接使用类名.方法名()的形式调用

静态方法只能访问静态成员

static属性，方法消耗大，要慎用。

静态的特点:随着类的加载而加载

优先于对象存在

对所有对象共享

可以被类名直接调用

1. **是否可以从一个static方法内部发出对非static方法的调用？**

不可以。因为非static方法（实例方法）是要与对象关联在一起的，必须创建一个对象后，才可以在该对象上进行方法调用，而static方法调用时不需要创建对象，可以直接调用。也就是说，当一个static方法被调用时，可能还没有创建任何实例对象，如果从一个static方法中发出对非static方法的调用，那个非static方法是关联到哪个对象上的呢？这个逻辑无法成立，所以，一个static方法内部发出对非static方法的调用。

1. **静态变量和成员/实例变量的区别**

A：调用方式

静态变量也称为类变量，可以直接通过类名调用。也可以通过对象名调用。 这个变量属于类。

成员变量也称为实例变量，只能通过对象名调用。这个变量属于对象。

B：存储位置

静态变量存储在方法区栈中的静态区。

成员变量存储在堆内存。

C：生命周期

静态变量随着类的加载而存在，随着类的消失而消失。生命周期长。

成员变量随着对象的创建而存在，随着对象的消失而消失。

D：与对象的相关性

静态变量是所有对象共享的数据。

成员变量是每个对象所特有的数据。

静态方法可以调用其它的静态方法，但是不能调用非静态方法，这个好比Java中的类变量与实例变量的关系。类变量是被所有类成员共享，而实例变量只被该实例共享

1. **单态（Singleton）设计模式**

在实际开发中常用到，比如数据库的链接。使用Singleton模式可以节省内存，因为它限制了实例的个数，有利于Java垃圾回收。

单态设计模式创建一个类，应满足：

\*为了避免其他程序建立该类对象，先禁止其他程序建立该类对象，即将构造函数私有化

\*为了其他程序访问到该类对象，须在本类中创建一个该类私有对象

\*为了方便其他程序访问到该类对象，可对外提供一个公共访问方式

Runtime类就是单例设计模式。

单例设计模式的两种方式

A:饿汉式 当类加载的时候，就创建对象。

class Single

{

private Single(){}//将构造函数私有化，不让别的类建立该类对象

private static final Single s=new Single();//自己建立一个对象

public static Single getInstance()//提供一个公共访问方式

{

return s;

}

}

B:懒汉式 当使用的使用，才去创建对象。

class Single

{

private Single(){}

private static Single s;

public static Single getInstance()

{

if(s==null)

s=new Single();

return s;

}

} 饿汉式和懒汉式的区别：

饿汉式是类一加载进内存就创建好了对象；

懒汉式则是类才加载进内存的时候，对象还没有存在，只有调用了 getInstance()方法时，

对象才开始创建。

\*\*

懒汉式是延迟加载，如果多个线程同时操作懒汉式时就有可能出现线 程安全问题，解决线程安全问题

可以加同步来解决。但是加了同步之后，每一次都要比较锁，效率就 变慢了，所以可以加双重判断来提高程序效率。

如将上述懒汉式的Instance函数改成同步：

public static Single getInstance()

{

if(s==null)

{

synchronized(Single.class)

{

if(s==null)

s=new Single();

}

}

return s;

}

注：开发常用饿汉式，因为饿汉式简单安全。懒汉式多线程的时候易发生问题

1. **final关键字**

final类:不可被继承，如java.lang.Math就是一个 final类，不可被继承。

final变量:在初始化后不可改变变量值，用于常量定义。如果final变量是引用变量，则不可以改变它的引用对象，但可以改变对象的属性。final修饰的变量是一个常量。只能被赋值一次

final方法:不可被重写

定义常量:

//static作用于成员变量用来表示只保存一份副本，而final的作用是用来保证变量不可变。

public static final double pi=3.14;

1. **内部类**

定义在一个类内部的类（inner class）

内部类的对象能访问它所处类的私有数据

内部类能隐藏起来不为同一个包中其他类访问

当内部类定义在外部类的成员位置，而且非私有，则可以在其他外部类中直接建立内部类对象

格式：外部类名.内部类名 变量名 = new 外部类对象.内部类对象

如：Outer.Inner in = new Outer().new Inner()

1. **匿名内部类**

(1)前提：继承一个类或者实现一个接口

(2)格式：

new 父类名或者接口名()

{

重写父类方法或者实现接口中的方法。

也可以自定义其他方法。

};

(3)什么时候定义匿名内部类？

匿名内部类只是为了简化书写，匿名内部类有局限，通常定义匿 名内部类时，该类方法不超过3个

(4)匿名内部类的好处和弊端：

好处：简化代码书写

弊端：

不能直接调用自己的特有方法

不能执行强转换动作

如果该类里面方法较多，不允许使用匿名内部类

1. **String是最基本的数据类型吗?**

基本数据类型包括byte、int、char、long、float、double、boolean和short。

String是引用数据类型。

java.lang.String类是final类型的，因此不可以继承这个类、不能修改这个类。为了提高效率节省空间，我们应该用StringBuffer/StringBuilder类

1. **String类方法（不可改变）**

Java 中的 String 不可变是因为 Java 的设计者认为字符串使用非常频繁，将字符串设置为不可变可以允许多个客户端之间共享相同的字符串。

java中字符串的处理专门有String类提供的很多API可以很方便的对字符串进行各种操作，如截取、查询、转换成其它类型。特别是在结合正则表达式可以灵活的对字符串进行筛选。

获取方法

int length() 获取字符串的长度

char charAt(int index) 获取特定位置的字符 (角标越界)

int indexOf(String str) 获取特定字符的位置(overload)

int lastIndexOf(int ch) 获取最后一个字符的位置

判断方法

boolean endsWith(String str) 是否以指定字符结束

boolean isEmpty()是否长度为0 如：“” null V1.6

boolean contains(CharSequences) 是否包含指定序列 应用：搜索

boolean equals(Object anObject) 是否相等

boolean equalsIgnoreCase(String anotherString) 忽略大小写是否相等

转换方法

String(char[] value) 将字符数组转换为字符串

String(char[] value, int offset, int count)

Static String valueOf(char[] data)

static String valueOf(char[] data, int offset, int count)

char[] toCharArray() 将字符串转换为字符数组

转换方法

String replace(char oldChar, char newChar) 替换

String[] split(String regex) 切割

String substring(int beginIndex)

String substring(int beginIndex, int endIndex)截取字串

String toUpperCase() 转大写

String toLowerCase() 转小写

1. **StringBuffer与StringBuilder的区别**

这两个类都实现了CharSequence接口。

1. 类型不同，因为不是一个类，也没有继承关系，做参数时不能共用

2. String对象是不可变对象，不能修改值。StringBuffer是可变对象，能修改值。

3. 拼接字符串时，String会产生新对象，而StringBuffer只是增加新字符，不产生新对象，因此效率高。

4. String覆盖了equals方法和hashCode方法，而StringBuffer没有覆盖equals方法和hashCode方法，所以，将StringBuffer对象存储进Java集合类中时会出现问题。

1. 用final关键字修饰一个变量时，是引用不能变，还是引用的对象不能变？

引用变量不能重新赋值，但是引用指向的对象的内容可以变化

例1：final StringBuffer a=new StringBuffer("immutable");

a=new StringBuffer("");

有编译错

例2：

final StringBuffer a=new StringBuffer("immutable");

a.append(“123”);

正确

1. Dd
2. D
3. D
4. d
5. **异常处理**
6. **Error、Exception区别**

Error类和Exception类的父类都是throwable类，他们的区别是：  
Error类一般是指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢等。对于这类错误的导致的应用程序中断，仅靠程序本身无法恢复和和预防，遇到这样的错误，建议让程序终止。  
Exception类表示程序可以处理的异常，可以捕获且可能恢复。遇到这类异常，应该尽可能处理异常，使程序恢复运行，而不应该随意终止异常。

1. **Java中的异常处理机制的简单原理和应用。**

异常是指java程序运行时（非编译）所发生的非正常情况或错误。

Java使用面向对象的方式来处理异常，它把程序中发生的每个异常也都分别封装到一个对象中，该对象中包含有异常的信息。

Java可以自定义异常类，所有异常的根类为java.lang.Throwable，Throwable下面又派生了两个子类：Error和Exception。

1、Error表示应用程序本身无法克服和恢复的一种严重问题，程序只有退的份了，例如说内存溢出和线程死锁等系统问题。

2、Exception表示程序还能够克服和恢复的问题，其中又分为运行时异常和检查异常，运行时异常是软件本身缺陷所导致的问题，也就是软件开发人员考虑不周所导致的问题，软件使用者无法克服和恢复这种问题，但在这种问题下还可以让软件系统继续运行或者让软件死掉。

例如，数组越界（ArrayIndexOutOfBoundsException），空指针异常（NullPointerException）、类转换异常（ClassCastException）；检查异常是运行环境的变化或异常所导致的问题，是用户能够克服的问题，例如，网络断线，硬盘空间不够，发生这样的异常后，程序不应该死掉。

Java为运行时异常和检查异常提供了不同的解决方案，编译器强制检查异常必须try..catch处理或用throws声明继续抛给上层调用方法处理，所以检查异常也称为checked异常，而运行异常可以处理也可以不处理，所以编译器不强制用try..catch处理或用throws声明，所以运行异常也称为Runtime异常。

1. **常见异常**

ArithmeticException（算术异常）

ClassCastException （类转换异常）

IllegalArgumentException （非法参数异常）

IndexOutOfBoundsException （下标越界异常）

NullPointerException （空指针异常）

1. **Java的异常有哪几种，有什么区别？**

两大类，一般异常和运行时异常。一般异常，这些异常是在定义方法时声明抛出的，这些异常必需用try catch抛出，或throws处理，如果不处理程序将编译失败。比如：IOException、FileNotFoundException、SQLException等。

运行时异常是程序运行时可能报出的异常。可以用try catch抓取，也可以不做任何处理。例如：NullPointerException异常是一种比较常见的运行时异常。

1. **try-catch-finally （不要在循环内写try…catch）**

try{

// 可能会抛出特定异常的代码段

}catch(MyExceptionType myException){

// 如果myException 被抛出，则执行这段代码

}catch(Exception otherException){  
//如果另外的异常otherException被抛出，则执行这段代码

} [finally{ //用于资源回收

//无条件执行的语句

}]

1. **throw 和 和 throws 有什么区别？**

throw 关键字用来在程序中明确的抛出异常，相反，throws 语句用来表明方法不能处理的异常。每一个方法都必须要指定哪些异常不能处理，所以方法的调用者才能够确保处理可能发生的异常，多个异常是用逗号分隔的。

1. **final,finally,finalize区别**

final用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。

finally是异常处理语句结构的一部分，表示总是执行。

finalize是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。JVM不保证此方法总被调用

1. a
2. d
3. d
4. 集合
5. 集合类型可以归纳为三种

Iterable

->Collection

->List

->ArrayList

->LinkedList

->Vector

->Stack

->Set

->HashSet

->TreeSet

Map

->Hashtable

->HashMap

->LinkedHashMap

Collections，不属于集合，是集合类的工具类

Arrays，不属于集合类，是数据对象的工具类

1) 列表(List)集合区分元素的顺序，且允许包含重复元素。

Collection:顶层接口

|--->List:列表，元素是有序的(元素带角标索引)，可以有重复元素,可以有null元素。

|--->ArrayList:底层的数据结构是数组数据结构，特点是查询速度快(因为带角标)，但是增删速度稍慢,因为当元素多时，增删一个元素则所有元素的角标都得改变，线程不同步。默认长度是10，当超过长度时，按50%延长集合长度。

|--->LinkedList:底层数据结构式链表数据结构(即后面一个元素记录前一个)，

特点：查询速度慢，因为每个元素只知道前面一个元素，但增删速度快，因为元素再多，增删一个只要让其前后的元素重新相连即可，线程是不同步的。|--->Vector:底层数据结构是数组数据结构.特点是查询和增删速度都很慢。默认长度是10，当超过长度时,按100%延长集合长度。线程同步。

一般情况下，使用哪种List接口下的实现类呢？

如果要求增删快，考虑使用LinkedList

如果要求查询快，考虑使用ArrayList

如果要求线程安全，考虑使用Vector。

2)集(Set):Set集合中不区分元素的顺序，不允许出现重复元素。

|--->HashSet:底层数据结构是哈希表、存取速度快、元素唯一、线程不同步。|--->TreeSet:底层数据结构式二叉树。可以对Set集合中的元素进行排序。元素有序、线程不同步。

3)映射(Map):顶层接口,该集合存储的是键值对,而且键是唯一的,Map和Set很像,Set集合底层就是使用了Map集合。Map集合没有迭代器，要取出元素必须先将Map集合转换成Set集合才能遍历元素

|--->HashTable: 底层是哈希表数据结构；不可以使用null键和null值；用作键的对象必须实现hashCode和equals方法来保证键的唯一性。线程同步效率低|--->HashMap:底层是哈希表数据结构；允许使用null键和null值；线程不同步，效率高；

|--->TreeMap:底层是二叉树结构；允许使用null键和null值；线程不同步；

1. **HashMap的工作原理是什么？**

Java 中的 HashMap 是以键值对(key-value)的形式存储元素的。HaspMap的key可以为null

HashMap 需要一个 hash 函数，它使用 hashCode()和 equals()方法来向集合添加和检索元素。

当调用 put()方法的时候，HashMap 会计算 key 的 hash 值，然后把键值对存储在集合中合适的索引上。如果 key已经存在了，value 会被更新成新值。HashMap 的一些重要的特性是它的容量(capacity)，负载因子(load factor)和扩容极限(threshold resizing)。当put的时候大于等于容量的0.75时，会进行扩容。

多线程环境下若使用HashMap需要使用Collections.synchronizedMap()方法来获取一个线程安全的集合

1. **HashMap，TreeMap，HashTable的区别？**

HashTable不允许<键,值>有空值，HashMap允许<键,值>有空值。

HashTable线程同步，但是HashMap非线程同步。

HashTable中hash数组的默认大小是11，增加方式的old\*2+1，

HashMap中hash数组的默认大小是16，增长方式一定是2的指数倍。

TreeMap能够把它保存的记录根据键排序，默认是按升序排序。

HashTable使用Enumeration，HashMap使用Iterator。

1. **数组(Array) 和列表(ArrayList) 有什么区别？什么时候应该使用 Array 而不是 ArrayList ？**

下面列出了 Array 和 ArrayList 的不同点：

Array 可以包含基本类型和对象类型，ArrayList 只能包含对象类型。

Array 大小是固定的，ArrayList 的大小是动态变化的。

ArrayList 提供了更多的方法和特性，比如：addAll()，removeAll()，iterator()等。

对于基本类型数据，集合使用自动装箱来减少编码工作量。但是，当处理固定大小的基本数据类型的时候，这种方式相对比较慢

1. **ArrayList和Vector的区别**

1. 线程同步，Vector线程安全，ArrayList线程不安全

2. 效率问题，Vector效率低，ArrayList效率高

3. 增长数量，Vector以1.5倍增长，ArrayList以2倍增长

1. **HashSet和TreeSet有什么区别？**

HashSet是由一个 hash 表来实现的，因此，它的元素是无序的。add()，remove()，contains() 方法的时间复杂度是 O(1)。

另一方面，TreeSet 是由一个树形的结构来实现的，它里面的元素是有序的。因此，add()，remove()，contains()方法的时间复杂度是 O(logn)。

1. **Collections和Collection的区别**

Collection是个java.util下的接口，它是各种集合结构的父接口，定义了集合对象的 基本操作方法。Collections是个java.util下的工具类，它包含有各种有关集合操作的静态方法，主要是针对集合类的一个帮助类或者叫包装类，它提供一系列对各种集合的搜索，排序，线程安全化等操作方法。

1. **Comparable和Comparator接口是干什么的？列出它们的区别。**

Java 提供了只包含一个 compareTo()方法的 Comparable 接口。这个方法可以个给两个对象排序。具体来说，它返回负数，0，正数来表明输入对象小于，等于，大于已经存在的对象。

Java 提供了包含 compare()和 equals()两个方法的 Comparator 接口。compare()方法用来给两个输入参数排序，返回负数，0，正数表明第一个参数是小于，等于，大于第二个参数。equals()方法需要一个对象作为参数，它用来决定输入参数是否和 comparator 相等。只有当输入参数也是一个 comparator 并且输入参数和当前 comparator 的排序结果是相同的时候，这个方法才返回 true。

1. **集合中那些类是线程安全类，哪 些是不安全的，哪 些是支持排序的类**

线程安全类：Vector、Hashtable、Stack。

线程不安全的类：ArrayList、Linkedlist、HashSet、TreeSet、HashMap、TreeMap

支持排序的类有HashSet、LinkedHashSet、TreeSet等（Set接口下的实现都支持排序）

此题主要考查集合框架的知识。在集合框架中Collection接口为集合的根类型，提供集合操作的常用API方法，该接口下派生出两个子接口，一个是不支持排序的List接口，一个是有自身排序的Set接口，所以回答排序与不排序分别从两接口的实现中在作答。线程安全上来说，Vector类比同属于List接口的ArrayList要早，是一个线程安全的类，在JDK1.2以后才推出一个异步的ArrayList类，比Vector类效率高。同理Stack继承自Vector也线程安全的类，另外在在Map接口的实现在Hashtable也是个线程安全的类。

1. D
2. D
3. d
4. **IO**
5. **Java中有几种类型的流？JDK为每种类型的流提供了一些抽象类以供继承，请说出他们分别是哪些类？**

字节流，字符流两种类型流。

字节流继承于InputStream、OutputStream

字符流继承于Reader、Writer。

其它与IO操作相关的类都是派生至上述4个抽象类。

如字节相关的：FileInputStream、FileOutputStream类；

字符相关的：BufferedReader、BufferedWriter类

1. **什么是序列化，如何实现序列化？请解释Serializable接口的作用。**

我们有时候将一个java对象变成字节流的形式传出去或者从一个字节流中恢复成一个java对象，例如，要将java对象存储到硬盘或者传送给网络上的其他计算机， 这个过程我们可以自己写代码去把一个java对象变成某个格式的字节流再传输，但是jre本身就提供了这种支持，我们可以调用OutputStream的writeObject方法来做，如果要让java帮我们做，要被传输的对象必须实现serializable接口，这样javac编译时就会进行特殊处理，编译的类才可以被writeObject方法操作，这就是所谓的序列化。

需要被序列化的类必须实现Serializable接口，该接口是一个标记接口，其中没有需要实现的方法，implements Serializable只是为了标注该对象是可被序列化的。

例如，在web开发中，如果对象要经过分布式系统进行网络传输或通过rmi等远 程调用 ，这就需要在网络上传输对象，被传输的对象就必须实现Serializable接口。

1. **java里面的io跟nio有什么区别**
2. Java NIO和IO之间第一个最大的区别是，IO是面向流的，NIO是面向缓冲区的。
3. Java IO的各种流是阻塞的。而Java NIO的非阻塞模式

传统的阻塞式IO，每个连接必须要开一个线程来处理，并且没处理完线 程不能退出。   
 非阻塞式IO，由于基于反应器模式，用于事件多路分离和分派的体系结 构模式，所以可以利用线程池来处理。事件来了就处理，处理完了就把线 程归还。而传统阻塞方式不能使用线程池来处理，假设当前有10000个连 接，非阻塞方式可能用1000个线程的线程池就搞定了，而传统阻塞方式 就需要开10000个来处理。如果连接数较多将会出现资源不足的情况。非 阻塞的核心优势就在这里。   
 为什么会这样，下面就对他们做进一步细致具体的分析：   
 首先，我们来分析传统阻塞式IO的瓶颈在哪里。在连接数不多的情况下， 传统IO编写容易方便使用。但是随着连接数的增多，问题传统IO就不行 了。因为前面说过，传统IO处理每个连接都要消耗一个线程，而程序的 效率当线程数不多时是随着线程数的增加而增加，但是到一定的数量之后， 是随着线程数的增加而减少。这里我们得出结论，传统阻塞式IO的瓶颈 在于不能处理过多的连接。   
 然后，非阻塞式IO的出现的目的就是为了解决这个瓶颈。而非阻塞式IO 是怎么实现的呢？非阻塞IO处理连接的线程数和连接数没有联系，也就 是说处理 10000个连接非阻塞IO不需要10000个线程，你可以用1000 个也可以用2000个线程来处理。因为非阻塞IO处理连接是异步的。当某 个连接发送请求到服务器，服务器把这个连接请求当作一个请求"事件"， 并把这个"事件"分配给相应的函数处理。我们可以把这个处理函数放到线 程中去执行，执行完就把线程归还。这样一个线程就可以异步的处理多个 事件。而阻塞式IO的线程的大部分时间都浪费在等待请求上了。

1. 选择器上，Java IO无选择器，而NIO有选择器，Java NIO的选择器允许一个单独的线程来监视多个输入通道，你可以注册多个通道使用一个选择器，然后使用一个单独的线程来“选择”通道：这些通道里已经有可以处理的输入，或者选择已准备写入的通道。
2. D
3. D
4. D
5. d
6. **线程**
7. **创建线程方式**

实现多线程可以通过继承Thread类和实现Runnable接口。

(1)继承Thread

定义一个类继承Thread类

复写Thread类中的public void run()方法，将线程的任务代码封装到run 方法中

直接创建Thread的子类对象，创建线程

调用start()方法，开启线程(调用线程的任务run方法)

另外可以通过Thread的getName()获取线程的名称。

(2)实现Runnable接口；

定义一个类，实现Runnable接口；

覆盖接口的public void run()的方法，将线程的任务代码封装到run方法中；

创建Runnable接口的子类对象

将Runnabl接口的子类对象作为参数传递给Thread类的构造函数，创建 Thread类对象

（原因：线程的任务都封装在Runnable接口子类对象的run方法中。

所以要在线程对象创建时就必须明确要运行的任务）。

调用start()方法，启动线程。

两种方法区别：

(1)实现Runnable接口避免了单继承的局限性

(2)继承Thread类线程代码存放在Thread子类的run方法中

实现Runnable接口线程代码存放在接口的子类的run方法中；

在定义线程时，建议使用实现Runnable接口，因为几乎所有多线程都 可以使用这种方式实现

推荐使用线程池

1. **启动一个线程是用run()还是start()?**

启动一个线程是调用start()方法，使线程就绪状态，以后可以被调度为运行状态，一个线程必须关联一些具体的执行代码，run()方法是该线程所关联的执行代码。

1. **进程和线程的区别是什么？**

进程是执行着的应用程序，而线程是进程内部的一个执行序列。一个进程可以有多个线程。

线程又叫做轻量级进程。

1. [并发编程的3个概念：原子性、可见性、有序性](http://blog.csdn.net/u010796790/article/details/52155664)

并发程序正确地执行，必须要保证原子性、可见性以及有序性。只要有一个没有被保证，就有可能会导致程序运行不正确。

**原子性：**一个操作或多个操作要么全部执行完成且执行过程不被中断，要么就不执行。

**可见性：**当多个线程同时访问同一个变量时，一个线程修改了这个变量的值，其他线程能够立即看得到修改的值。

**有序性：**程序执行的顺序按照代码的先后顺序执行。

对于单线程，在执行代码时jvm会进行指令重排序，处理器为了提高效率，可以对输入代码进行优化，它不保证程序中各个语句的执行先后顺序同代码中的顺序一致，但是它会保证保存最终执行结果和代码顺序执行的结果是一致的。

Java语言对原子性、可见性、有序性的保证

1、**原子性**

在java中，对**基本数据类型的变量的读取和赋值操作是原子性操作**，即这些操作是不可被中断，要么执行，要么不执行。

X=10;  //原子性（简单的读取、将数字赋值给变量）

Y = x;  //变量之间的相互赋值，不是原子操作

X++;  //对变量进行计算操作

X = x+1;

语句2实际包括两个操作，它先要去读取x的值，再将y值写入，两个操作分开是原子性的。合在一起就不是原子性的。

语句3、4:x++  x=x+1包括3个操作：读取x的值，x+1，将x写入

注：可以通过 synchronized和Lock实现原子性。因为synchronized和Lock能够保证任一时刻只有一个线程访问该代码块。

2、**可见性**

Java提供了**volatile关键字**保证可见性。

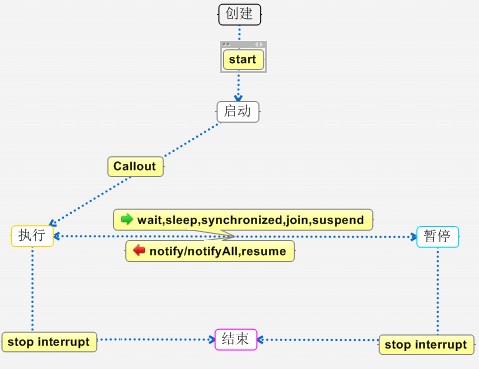
当一个共享变量被volatile修饰时，它会保证修改的值立即被其他的线程看到，即修改的值立即更新到主存中，当其他线程需要读取时，它会去内存中读取新值。

**Synchronized和Lock**也可以保证可见性，因为它们可以保证任一时刻只有一个线程能访问共享资源，并在其释放锁之前将修改的变量刷新到内存中，

 3、**有序性**

在Java里面，可以通过**volatile关键字**来保证一定的“有序性”。另外可以通过**synchronized和Lock**来保证有序性，很显然，synchronized和Lock保证每个时刻是有一个线程执行同步代码，相当于是让线程顺序执行同步代码，自然就保证了有序性。

1. **线程生命周期**



线程的生命周期:一个线程从它创建到启动，然后运行，直到最后执行完的整个过程。

* 新建状态:即创建一个新的线程对象，注意新创建的线程对象如果没有调 用start()方法将永远得不到运行。
* 就绪状态:当新的线程对象调用start()方法时，就进入了就绪状态，进入就 绪状态的线程不一定立即就开始运行。
* 运行状态:进入运行状态的线程，会由CPU处理其线程体中的代码。
* 阻塞状态:运行状态的线程有可能出现意外情况而中断运行,比如进行IO 操作，内存的读写，等待键盘输入数据(注意不是出错，出错将提前终止 线程)而进入阻塞状态。当阻塞条件解除后，线程会恢复运行。但其不是 立即进入运行状态，而是进入就绪状态。
* 终止状态:当线程中run()方法语句执行完后进入终止状态。

1. **sleep()和wait()区别**

sleep()方法是Thread类中方法，而wait()方法是Object类中的方法。  
 sleep()方法导致了程序暂停执行指定的时间，让出cpu该其他线程，但是 他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态，在 调用sleep()方法的过程中，线程不会释放对象锁。而当调用wait()方法的 时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对 象调用notify()方法后本线程才进入对象锁定池准备。

1. **stop()和suspend()方法为何不推荐使用？**

反对使用stop()，是因为它不安全。它会解除由线程获取的所有锁定，而且如果对象处于一种不连贯状态，那么其他线程能在那种状态下检查和修改它们。结果很难检查出真正的问题所在。suspend()方法容易发生死锁。调用suspend()的时候，目标线程会停下来，但却仍然持有在这之前获得的锁定。此时，其他任何线程都不能访问锁定的资源，除非被"挂起"的线程恢复运行。对任何线程来说，如果它们想恢复目标线程，同时又试图使用任何一个锁定的资源，就会造成死锁。所以不应该使用suspend()，而应在自己的Thread类中置入一个标志，指出线程应该活动还是挂起。若标志指出线程应该挂起，便用wait()命其进入等待状态。若标志指出线程应当恢复，则用一个notify()重新启动线程。

1. **互斥锁**

* 在Java中，引入对象互斥锁的概念，来保证共享数据操作的完整性。
* 每个对象都对应于一个可称为”互斥锁”的标记，这个标记用来保证在任 一时刻，只能有一个线程访问该对象。
* 关键字synchronized来与对象的互斥锁联系。当某个对象用synchronized 修饰时，表明该对象在任一时刻只能由一个线程访问。
* 尽量一要去对一个方法进行synchronized，对代码块来进行synchronized

1. **乐观锁和悲观锁**

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。

1. **线程死锁**

产生死锁的原因：  
一.因为系统资源不足。  
二.进程运行推进的顺序不合适。  
三.资源分配不当。

并发运行的多个线程间彼此等待、都无法运行的状态称为线程死锁。(并发运行的线程分别占有对方需要的资源，但又都不肯首先释放自己占用的资源，这样导致并发的线程无法继续运行下去，而进入阻塞状态)

为避免死锁，在线程进入阻塞状态时应尽量释放其锁定的资源，以为其他的线程提供运行的机会。

* public final void wait()
* public final void notify() //唤醒一个指定的线程
* public final void notifyAll()//唤醒所有等待的线程

死锁的预防

1：按照特定顺序加锁

2：代码双重判断检测死锁

3：给锁对象赋值唯一对象值判断

1. **如何确保N个线程可以访问 N个资源同时又不导致死锁？**

使用多线程的时候，一种非常简单的避免死锁的方式就是：指定获取锁的顺序， 并强制线程按照指定的顺序获取锁。因此，如果所有的线程都是以同样的顺序 加锁和释放锁，就不会出现死锁了

1. **如何停止一个线程**

停止一个线程意味着在任务处理完任务之前停掉正在做的操作，也就是放弃当 前的操作。停止一个线程可以用Thread.stop()方法，但最好不要用它。虽然它 确实可以停止一个正在运行的线程，但是这个方法是不安全的，而且是已被废 弃的方法。  
 在java中有以下3种方法可以终止正在运行的线程：

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。

使用stop方法强行终止，但是不推荐这个方法，因为stop和suspend及resume 一样都是过期作废的方法。

使用interrupt方法中断线程。

interrupt()方法的使用效果并不像for+break语句那样，马上就停止循环。调用interrupt方法是在当前线程中打了一个停止标志，并不是真的停止线程。

1. **多线程间通讯**

多线程间通讯就是多个线程在操作同一资源,但是操作的动作不同.

(1)为什么要通信

多线程并发执行的时候, 如果需要指定线程等待或者唤醒指定线程, 那么就需要通信.比如生产者消费者的问题，生产一个消费一个,生产的时候需要负责消费的进程等待,生产一个后完成后需要唤醒负责消费的线程,同时让自己处于等待，消费的时候负责消费的线程被唤醒，消费完生产的产品后又将等待的生产线程唤醒，然后使自己线程处于等待。这样来回通信，以达到生产一个消费一个的目的。

(2)怎么通信

在同步代码块中, 使用锁对象的wait()方法可以让当前线程等待, 直到有其他线程唤醒为止.使用锁对象的notify()方法可以唤醒一个等待的线程，或者notifyAll唤醒所有等待的线程.

多线程间通信用sleep很难实现，睡眠时间很难把握。

/\*

线程通讯： 一个线程完成了自己的任务时，要通知另外一个线程去完成另外一个任务.

生产者与消费者

wait(): 等待 如果线程执行了wait方法，那么该线程会进入等待的状态，等待状态下的线程必须要被其他线程调用notify方法才能唤醒。

notify()： 唤醒 唤醒线程池等待线程其中的一个。

notifyAll() : 唤醒线程池所有等待 线程。

wait与notify方法要注意的事项：

1. wait方法与notify方法是属于Object对象 的。

2. wait方法与notify方法必须要在同步代码块或者是同步函数中才能 使用。

3. wait方法与notify方法必需要由锁对象调用。

问题一：出现了线程安全问题。 价格错乱了...苹果价格为2.0

生产者获得cpu，i=0生产苹果，价格6.5，当生产者继续获得cpu,i=1生产香蕉，价格为2.0, 此时P对象的名字是香蕉，价格是2.0,

当生产者继续获得cpu,i=3生产苹果，当刚执行完p.name="苹果", 还没有来的及执行p.price=6.5时，消费者获得cpu运行，导致此时p的name="苹果",但价格还是2.0

\*/

**class** Product{

String name; //名字

**double** price; //价格

**boolean** flag = **false**; //产品是否生产完毕的标识，默认情况是没有生产完成。

}

//因为生产者和消费者共享一个产品，所以他们是线程类。

//生产者类

**class** Producer **extends** Thread{

Product p ; //产品

**public** Producer(Product p) {

**this**.p = p ;

}

@Override

**public** **void** run() {

**int** i = 0 ; //为产生不同的产品

**while**(**true**){

**synchronized** (p) { //让生产者与消费者是同一个对象

**if**(p.flag==**false**){

**if**(i%2==0){

p.name = "苹果";

p.price = 6.5;

}**else**{

p.name="香蕉";

p.price = 2.0;

}

System.***out***.println("生产者生产出了："+ p.name+" 价格是："+ p.price);

p.flag = **true**;

i++;

p.notifyAll(); //唤醒消费者去消费

}**else**{

//已经生产 完毕，等待消费者先去消费

**try** {

p.wait(); //生产者等待,并释放锁对象

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

}

}

//消费者

**class** Customer **extends** Thread{

Product p;

**public** Customer(Product p) {

**this**.p = p;

}

@Override

**public** **void** run() {

**while**(**true**){

**synchronized** (p) { //利用p对象做为锁对象，p是相同的

**if**(p.flag==**true**){ //产品已经生产完毕

System.***out***.println("消费者消费了"+p.name+" 价格："+ p.price);

p.flag = **false**;

p.notifyAll(); // 唤醒生产者去生产

}**else**{

//产品还没有生产,应该 等待生产者先生产。

**try** {

p.wait(); //消费者也等待了...

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

}

}

}

}

**public** **class** Demo9 {

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

Product p = **new** Product(); //产品

//创建生产对象

Producer producer = **new** Producer(p);

//创建消费者

Customer customer = **new** Customer(p);

//调用start方法开启线程

producer.start();

customer.start();

}

}

1. **多线程安全问题**

原因:当程序的多条语句在操作线程共享数据时(如买票例子中的票就是共享资源)，由于线程的随机性导致一个线程对多条语句，执行了一部分还没执行完，另一个线程抢夺到cpu执行权参与进来执行，此时就导致共享数据发生错误。比如买票例子中打印重票和错票的情况。

解决方法:

同步是用来解决多线程的安全问题的，在多线程中，同步能控制对共享数据的访问。如果没有同步，当一个线程在修改一个共享数据时，而另外一个线程正在使用或者更新同一个共享数据，这样容易导致程序出现错误的结果。

1. **同步和异步有何异同，在什么情况下分别使用他们？举例说明。**

同步是指所有操作串行化执行，顺序不能改变，前一操作未完成，后个操作不执行。

异步是指所有操作可以并行执行，顺序无关。

例如寄信

同步：如果没有寄完，不能吃饭，邮递员10天后送到，发送人被饿死

异步：寄出后可以立即吃饭，邮递员送完后，通知发送人送信结果。

如果强调执行顺序的话，用同步。如果顺序无关，则可以用异步。

异步执行效率比同步高。

1. **多线程有几种实现方法?同步有几种实现方法?**

多线程有两种实现方法，分别是继承Thread类与实现Runnable接口

同步的实现方面有五种，分别是synchronized、wait与notify、sleep、suspend、join

synchronized: 一直持有锁，直至执行结束

wait():使一个线程处于等待状态，并且释放所持有的对象的lock，需捕获异常。

sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，需捕获异常，不释放锁。

notify():唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由JVM确定唤醒哪个线程，而且不是按优先级。

notityAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争。

1. **synchronized如何使用**

synchronized是Java中的关键字，是一种同步锁。它修饰的对象有以下几种：  
1). 修饰一个代码块，被修饰的代码块称为同步语句块，其作用的范围是大括号{}括起来的代码，作用的对象是调用这个代码块的对象；  
2). 修饰一个方法，被修饰的方法称为同步方法，其作用的范围是整个方法，作用的对象是调用这个方法的对象；  
3). 修改一个静态的方法，其作用的范围是整个静态方法，作用的对象是这个类的所有对象；  
4). 修改一个类，其作用的范围是synchronized后面括号括起来的部分，作用主的对象是这个类的所有对象。

1. **简述synchronized和java.util.concurrent.locks.Lock的异同？**

主要相同点：Lock能完成synchronized所实现的所有功能  
主要不同点：Lock有比synchronized更精确的线程语义和更好的性能。Lock的锁定是通过代码实现的，而synchronized是在JVM层面上实现的，synchronized会自动释放锁，而Lock一定要求程序员手工释放，并且必须在finally从句中释放。Lock还有更强大的功能，例如，它的tryLock方法可以非阻塞方式去拿锁。Lock锁的范围有局限性，块范围，而synchronized可以锁住块、对象、类。

1. **什么是线程池（thread pool）**

答：在面向对象编程中，创建和销毁对象是很费时间的，因为创建一个对象要获取内存资源或者其它更多资源。在Java中更是如此，虚拟机将试图跟踪每一个对象，以便能够在对象销毁后进行垃圾回收。所以提高服务程序效率的一个手段就是尽可能减少创建和销毁对象的次数，特别是一些很耗资源的对象创建和销毁，这就是”池化资源”技术产生的原因。线程池顾名思义就是事先创建若干个可执行的线程放入一个池（容器）中，需要的时候从池中获取线程不用自行创建，使用完毕不需要销毁线程而是放回池中，从而减少创建和销毁线程对象的开销。

Java线程池的构造方法，里面参数的含义? ThreadPoolExecutor是线程池的真正实现，参数有:corePoolSize（核心线程数）、maximumPoolSize（最大线程数）、keepAliveTime（闲置超时时间）等

Java 5+中的Executor接口定义一个执行线程的工具。它的子类型即线程池接口是ExecutorService。要配置一个线程池是比较复杂的，尤其是对于线程池的原理不是很清楚的情况下，因此在工具类Executors面提供了一些静态工厂方法，生成一些常用的线程池，如下所示：

- newSingleThreadExecutor：创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。

- newFixedThreadPool：创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

- newCachedThreadPool：创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，那么就会回收部分空闲（60秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说JVM）能够创建的最大线程大小。

- newScheduledThreadPool：创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。

- newSingleThreadExecutor：创建一个单线程的线程池。此线程池支持定时以及周期性执行任务的需求。

1. **什么是ThreadLocal类,怎么使用它？**

ThreadLocal类提供了线程局部 (thread-local) 变量。是一个线程级别的局部变量，并非“本地线程”。

线程局部变量(ThreadLocal)其实功用非常简单，就是为每一个使用该变量的线程都提供一个变量值的副本，是每一个线程都可以独立地改变自己的副本，而不会和其它线程的副本冲突。从线程的角度看，就好像每一个线程都完全拥有一个该变量。

应用场景:[用ThreadLocal 来管理 Hibernate Session](http://www.iteye.com/problems/5218)

我们知道Session是由SessionFactory负责创建的，而SessionFactory的实现是线程安全的，多个并发的线程可以同时访问一个SessionFactory并从中获取Session实例，那么Session是否是线程安全的呢？答案是否定的。Session中包含了数据库操作相关的状态信息，那么说如果多个线程同时使用一个Session实例进行CRUD，就很有可能导致数据存取的混乱

  使用ThreadLocal集合保存当前业务线程中的SESSION  
     private static ThreadLocal session = new ThreadLocal();

ThreadLocal底层是一个Map集合，key是当前线程对象，value是session的副本。

线程局部变量是局限于线程内部的变量，属于线程自身所有，不在多个线程间共享。Java 提供 ThreadLocal 类来支持线程局部变量，是一种实现线程安全的方式。但是在管理环境下（如 web 服务器）使用线程局部变量的时候要特别小心，在这种情况下，工作线程的生命周期比任何应用变量的生命周期都要长。任何线程局部变量一旦在工作完成后没有释放，Java 应用就存在内存泄露的风险。

1. **假如新建T1、T2、T3三个线程，如何保证它们按顺序执行？**

T3先执行，在T3的run中，调用t2.join，让t2执行完成后再执行t3

在T2的run中，调用t1.join，让t1执行完成后再让T2执行

public class Test {

//现在有T1、T2、T3三个线程，你怎样保证T2在T1执行完后执行， T3在T2执行完后执行

public static void main(String[] args) {

final Thread t1 = new Thread(new Runnable() {

@Override

public void run() {

System.out.println("t1");

}

});

final Thread t2 = new Thread(new Runnable() {

@Override

public void run() {

try {

//引用t1线程，等待t1线程执行完

t1.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("t2");

}

});

Thread t3 = new Thread(new Runnable() {

@Override

public void run() {

try {

//引用t2线程，等待t2线程执行完

t2.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("t3");

}

});

t3.start();

t2.start();

t1.start();//这三个线程启动没有先后顺序的

}

}

1. D
2. d
3. **网络**
4. **常用网络通信协议**

* TCP/IP协议

TCP传输控制协议:TCP协议是一种可靠的端对端协议，重发一切没有收到 的数据，进行数据内容准确性检查并保证分组的正确顺序。

IP网际协议:规定数据传输格式

* HTTP协议/HTTPS

HTTP超文本传输协议,基于请求和响应模式。

* FTP协议

文件传输协议

* SMTP协议

简单邮件传输协议

* POP3/IMAP协议

POP3邮局协议版本3

IMAP:Internet消息访问协议

1. **SOCKET中有几中连接方式，各有什么区别？**

Sockets有两种主要的操作方式:面向连接(TCP/IP)的和无连接(UDP)的。

无连接的操作使用数据报协议，无连接的操作是快速的和高效的,但是数据安全性不佳.

面向连接的操作使用TCP协议.面向连接的操作比无连接的操作效率更低,但是数据的安全性更高

二者最大区别:TCP是面向连接的,而UDP是无连接的.区别大致如下:

1）UDP传送的数据单位协议是UDP报文或用户数据报,TCP传送的数据单 位协议是TCP报文段。

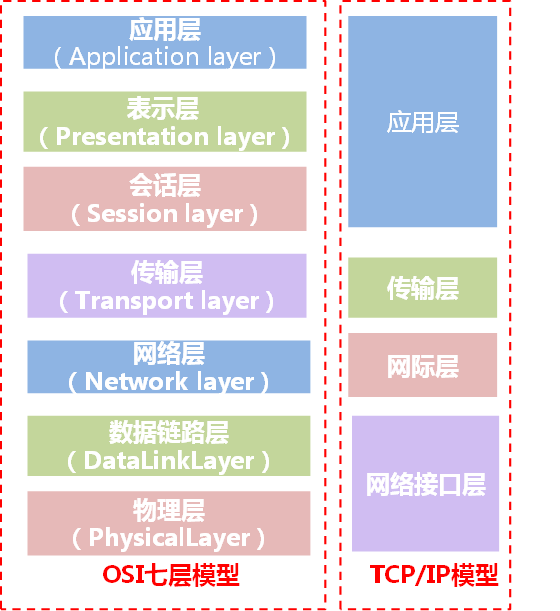
2）UDP发送数据之前不需要建立连接,因此减少了开销和发送之前的时延。 TCP提供面向连接的服务,不提供广播或多播服务。

3）对方的运输层在收到UDP报文后,不需要给出任何确认。TCP则需要确 认。

4）UDP没有拥塞控制,因此网络出现的拥塞不会使源主机的发送速率降低, 也不保证可靠交付,因此主机不需要维持具有许多参数的、复杂的连接状 态表。TCP要提供可靠的、面向连接的运输服务,因此不可避免地增加了许 多的开销,这不仅使协议数据单元的首部增大很多,还要占用许多的处理机 资源。

5）UDP用户数据报只有8个字节的首部开销,比TCP的20个字节的首部 要短。

1. OSI七层模型



应用层：就是应用软件使用的协议，如邮箱使用的POP3，SMTP、远程登录使用的Telnet、获取IP地址的DHCP、域名解析的DNS、网页浏览的http协议等；这部分协议主要是规定应用软件如何去进行通信的。

表示层：决定数据的展现（编码）形式，如同一部电影可以采样、量化、编码为RMVB、AVI，一张图片能够是JPEG、BMP、PNG等。

会话层：为两端通信实体建立连接（会话），中间有认证鉴权以及检查点记录（供会话意外中断的时候可以继续，类似断点续传）。

传输层：将一个数据/文件斩件分成很多小段，标记顺序以被对端接收后可以按顺序重组数据，另外标记该应用程序使用的端口号及提供QOS。（不同的应用程序使用不同计算机的端口号，同样的应用程序需要使用一样的端口号才能正常通信）

网络层：路由选路，选择本次通信使用的协议（http、ftp等），指定路由策略及访问控制策略。（IP地址在这一层）

数据链路层：根据端口与MAC地址，做分组（VLAN）隔离、端口安全、访问控制。（MAC地址在这一层）处理VLAN内的数据帧转发，跨VLAN间的访问，需要上升到网络层。

物理层：将数据最终编码为用0、1标识的比特流，然后传输。（例如将题主头像的图片，变为一串01100111100这样的数字来表示）。

基础理论说完,看个实例

两主机通信的过程，从发送者（以下简称A）到接收者（以下简称B），属于从7层（应用层）->1层（物理层）封装，然后传输到远端，再从1层（物理层）->7层（应用层）解封装的过程。

1、 当A打开了QQ这个软件，相当就到达应用层了；因为软件会根据你的操作调动机器底层的硬件工作了。

2、 当A往QQ这个软件的聊天窗口里面输入信息，发出后，QQ会将这个信息保存在本地聊天记录文件MSGEX.db（一般就保存在QQ目录下以你的QQ号码为文件夹里）。以某种格式编码/保存某种信息，这可以理解为表示层了。

3、 当A打开与B的聊天窗口，输入信息，按下“输入”按钮，用户的操作就完结了，剩下都是机器自己的操作了。实际传输之前QQ会先建立A与B的会话连接，才真正开始传输信息/数据（你可以理解借传输文件理解：你发送文件给对方，要等待对方按下接收，才算建立了会话，然后才开始传输。）这算会话层了。

4、 会话建立后，会将A发的信息斩件，如A发送“你吃了饭没有”？传输层将这句话斩成“你”“吃”“了”“饭”“没”“有”6个数据段，标记号使用的端口号，然后准备发出去。

5、 接上一层，信息还未发出去，这时候在网络层做路由选路，可以理解为，从A家出去，可以分别经“联通”“电信”“移动”3个网络中的一个再到B家。

网络层根据路由协议负责选路（根据链路质量、带宽、开销等方法论）。假设最后选了2条，可能就A->联通->B发送“你”“吃”“了”3个数据段，A->电信->B发送“饭”“没”“有”3个数据段。

选路后，这一层要标记IP包头，包头主要内容是源IP地址，目的IP地址，使用什么协议。其中源、目的IP相当于你寄信的时候的收发的地址与邮政编码，标记出发送者与接收者。而协议相当于这封信到底用什么语言书写。（只有保证2端使用同种语言，才能确保通信起来，否则你用英文写信给大妈，大妈怎么看得懂呢？）

6、 然后再到数据链路层，数据链路层主要是负责同一个子网内的通信的。例如A、B连接在同一台二层交换机，就属于同一个子网，那么数据帧的通信室是不需要通过网络层的（即三层交换机或者路由器），直接在这台二层交换机就过去了。这一层打的是MAC地址的帧头，对于上述通信过程来说，就是为数据帧打上A的机器的MAC与A的网关的MAC。这一层的工作就完成了。

7、 最后一层了，经过上述斩件、打完各层标签后的6个数据帧，物理层将他们翻译文6段0、1表示的比特流，然后通过光纤、铜缆进行传输。

8、 当比特流传输到了远端，接着B的机器按照上述的1~7的步骤反方向运行一次即可（即有物理层到应用层）。就是一层层读取标签，传输给标签标记着的相应对象，然后摘除标签，再读取上一层标签，直到最后B的应用程序能够读到A往应用程序输入的数据为止。

1. [**TCP的三次握手过程**](http://blog.csdn.net/sssnmnmjmf/article/details/68486261)**和四次挥手**

TCP报文格式

  
上图中有几个字段需要重点介绍下：  
        （1）序号：Seq序号，占32位，用来标识从TCP源端向目的端发送的字节流，发起方发送数据时对此进行标记。  
        （2）确认序号：Ack序号，占32位，只有ACK标志位为1时，确认序号字段才有效，Ack=Seq+1。  
        （3）标志位：共6个，即URG、ACK、PSH、RST、SYN、FIN等，

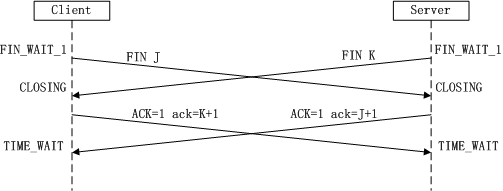
具体含义如下：  
               （A）URG：紧急指针（urgent pointer）有效。  
                （B）ACK：确认序号有效。  
                （C）PSH：接收方应该尽快将这个报文交给应用层。  
                （D）RST：重置连接。  
                （E）SYN：发起一个新连接。  
                （F）FIN：释放一个连接。  
**三次握手**  
        所谓三次握手（Three-Way Handshake）即建立TCP连接，就是指建立一个TCP连接时，需要客户端和服务端总共发送3个包以确认连接的建立。在socket编程中，这一过程由客户端执行connect来触发，整个流程如下图所示：

  
TCP三次握手

        （1）第一次握手：Client将标志位SYN置为1，随机产生一个值seq=J，并将该数据包发送给Server，Client进入SYN\_SENT状态，等待Server确认。  
        （2）第二次握手：Server收到数据包后由标志位SYN=1知道Client请求建立连接，Server将标志位SYN和ACK都置为1，ack=J+1，随机产生一个值seq=K，并将该数据包发送给Client以确认连接请求，Server进入SYN\_RCVD状态。  
        （3）第三次握手：Client收到确认后，检查ack是否为J+1，ACK是否为1，如果正确则将标志位ACK置为1，ack=K+1，并将该数据包发送给Server，Server检查ack是否为K+1，ACK是否为1，如果正确则连接建立成功，Client和Server进入ESTABLISHED状态，完成三次握手，随后Client与Server之间可以开始传输数据了。  
**四次挥手**  
       所谓四次挥手（Four-Way Wavehand）即终止TCP连接，就是指断开一个TCP连接时，需要客户端和服务端总共发送4个包以确认连接的断开。在socket编程中，这一过程由客户端或服务端任一方执行close来触发，整个流程如下图所示：

 TCP四次挥手

        由于TCP连接时全双工的，因此，每个方向都必须要单独进行关闭，这一原则是当一方完成数据发送任务后，发送一个FIN来终止这一方向的连接，收到一个FIN只是意味着这一方向上没有数据流动了，即不会再收到数据了，但是在这个TCP连接上仍然能够发送数据，直到这一方向也发送了FIN。首先进行关闭的一方将执行主动关闭，而另一方则执行被动关闭，上图描述的即是如此。  
        （1）第一次挥手：Client发送一个FIN，用来关闭Client到Server的数据传送，Client进入FIN\_WAIT\_1状态。  
        （2）第二次挥手：Server收到FIN后，发送一个ACK给Client，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号），Server进入CLOSE\_WAIT状态。  
        （3）第三次挥手：Server发送一个FIN，用来关闭Server到Client的数据传送，Server进入LAST\_ACK状态。  
        （4）第四次挥手：Client收到FIN后，Client进入TIME\_WAIT状态，接着发送一个ACK给Server，确认序号为收到序号+1，Server进入CLOSED状态，完成四次挥手。  
        上面是一方主动关闭，另一方被动关闭的情况，实际中还会出现同时发起主动关闭的情况，具体流程如下图：



1. **什么是长连接、短连接？**

在HTTP/1.0中，默认使用的是短连接。也就是说，浏览器和服务器每进行一次HTTP操作，就建立一次连接，但任务结束就中断连接。如果客户端浏览器访问的某个HTML或其他类型的 Web页中包含有其他的Web资源，如JavaScript文件、图像文件、CSS文件等；当浏览器每遇到这样一个Web资源，就会建立一个HTTP会话。

但从 HTTP/1.1起，默认使用长连接，用以保持连接特性。使用长连接的HTTP协议，会在响应头有加入这行代码：

Connection:keep-alive

在使用长连接的情况下，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的 TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache）中设定这个时间。实现长连接要客户端和服务端都支持长连接。

HTTP协议的长连接和短连接，实质上是TCP协议的长连接和短连接。

 长连接和短连接的优点和缺点

由上可以看出，长连接可以省去较多的TCP建立和关闭的操作，减少浪费，节约时间。对于频繁请求资源的客户来说，较适用长连接。不过这里存在一个问题，存活功能的探测周期太长，还有就是它只是探测TCP连接的存活，属于比较斯文的做法，遇到恶意的连接时，保活功能就不够使了。在长连接的应用场景下，client端一般不会主动关闭它们之间的连接，Client与server之间的连接如果一直不关闭的话，会存在一个问题，随着客户端连接越来越多，server早晚有扛不住的时候，这时候server端需要采取一些策略，如关闭一些长时间没有读写事件发生的连接，这样可 以避免一些恶意连接导致server端服务受损；如果条件再允许就可以以客户端机器为颗粒度，限制每个客户端的最大长连接数，这样可以完全避免某个客户端连累后端服务。

短连接对于服务器来说管理较为简单，存在的连接都是有用的连接，不需要额外的控制手段。但如果客户请求频繁，将在TCP的建立和关闭操作上浪费时间和带宽。

长连接和短连接的产生在于client和server采取的关闭策略，具体的应用场景采用具体的策略，没有十全十美的选择，只有合适的选择。

1. D
2. d
3. **高级问题**
4. **JVM优化**

1.升级JVM版本。如果能使用64-bit，使用64-bitJVM。

2.选择一个正确的GC（GargageCollection）。

由于当JAVA程序GC的时候，会停下当前程序。尤其FullGC的时候，会停留很长时间。一般对于GUI程序来说，是很难接受的。JAVA5以后，开始自带了好几种GC，你可以选择一个适合你的种类。有以下四种SerialCollector，Parallelcollector(推荐使用并行收集)，ConcurrentCollector。

3.正确设置内存大小。进行JVM调优时对JVM堆内的各个区域（young,old,perm）正确设置大小。

通常使用一下几个参数调整-Xms-Xmx-XX:MaxPermSize。

3.1调高-XX:NewRatio（NewSize/MaxNewSize）的值，会减少younggc的次数，但会增加oldgc的时间。

3.2增加普通GC的方法（减小FullGC）。扩大young区域的大小（最大40%），并过大Survivor的区域。使得更多的object留在younggen。

4.以下是几个写程序时，应该注意的地方。也可减小GC，提高JVM性能。

1.不要使用System.gc()方法。

因为它会产生FullGC。

2.尽可能少分配大的临时对象（生命周期短的）

可能会直接分配到old区域里，old区域只有FullGC的时候会收集。

3.避免使用finalize()方法。

finalize()会增加GC的负担，使用java.lang.ref代替。

1. **JVM内存溢出问题**

最近在熟悉一个开发了有几年的项目，需要把数据库从mysql移植到oracle，首先把jdbc的连接指向mysql，打包放到tomcat里面，可以跑起来，没有问题，可是当把jdbc连接指向oracle的时候，tomcat就连续抛java.lang.OutOfMemoryError的错误。

1、JVM内存溢出现象提示一：java.lang.OutOfMemoryError:Javaheapspace

解释：

Heapsize设置

JVM堆的设置是指java程序运行过程中JVM可以调配使用的内存空间的设置.JVM在启动的时候会自动设置Heapsize的值，其初始空间(即-Xms)是物理内存的1/64，最大空间(-Xmx)是物理内存的1/4。可以利用JVM提供的-Xmn-Xms-Xmx等选项可进行设置。Heapsize的大小是YoungGeneration和TenuredGeneraion之和。

提示：在JVM中如果98％的时间是用于GC且可用的Heapsize不足2％的时候将抛出此异常信息。

提示：HeapSize最大不要超过可用物理内存的80％，一般的要将-Xms和-Xmx选项设置为相同，而-Xmn为1/4的-Xmx值。

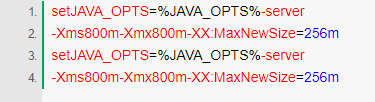
解决方法：

手动设置Heapsize

修改

TOMCAT\_HOME/bin/catalina.bat

在“echo"UsingCATALINA\_BASE:$CATALINA\_BASE"”上面加入以下行：

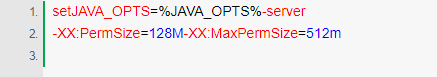


**2、JVM内存溢出现象提示二：java.lang.OutOfMemoryError:PermGenspace**

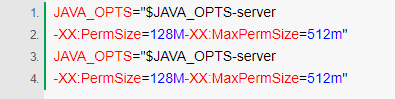
原因：  
PermGenspace的全称是PermanentGenerationspace,是指内存的永久保存区域，这块内存主要是被JVM存放Class和Meta信息的,Class在被Loader时就会被放到PermGenspace中，它和存放类实例(Instance)的Heap区域不同,GC(GarbageCollection)不会在主程序运行期对PermGenspace进行清理，所以如果你的应用中有很多CLASS的话,就很可能出现PermGenspace错误，这种错误常见在web服务器对JSP进行precompile的时候。如果你的WEBAPP下都用了大量的第三方jar,其大小超过了jvm默认的大小(4M)那么就会产生此错误信息了。

解决方法：**手动设置MaxPermSize大小**

修改TOMCAT\_HOME/bin/catalina.bat（Linux下为catalina.sh），在Java代码  
“echo"UsingCATALINA\_BASE:$CATALINA\_BASE"”上面加入以下行：



catalina.sh下为：



1. **泛型**

从Java SE5.0开始引入，其实质是将原本确定不变的数据类型参数化。

为什么会出现泛型？

因为集合存放的数据类型不固定，故往集合里面存放元素时，存在安全隐 患，如果在定义集合时，可以想定义数组一样指定数据类型，那么就可以 解决该类安全问题。JDK1.5后出现了泛型，用于解决集合框架的安全问题。

泛型是一个类型安全机制。

泛型优点:作为对原有Java类型体系的扩充，使用泛型可以提高Java应用程序的类型安全、可维护性和可靠性。范型类，范型方法

1. **枚举**

一些方法在运行时，它需要的数据不能是任意的，而必须是一定范围内的值， Java5以后可以直接使用枚举予以解决。

* 可以这样来定义一个枚举类型。

public　enum　Color {

Red,

White,

Blue

}

* 可以这样使用

Color　myColor　=　Color.Red

1. **设计模式**

单态模式、工厂模式、MVC、代理模式（spring中的AOP功能）、DAO、

观察者模式(发布-订阅模式)：定义了一种一对多的依赖关系，让多个观察者对象同事监听某一个主题对象。这个主题对象在状态发生变化时，会通知所有观察者对象，使他们能够自动更新自己。

工厂模式：该模式将创建对象的过程放在了一个静态方法中来实现.在实际编程中,如果需要大量的创建对象,该模式是比较理想的。

1. **反射技术**

类字节码文件是在硬盘上存储的，是一个个的.class文件。我们在new一个对象时，JVM会先把字节码文件的信息读出来放到内存中，第二次用时，就不用在加载了，而是直接使用之前缓存的这个字节码信息。

字节码的信息包括：类名、声明的方法、声明的字段等信息。在Java中“万物皆对象”，这些信息当然也需要封装一个对象，这就是Class类、Method类、Field类。

通过Class类、Method类、Field类等类可以得到这个类型的一些信息，甚至可以不用new关键字就创建一个实例，可以执行一个对象中的方法，设置或获取字段的值，这就是反射技术。

举例:在spring中，生成对象时就是使用反射技术

如何通过反射创建对象？

方法1：通过类对象调用newInstance()方法，例如：String.class.newInstance()

方法2：通过类对象的getConstructor()或getDeclaredConstructor()方法获得构造器（Constructor）对象并调用其newInstance()方法创建对象，例如：String.class.getConstructor(String.class).newInstance(“Hello”);

1. **Junit单元测试框架的基本使用**

软件测试有很多分类

从测试的方法上可分为：黑盒测试、白盒测试、静态测试、动态测试等；

从软件开发的过程分为：单元测试、集成测试、确认测试、验收、回归等。

在众多的分类中，与开发人员关系最紧密的莫过于单元测试了。像其他种类的测试基本上都是由专门的测试人员来完成，只有单元测试是完全由开发人员来完成的。那么今天我们就来说说什么是单元测试，为什么要进行单元测试，以及如更好的何进行单元测试。

单元测试（unit testing），是指对软件中的最小可测试单元进行检查和验证。比如我们可以测试一个类，或者一个类中的一个方法。

1. **如何实现对象克隆？**

有两种方式：

1)实现Cloneable接口并重写Object类中的clone()方法；

2)实现Serializable接口，通过对象的序列化和反序列化实现克隆，可以 实现真正的深度克隆

1. **Java对XML文件的处理方式**

DOM解析和SAX解析

DOM解析原理：xml解析器一次性把整个xml文档加载进内存，然后在内存中构 建一棵Document的对象树，通过Document对象，得到树上的节点对象，通过节 点对象访问（操作）到xml文档的内容。（DOM4J）

DOM解析原理：一次性把xml文档加载进内存，然后在内存中构建Document树。

对内存要求比较要。

缺点： 不适合读取大容量的xml文件，容易导致内存溢出。

SAX解析原理： 加载一点，读取一点，处理一点。对内存要求比较低。

XML是一种重量级的数据交换格式。JSON是轻量级数据交换格式。

|  |  |
| --- | --- |
| **DOM解析** | **SAX解析** |
| 原理： 一次性加载xml文档，不适合大容量的文件读取 | 原理： 加载一点，读取一点，处理一点。适合大容量文件的读取 |
| DOM解析可以任意进行增删改成 | SAX解析只能读取 |
| DOM解析任意读取任何位置的数据，甚至往回读 | SAX解析只能从上往下，按顺序读取，不能往回读 |
| DOM解析面向对象的编程方法（Node，Element，Attribute）,Java开发者编码比较简单。 | SAX解析基于事件的编程方法。java开发编码相对复杂。 |

1. **你在项目中哪些地方用到了XML？**

答：XML的主要作用有两个方面：数据交换和信息配置。在做数据交换时，XML将数据用标签组装成起来，然后压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再从XML文件中还原相关信息进行处理，XML曾经是异构系统间交换数据的事实标准，但此项功能几乎已经被JSON（JavaScript Object Notation）取而代之。当然，目前很多软件仍然使用XML来存储配置信息，我们在很多项目中通常也会将作为配置信息的硬代码写在XML文件中，Java的很多框架也是这么做的，而且这些框架都选择了dom4j作为处理XML的工具。

1. **你了解大O符号(big-O notation) 么？你能给出不同数据结构的例子么？**

此问题是问算法的时间复杂度

大 O 符号描述了当数据结构里面的元素增加的时候，算法的规模或者是性能在最坏的场景下有多么好。

大 O 符号也可用来描述其他的行为，比如：内存消耗。因为集合类实际上是数据结构，我们一般使用大 O 符号基于时间，内存和性能来选择最好的实现。大 O 符号可以对大量数据的性能给出一个很好的说明。

容易计算的方法是：看看有几重for循环，只有一重则时间复杂度为O(n)，二重则为O(n^2)，依此类推，如果有二分则为O(logn)，二分例如[快速幂](https://baike.baidu.com/item/%E5%BF%AB%E9%80%9F%E5%B9%82" \t "_blank)、二分查找，如果一个for循环套一个二分，那么时间复杂度则为O(nlogn)。

1. **Java注解及实现原理**

一.什么是注解：

注解是标记，也可以理解成是一种应用在类、方法、参数、属性、构造器上的特殊修饰符。注解作用有以下三种：

第一种：生成文档，常用的有@param@return等。

第二种：替代配置文件的作用，尤其是在spring等一些框架中，使用注解可以大量的减少配置文件的数量。

第三种：检查代码的格式，如@Override，标识某一个方法是否覆盖了它的父类的方法。

二.注解的底层实现原理：

注解的底层也是使用反射实现的，我们可以自定义一个注解来体会下。注解和接口有点类似，不过申明注解类需要加上@interface，注解类里面，只支持基本类型、String及枚举类型，里面所有属性被定义成方法，并允许提供默认值。

java 5.0开始，在java.lang.annotations中提供了四种元注解，专门注解其他的注解：

@Target? — —注解用于什么地方

TYPE, //给类（型）注解

FIELD, //给字段注解，不要忘了，字段可以是对象

METHOD, //给方法注解

PARAMETER, //给参数注解

CONSTRUCTOR, //给构造方法注解

LOCAL\_VARIABLE, //给局部变量注解

ANNOTATION\_TYPE,//给注解注解（这貌似把自己不当类来看）

PACKAGE, //给包注解

@Retention — —注解运行状态

SOURCE, //源码状态运行，

CLASS, //编译类文件时运行

RUNTIME //运行时运行

@Documented — — 生成说明文档，添加类的解释

@Inherited — —允许子类继承父类中的注解。

我们自定义一个注解来感受下：

定义注解类：

[java] view plain copy

@Target({ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Documented

public @interface User {

String name() default "张三";

}

由于我们的注解是类注解，所以我们创建一个类

[java] view plain copy

@User

public class test {

}

到了这里，注解可能还没起什么作用。我们需要通过反向代理去读取类中定义的注解，读出来使用就简单了。

创建测试类，来读取：

[java] view plain copy

public class testamin {

public static void main(String[] args) throws ClassNotFoundException{

Class<?> classTest=Class.forName("com.test.test");

Annotation[] ann=classTest.getAnnotations();

for(Annotation aa:ann){

User u=(User)aa;

System.out.println(u.name());

}

}

}

1. **Java8新特性**

1）Lambda表达式和函数式接口

Lambda表达式（也称为闭包）是Java 8中最大和最令人期待的语言改变。它允许我们将函数当成参数传递给某个方法，或者把代码本身当作数据处理：[函数式开发者](http://www.javacodegeeks.com/2014/03/functional-programming-with-java-8-lambda-expressions-monads.html" \t "_blank)非常熟悉这些概念。很多JVM平台上的语言（Groovy、[Scala](http://www.javacodegeeks.com/tutorials/scala-tutorials/" \t "_blank)等）从诞生之日就支持Lambda表达式，但是Java开发者没有选择，只能使用匿名内部类代替Lambda表达式。

2）接口的默认方法和静态方法

Java 8使用两个新概念扩展了接口的含义：默认方法和静态方法。[默认方法](http://blog.csdn.net/yczz/article/details/50896975" \t "_blank)使得接口有点类似traits，不过要实现的目标不一样。默认方法使得开发者可以在 不破坏二进制兼容性的前提下，往现存接口中添加新的方法，即不强制那些实现了该接口的类也同时实现这个新加的方法。

默认方法和抽象方法之间的区别在于抽象方法需要实现，而默认方法不需要。接口提供的默认方法会被接口的实现类继承或者覆写

3）方法引用

方法引用使得开发者可以直接引用现存的方法、Java类的构造方法或者实例对象。方法引用和Lambda表达式配合使用，使得java类的构造方法看起来紧凑而简洁，没有很多复杂的模板代码。

4）重复注解

自从Java 5中引入[注解](http://www.javacodegeeks.com/2012/08/java-annotations-explored-explained.html" \t "_blank)以来，这个特性开始变得非常流行，并在各个框架和项目中被广泛使用。不过，注解有一个很大的限制是：在同一个地方不能多次使用同一个注解。Java 8打破了这个限制，引入了重复注解的概念，允许在同一个地方多次使用同一个注解。

在Java 8中使用**@Repeatable**注解定义重复注解

1. **你知道哪些数据结构？**

线性表

队列

栈和堆

二叉树

二叉树可以为空。二叉树结点的子树要区分左子树和右子树，即使只有一棵子 树也要进行区分，说明它是左子树，还是右子树。这是二叉树与树的最主 要的差别。

1. D
2. d
3. **前端**
4. **HTML**
5. **超链接标签**

跳转：<a href = "资源路径" target="\_blank">

href : 用于指定链接的资源

target: 设置打开新资源的目标。

\_blank 在独立的窗口上打开新资源

\_self 在当前窗口打开新资源

1. **框架标签**

frameSet 一个frameSet可以把一个页面切割成多份。只能按照行或者列切割。

frame 不能被切割的。 frame是位于frameSet中。

注意： frameSet标签不能用于body标签体内容。

<frameset rows="20%,70%,\*" >

<frame src="top.html" />

<frameset cols="20%,\*">

<frame src="left.html" />

<frame name="center" src="center.html"/>

</frameset>

<frame src="foot.html"/>

</frameset>

<noframes>

</noframes>

1. **表单**

表单标签的作用是用于提交数据给服务器。

表单标签的根标签是<form>标签

常用的属性：

action: 该属性是用于指定提交数据的地址。

method： 指定表单的提交方式。

<form action="dopost.html" method="get">

<!-- 文本输入框 单行-->

用户名：<input name="userName" type="text"/><br/>

<!-- 密码框 -->

密码：<input name="password" type="password"/><br/>

<!-- 单选框 -->

性别： 男<input checked="true" value="man" name="sex" type="radio"/>

女<input name="sex" value="woman" type="radio"/><br/>

<!-- 下拉框 -->

来自的城市：<select name="city">

<option value="BJ">北京</option>

<option value="SH">上海</option>

<option value="GZ">广州</option>

<option value="SZ">深圳</option>

</select><br/>

<!-- 复选框 同一组的复选框name的属性值要一致 -->

兴趣爱好：

java <input value="java" name="hobit" checked="checked" type="checkbox" />

javascript <input type="checkbox" value="javascript" name="hobit" />

android <input value="android" name="hobit" type="checkbox" /><br/>

<!-- 文件上传框-->

大头照：<input name="image" type="file" /><br/>

个人简介:

<!-- 文本域 -->

<textarea name="intro" rows="10" cols="30"></textarea><br/>

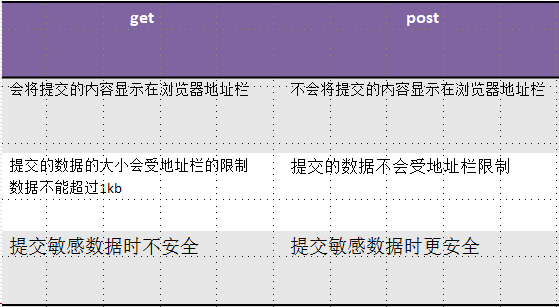
<!-- 提交按钮 -->

<input type="submit" value="注册"/>

<!-- 重置按钮 -->

<input type="reset" value="重置"/></form>

1. **Get与post提交的区别**



get : 默认使用的提交方式。 提交的数据会显示在地址栏上。

post :提交的数据不会显示在地址栏上。

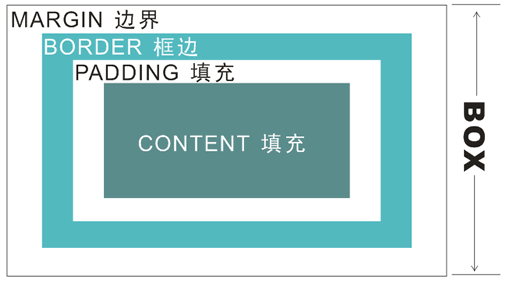
注意： 表单项的数据如果需要提交到服务器上面，那么表单项必须要有name 属性值。id属性值在做javascript验证时使用的，最好让id和name值相同。

1. **盒子模型**

  在网页布局中，为了能够使复杂的各个部分合理地进行组织，通过研究，总结 出了一套完整的、有效的原则和规范，这就是“盒子模型”的由来。

在CSS中，一个独立的盒子模型由内容(content)、边框(border)、填充(padding) 和边界(margin)4个部分组成：

内容(content)：对应盒内物品  
  边框(border)：对应包装盒的纸壳，具有厚度  
  填充(padding)：位于边框内部，是内容与边框的距离，对应包装盒的填充部分  
  边界(margin)：位于边框外部，是边框外面周围的间隙，对应纸壳外围间隙



1. **JavaScript**
2. **javascript与Java的不同**

Java代码必须编译才能执行，而JavaScript不需编译，只需由浏览器解释执行；

Java多运行于服务器，而JavaScript多用于客户端；

JavaScript使用的是弱类型的语言，而Java使用的是强类型的语言。

JSP(java server page) 运行于服务器

1. **数据类型**

Javascript 有三种主要数据类型、两种复合数据类型和两种特殊数据类型。

主要（基本）数据类型是：

* 字符串
* 数值
* 布尔

复合（引用）数据类型是：

* 对象
* 数组

特殊数据类型是：

* Null
* Undefined

NaN:not a number

1. **prototype**

**用来增加自定义方法到String对象**

<SCRIPT LANGUAGE="JavaScript">

String.prototype.startWith = function(str)

{

//alert(str.length);

//0,截取原字符串str.length字符

//alert(this.substr(0,str.length));

return this.substr(0, str.length) == str;

}

var s="ajaxbooks"

var flag = s.startWith("aj");

//alert(flag )

</SCRIPT>

1. **String中的方法**

**确定和定位字符串中的子串的方法:**

indexOf(搜寻值,[起始位置]):由左至右搜寻该字串第一次出现的位置

lastIndexOf(搜寻值,[起始位置]):由右至左搜寻该字串第一次出现的位置

charAt(index):输出索引值所指的字符

substring(索引值A[,索引值B]):输出由索引值A至索引值B-1所构成的字串

substr(索引值A, 字串长度):输出由索引值A开始的子串

**处理文本字符串的方法:**

toLowerCase():转换成小写

toUpperCase():转换成大写

split(“字符”) :按指定字符分割(返回值为数组)

concat(str2 [, str3]):将几段字串结合成一个字符串

replace(x,y) :用字串y 取代字串x

test(regExp):检验字符串与正则表达式是否匹配(返回值为true或false)

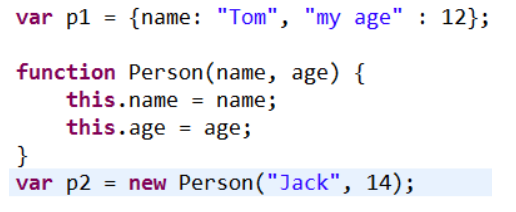
1. **一般事件**

* 获得焦点事件onFocus
* 失去焦点事件onBlur
* 内容改变事件 onChange
* 载入页面onLoad
* 卸载页面onUnload
* 单击事件onClick
* 鼠标移动事件onmousemove
* 鼠标按下事件onmousedown
* 鼠标抬起事件onmouseup
* 鼠标移上事件onmouseover
* 鼠标移开事件onmouseout
* 键按下事件onkeydown
* 键抬起事件onkeyup
* 键按下并抬起事件onkeypress
* 表单提交事件onsubmit
* 表单重置事件onreset

1. **DOM常用方法**

* createElement(标签名) 建立一个指定名称的元素。
* createTextNode (字符串) 建立一个新的text 节点
* appendChild(oNode) 追加一个子节点（作为最后的子结点）
* insertBefore 在某子结点前插入一个新的子节点
* cloneNode() 为给定节点创建一个副本，参数为true或false，true 表示同时复制该节点的子节点，false则不复制任何子节点。
* removeChild() 将从一个给定元素里删除一个子节点
* replaceChild() 把给定父元素里的一个子节点替换为另外一个节点
* hasChildNodes() 用来检查给定元素是否有子节点

1. **在 js中如何创建一个对象?**



1. **Ajax技术**

AJAX全称为“Asynchronous JavaScript and XML”（异步JavaScript和XML），是一种创建交互式网页应用的网页开发技术。Ajax的工作原理相当于在用户和服务器之间加了—个中间层(AJAX引擎),使用户操作与服务器响应异步化。并不是所有的用户请求都提交给服务器,像—些数据验证和数据处理等都交给Ajax引擎自己来做, 只有确定需要从服务器读取新数据时再由Ajax引擎代为向服务器提交请求。  
 **实现一个AJAX异步调用和局部刷新,通常需要以下几个步骤:**

创建XMLHttpRequest对象,也就是创建一个异步调用对象.

创建一个新的HTTP请求,使用open(get/post,url,true/false)方法，并指定该 HTTP请求的方法、URL及验证信息.

 设置响应HTTP请求状态变化的函数.

 使用send()方法发送HTTP请求.

 获取异步调用返回的数据.

使用JavaScript和DOM实现局部刷新.

**Ajax的优点**：

1.最大的一点是页面无刷新，给用户的体验非常好。

2.使用异步方式与服务器通信，不需要打断用户操作，具有更加迅速的响 应能力。

3.ajax 的原则是“按需取数据”，最大程度的减少冗余请求，减少服务器 的负荷。

**Ajax的缺点：**

1.破坏浏览器后退按钮的正常行为。在动态更新页面后，用户无法回到前 一个页面的状态.

2.使用 JavaScript 作 Ajax 的引擎，JavaScript 的兼容性和 Debug困难

**Ajax的应用场景：**

1.对数据进行联动过滤的场景(三级联动)

2.注册验证用户名是否存在

1. **Jquery**
2. **jquery有哪些好处？**

jQuery 是轻量级的 javascript 框架

强大的选择器

出色的 DOM 操作的封装

可靠的事件处理机制

完善的 ajax 封装

出色的浏览器的兼容性

支持链式操作，隐式迭代

支持丰富的插件

jquery 的文档也非常的丰富

1. **Query的常用选择器？**

ID选择器 通过ID获取一个元素

Class选择器 通过类(css)获取元素

标签选择器 通过标签获取元素

通用选择器(\*) 获取所有元素

层次选择器

1. **jquery中$.get() 提交和$.post() 提交的区别**

1. $.get() 方法使用 GET 方式提交请求,而$.post()使用 POST 方式。

2. GET 方式传输的数据大小不能超过 2KB 而 POST 要大的多

3. GET 方式请求的数据会被浏览器缓存起来，因此有安全问题。

1. **jquery的$.ajax()**

$.ajax({

url:"xxx.action",

type:"get",

success:function(返回数据){

}

dataType:"返回数据类型"

});

1. **$(document).ready() 方法和 window.onload 区别**

答: 两个方法有相似的功能，但是在实行时机方面是有区别的。

1 window.onload 方法是在网页中所有的元素(包括元素的所有关联文件) 完全加载到浏览器后才执行的。

2 $(document).ready() 方法可以在 DOM 载入就绪时就对其进行操纵，并 调用执行绑定的函数。

1. **bootstrap是什么？**

BootStrap是一个移动设备优先的UI框架。可以不用写任何css,js代码就能实现比较漂亮的有交互性的页面。

平时用得很多的：模态框、表单，表单项、布局、删格系统

1. **Jquery的validate校验？**

jquery.validate.js 是一个前端 form 表单校验插件

jquery.validate.js 不仅自身内置有封装一些常用的前端校验，还提供便捷的 自定义校验方法进行拓展。

前端内置自带校验：

required:true 必填字段

remote:"check.php" 使用ajax方法调用check.php验证输入值

email:true 输入内容必须为正确格式的电子邮件

url:true 输入内容必须为正确格式的网址

date:true 输入内容必须为正确格式的日期

只验证格式，不验证有效性

number:true 输入内容必须为合法的数字(负数，小数)

digits:true 输入内容必须为整数

creditcard: 输入内容必须为合法的信用卡号

equalTo:"#field" 输入值必须和#field相同

accept: "gif|png|jpg" 输入拥有合法后缀名的字符串（上传文件的后缀），多个后缀之间用’|’隔开

maxlength:5 输入长度最多是5的字符串(汉字算一个字符)

minlength:10 输入长度最小是10的字符串(汉字算一个字符)

rangelength:[5,10] 输入长度必须介于 5 和 10 之间的字符串")(汉字算一个字符)

range:[5,10] 输入值必须介于 5 和 10 之间

max:5 输入值不能大于5

min:10 输入值不能小于10

1. D
2. D
3. d
4. **XML**
5. **xml 有哪些解析技术? 区别是什么?**

答:有 DOM,DOM4j,SAX等

DOM: 一次性将整个文档加载到内存中, 生成一个对象树, 在处理大型文 件时其性能下降的非常厉害。

DOM4J: 对 DOM 的进一步封装, API 使用更简洁

SAX:基于事件驱动的方法回调机制。每读取一小部分数据时就会回调事件 处理器对象的方法, 但解析一旦开始就不能停止.

1. **你在项目中用到了 xml 技术的哪些方面? 如何实现的?**

答:用到了数据存贮，信息配置两方面。

在做数据交换平台时，将数据源的数据组装成 XML 文件，然后将 XML 文件压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再同 XML 文件中还原相关信息进行处理。

在做软件配置时，利用 XML 可以很方便的进行，软件的各种配置参数都存贮在 XML 文件中。

1. **什么是格式良好的 XML**

答：一个格式良好的 XML 意味着该 XML 文档语法上是正确的，比如它有一个根元素，所有的开放标签合适地闭合，属性值必须加引号等等。

如果一个 XML 不是格式良好的，那么它可能不能被各种 XML 解析器正确地处理和解析。

1. **说说你对 JSON的理解**

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它基于标准JavaScript 的一个子集,是一个 Js 对象或数组结构的字符串。

JSON 有三类数据

1. 单个数据

有 number, string, boolean 和 null 四种类型数据

2. 多个有序的数据: 数组

用[]包含起来, 其元素可以是三类数据中的任意一种, 元素之间用,号隔开

3. 多个无序的数据: 对象

用{}包含起来, 其元素必须由 key-value 组成, key 是一个字符串, value 可以是任意类型数据, key 与 value 之间用:号隔开, 两个 key-value 之间用,号隔开.

1. **XML和JSON的比较**

**1.数据交换格式比较之关于XML和JSON：**

XML：extensible markup language,一种类似于HTML的语言，他没有预先定义的标 签，使用DTD(document type definition)文档类型定义来组织数据；格式统一，跨平 台和语言，早已成为业界公认的标准。具体的可以问Google或百度。相比之JSON 这种轻量级的数据交换格式，XML可以称为重量级的了。

JSON : JavaScript Object Notation 是一种轻量级的数据交换格式。易于人阅读和编写。 同时也易于机器解析和生成。它基于JavaScript Programming Language , Standard ECMA-262 3rd Edition – December 1999 的一个子集。 JSON采用完全独立于语言的 文本格式，但是也使用了类似于C语言家族的习惯（包括C, C++, C#, Java, JavaScript, Perl, Python等）。这些特性使JSON成为理想的数据交换语言。

**2.数据交换格式比较之关于轻量级和重量级：**

轻量级和重量级是相对来说的，那么XML相对于JSON的重量级体现在哪呢？我想 应该体现在解析上，XML目前设计了两种解析方式：DOM和SAX；

DOM是把一个数据交换格式XML看成一个DOM对象，需要把XML文件整个读入 内存，这一点上JSON和XML的原理是一样的，但是XML要考虑 父节点和子节点， 这一点上JSON的解析难度要小很多，因为JSON构建于两种结构：key/value，键 值对的集合；值的有序集合，可理解为数组；

SAX不需要整个读入文档就可以对解析出的内容进行处理，是一种逐步解析的方法。 程序也可以随时终止解析。这样，一个大的文档就可以逐步的、一点一点的展现出 来，所以SAX适合于大规模的解析。这一点，JSON目前是做不到得。

所以，JSON和XML的轻/重量级的区别在于：JSON只提供整体解析方案，而这种 方法只在解析较少的数据时才能起到良好的效果；而XML提供了对大规模数据的 逐步解析方案，这种方案很适合于对大量数据的处理。

**3.数据交换格式比较之关于数据格式编码及解析的难度：**

在编码上，虽然XML和JSON都有各自的编码工具，但是JSON的编码要比XML简 单，即使不借助工具，也可以写出JSON代码，但要写出好的 XML代码就有点困 难；与XML一样，JSON也是基于文本的，且它们都使用Unicode编码，且其与数 据交换格式XML一样具有可读性。

主观上来看，JSON更为清晰且冗余更少些。JSON网站提供了对JSON语法的严格 描述，只是描述较简短。从总体来看，XML比较适合于标记文档，而JSON却更适 于进行数据交换处理。

在解析上，在普通的web应用领域，开发者经常为XML的解析伤脑筋，无论是服 务器端生成或处理XML，还是客户端用 JavaScript 解析XML，都常常导致复杂的 代码，极低的开发效率。

1. D
2. s
3. **JSP**
4. **服务器**
5. **Tomcat，Apache，JBoss的区别？**

Apache解析静态的html文件；Tomcat可解析jsp动态页面、也可充当servlet容器。

Apache是Http服务器，Tomcat是web服务器,只支持jsp+servlet

JBoss是应用服务器，支持EJB. Weblogic，WebSphere

1. **虚拟目录**

设置虚拟路径的优势有两点。

第一、我们开发一个项目，在未定版之前需要进行无数次的修改，如果部署测试项目的时候直接将项目打成war包放到webapps目录下，那么每一次版本变更都需要将新的版本放到webapps目录下覆盖原先的版本，这样对于我们测试项目来说是不是非常麻烦呢？如果建立一个虚拟路径，直接映射到项目原件，这样一来在测试阶段是非常的方便

第二、Tomcat是一个服务器，既然是一个服务器是不是就有容量这么一说呢？虽然这个容量是你说了算，但是再怎么大是不是也有盘符容量限制？如果将所有的项目都部署到webapps目录下，当项目比较大而且项目较多的时候是不是需要考虑一下盘符的感受呢？所以使用虚拟路径也可以为搭载Tomcat服务器的盘符分压

添加虚拟目录

tomcat6\conf\server.xml

在</Host>之上进行配置

<Context path="/test" docBase="d:/testweb"/>

path=“/test”表示此虚拟目录的名称:http://localhost:8080/test

docBase=“d:/testweb”表示虚拟目录在硬盘上的绝对路径

1. **HTTP 响应的结构是怎么样的？**

HTTP 响应由三个部分组成：

状态码(Status Code)：描述了响应的状态。可以用来检查是否成功的完成了请求。请求失败的情况下，状态码可用来找出失败的原因。如果 Servlet 没有返回状态码，默认会返回成功的状态码 HttpServletResponse.SC\_OK

HTTP 头部(HTTP Header)：它们包含了更多关于响应的信息。比如：头部可以指定认为响应过期的过期日期，或者是指定用来给用户安全的传输实体内容的编码格式。如何在 Serlet中检索 HTTP 的头部看这里。

主体(Body)：它包含了响应的内容。它可以包含 HTML 代码，图片，等等。主体是由传输在HTTP 消息中紧跟在头部后面的数据字节组成的

1. **JSP**
2. **jsp 有哪些内置对象? 作用分别是什么?**

答:JSP 共有以下 9 个内置的对象：

request: 用户端请求，此请求会包含来自 GET/POST 请求的参数

response: 网页传回用户端的回应

pageContext: 网页的属性是在这里管理

session: 与请求有关的会话期

application: 与当前应用对应的 ServletContext 对象, 应用中只有一个

out: 用来传送回应的输出 {}<%=%>

config: 与 jsp 配置对象的对象, 一般无用

page: jsp 对应的 Servlet 对象

exception: 针对错误网页，未捕捉的异常对象

1. **在JSP中提供了四种属性保存范围**

在一个页面范围内:pagecontext

在一次服务器请求范围内:request

在一次会话范围内:session

 在一个应用服务器范围内:application

1. **JSP请求是如何被处理的**

浏览器首先要请求一个以.jsp 扩展名结尾的页面，发起 JSP 请求，然后，Web 服务器读取这个请求，使用 JSP 编译器把 JSP 页面转化成一个 Servlet 类。需要注意的是，只有当第一次请求页面或者是 JSP 文件发生改变的时候 JSP 文件才会被编译，然后服务器调用 servlet 类，处理浏览器的请求。一旦请求执行结束，servlet 会把响应发送给客户端。这里看下如何在 JSP中获取请求参数。

1. **jsp有哪些动作? 作用分别是什么?**

jsp:include：在页面被请求的时候引入一个文件。

jsp:forward：把请求转到一个新的页面。

jsp:useBean：寻找或者实例化一个 JavaBean。

jsp:setProperty：设置 JavaBean 的属性。

jsp:getProperty：输出某个 JavaBean 的属性。

1. **JSP中动态 INCLUDE与静态 INCLUDE的区别？**

1. 动态包含: 用<jsp:include>, 包含的动作是在 jsp 对应的 Serlet 处理请求时去执行的,每次请求都会执行.

2. 静态包含: 用 include 指令, 包含的动作是在 jsp 被编译成 java 文件时执行的,只有第一次请求时执行.

1. **Cookie**

Cookie是服务器端保存在客户端的一组资源。

应用:在登陆时会问是否要记住密码，或是在一定时间不用再登陆，这样的功能就是通过Cookie实现的。

 public void addCookie(Cookie cookie)

 Cookie是通过服务器设置到客户端上去的，用response对象。

 如果要在服务器端取得Cookie，用request对象。

 如果要Cookie长留在本地计算机上，可以设置Cookie的保存时间。(此时就可以在新的页面中取出Cookie的值)

Cookie是保存在客户端上的信息，安全性较差，不要把过多的隐秘信息保留在Cookie中，这样不安全。

 Cookie可以用setMaxAge设置最大保留时间

 服务器端在客户端第一次访问之后，会设置一个默认的Cookie在客户端上:JsessionID

1. **Session**

session对象:主要用于保存用户的各种信息，直到它的生命周期(一般为900s)超时或人为释放掉为止。用户只要一连接到服务器，就立刻分配一个session给用户。

可以通过session对象来判断此用户是否是合法用户

Session的主要方法：

 服务器上通过sessionID来区分不同的用户, 任何连接到服务器上的用户，服务器都会为之分配唯一的一个不会重复的sessionID。session.getId() ，长度为:32

 判断是否是新的session。Session.isNew()

 设置属性:

public void setAttribute(String name,Object value)

 取得属性:

public Object getAttribute(String name)

 删除属性

public void removeAttribute(String name)

让用户的session失效

Session.invalidate()

 取得用户最后一次操作的时间

public long getLastAccessedTime()

获得创建时间

public long getCreationTime();

1. **说说你对 Cookie与 Session技术的理解?**

cookie 是一种浏览器端的缓存技术, 而 Session 是一种服务器端的缓存技术(依赖 cookie)

session和cookie用于跨网页共享数据

session:记录独有的个人信息，在不同的页面中传递。

主要方法:setAttribute(),getAttribute()

cookie:保存于客户端，供浏览器与Web服务器互通数据用的纯文字文件。当IE执行时，会在计算机中产生一个cookie。

session将信息保存在服务器上，而cookie保存在客户端上。

session比cookie更安全，session比cookie更占用资源。

开发原则:session少用。尽量少向session中保存信息。Session使用了cookie的机制，如果cookie禁用，则session也无法使用。此时可以使用URL地址重定向解决。

当客户端第一次访问web应用或者第一次使用request.getSession()获取HttpSession时，Servlet容器会创建Session，生成一个long类型的唯一ID（你可以使用session.getId()获取它）并把它保存在服务器的内存里。Servlet容器同样会在HTTP响应里设置一个Cookie，cookie的名是JSESSIONID并且cookie的值是session的唯一ID。

根据HTTP cookie规范（正规的web浏览器和web服务器必须遵守的约定），在cookie的有效期间，客户端（web浏览器）之后的请求都要把这个cookie返回给服务器。Servlet容器会利用带有名为JSESSIONID的cookie检测每一个到来的HTTP请求头，并使用cookie的值从服务器内容里获取相关的HttpSession。

HttpSession会一直存活着，除非超过一段时间没使用。你可以在web.xml里设定这个时间段，默认时间段是30分钟。因此，如果客户端已经超过30分钟没有访问web应用的话，Servlet容器就会销毁Session。之后的每一个请求，即使带有特定的cookie，都再也不会访问到同一个Session了。servletcontainer会创建一个新的Session。

另外，在客户端的session cookie拥有一个默认的存活时间，这个时间与浏览器的运行时间相同。因此，当用户关闭浏览器后（所有的标签或者窗口），客户端的Session就会被销毁。重新打开浏览器后，与之前的Session关联的cookie就再也不会被发送出去了。再次使用request.getSession()会返回一个全新的HttpSession并且使用一个全新的session ID来设置cookie。

购物车最好使用cookie，但是cookie是可以在客户端禁用的，这时候我们要使用cookie+数据库的方式实现，当从cookie中不能取出数据时，就从数据库获取。

1. **自动登陆功能的编码实现?**

1. 登陆功能是用 Session 实现的,就是向 Session 对象中保存当前用户的对象2. 自动的功能由Cookie 实现, 在登陆时将用户的信息保存为持久化 Cookie

3. 下次访问时, 读取请求中如果有用户信息的 Cookie 就可以自动登陆

1. **如何防止表单重复提交?**

答: 使用 Session 技术:

1. 在 regist.jsp 页面中生成一个唯一随机值, 将其保存到 Session 中, 同时将其保存为表单的隐藏域的值

2. 在处理注册的请求时,获取 Session 中值,获取请求参数的值,比较两者是否相同, 如果相同说明不是重复提交,请求通过同时删除 session 中保存的值,如果不相同则是重复提交, 不能通过。

1. **结合项目谈谈你对MVC的理解**

MVC是Model—View—Controler的简称。即模型—视图—控制器。MVC是一种设计模式，它强制性的把应用程序的输入、处理和输出分开。

 MVC中的模型、视图、控制器它们分别担负着不同的任务。

    视图: 视图是用户看到并与之交互的界面。视图向用户显示相关的数据，并接受用户的输入。视图不进行任何业务逻辑处理。

    模型: 模型表示业务数据和业务处理。相当于JavaBean。一个模型能为多个视图提供数据。这提高了应用程序的重用性

    控制器: 当用户单击Web页面中的提交按钮时,控制器接受请求并调用相应的模型去处理请求。

然后根据处理的结果调用相应的视图来显示处理的结果。

MVC的处理过程：首先控制器接受用户的请求，调用相应的模型来进行业务处理，并返回数据给控制器。控制器调用相应的视图来显示处理的结果。并通过视图呈现给用户。

如在项目中要对应MVC的话：View 对应项目中Jsp，Controler对应Action，Model 对应service+dao层的业务逻辑和持久层的操作。

1. **JDBC**
2. **Class.forName的作用?为什么要用?**

按参数中指定的字符串形式的类名去搜索并加载相应的类，如果该类字节码已经被加载过，则返回代表该字节码的Class实例对象，否则，按类加载器的委托机制去搜索和加载该类，如果所有的类加载器都无法加载到该类，则抛出ClassNotFoundException。加载完这个Class字节码后，接着就可以使用Class字节码的newInstance方法去创建该类的实例对象了。有时候，我们程序中所有使用的具体类名在设计时（即开发时）无法确定，只有程序运行时才能确定，这时候就需要使用Class.forName去动态加载该类，这个类名通常是在配置文件中配置的，例如，spring的ioc中每次依赖注入的具体类就是这样配置的，jdbc的驱动类名通常也是通过配置文件来配置的，以便在产品交付使用后不用修改源程序就可以更换驱动类名。

1. **用JDBC如何调用存储过程**

CallableStatement 用来执行存储过程。存储过程是由数据库存储和提供的。存储过程可以接受输入参数，也可以有返回结果。非常鼓励使用存储过程，因为它提供了安全性和模块化。准备一个CallableStatement 的方法是：CallableStament.prepareCall();

1. **使用JDBC操作数据库时，如何提升读取数据的性能？如何提升更新数据的性能**

要提升读取数据的性能，可以指定通过结果集（ResultSet）对象的setFetchSize()方法指定每次抓取的记录数（典型的空间换时间策略）；要提升更新数据的性能可以使用PreparedStatement语句构建批处理，将若干SQL语句置于一个批处理中执行。

1. **Statement接口与PreparedStatement接口区别**

* Statement类的对象将SQL语句发送给数据库，如果SQL语句运行后产后结果集，Statement对象会将结果集返回给一个ResultSet对象。
* PreparedStatement可以接受一个带有占位符?的查询语句，并且PreparedStatement对象会将传入的SQL语句进行编译并暂时保存在内存中。
* 动态SQL语句：就是在SQL语句中可以提供参数。
* SQL 注入是用户利用某些系统没有对输入数据进行充分的检查，从而进行恶意破坏的行为。
* PreparedStatement可防止SQL注入现象发生

1. **SexecuteQuery()和executeUpdate()区别**

executeQuery()方法是处理查询操作，当用select对数据库做查询时，用该方法。且executeQuery()方法会建立一个ResultSet的对象保存查询的结果。

Insert、Update、delete等操作，可用executeUpdate()方法执行。

1. **用JDBC来实现访问数据库记录可以采用下面的几个步骤**
2. 通过驱动器管理器获取连接接口（Connection）。
3. 获得Statement或它的子类。
4. 指定Statement中的参数。
5. 通过Statement发送SQL语句。
6. 检查并处理返回的结果。
7. 关闭Statement。
8. 关闭连接
9. **用 JDBC 查询学生成绩单, 把主要代码写出来**

Connection cn = null;

PreparedStatement pstmt =null;

Resultset rs = null;

try

{

Class.forname(driveClassName);

cn = DriverManager.getConnection(url,username,password);

pstmt = cn.prepareStatement(“select score.\* from score ,student “ +

“where score.stuId = student.id and student.name = ?”);

pstmt.setString(1,studentName);

Resultset rs = pstmt.executeQuery();

while(rs.next())

{

system.out.println(rs.getInt(“subject”) + “ ” + rs.getFloat(“score”) );

}

}catch(Exception e){e.printStackTrace();}

finally

{

if(rs != null) try{ rs.close() }catch(exception e){}

if(pstmt != null) try{pstmt.close()}catch(exception e){}

if(cn != null) try{ cn.close() }catch(exception e){}

}

1. **在进行数据库编程时，连接池有什么作用？**

答：由于创建连接和释放连接都有很大的开销（尤其是数据库服务器不在本地时，每次建立连接都需要进行TCP的三次握手，释放连接需要进行TCP四次握手，造成的开销是不可忽视的），为了提升系统访问数据库的性能，可以事先创建若干连接置于连接池中，需要时直接从连接池获取，使用结束时归还连接池而不必关闭连接，从而避免频繁创建和释放连接所造成的开销，这是典型的用空间换取时间的策略（浪费了空间存储连接，但节省了创建和释放连接的时间）。池化技术在Java开发中是很常见的，在使用线程时创建线程池的道理与此相同。基于Java的开源数据库连接池主要有：C3P0、DBCP、Druid等。

补充：在计算机系统中时间和空间是不可调和的矛盾，理解这一点对设计满足性能要求的算法是至关重要的。大型网站性能优化的一个关键就是使用缓存，而缓存跟上面讲的连接池道理非常类似，也是使用空间换时间的策略。可以将热点数据置于缓存中，当用户查询这些数据时可以直接从缓存中得到，这无论如何也快过去数据库中查询。当然，缓存的置换策略等也会对系统性能产生重要影响

1. **什么是DAO模式**

答：DAO（Data Access Object）是一个为数据库或其他持久化机制提供了抽象接口的对象，在不暴露底层持久化方案实现细节的前提下提供了各种数据访问操作。在实际的开发中，应该将所有对数据源的访问操作进行抽象化后封装在一个公共API中。用程序设计语言来说，就是建立一个接口，接口中定义了此应用程序中将会用到的所有事务方法。在这个应用程序中，当需要和数据源进行交互的时候则使用这个接口，并且编写一个单独的类来实现这个接口，在逻辑上该类对应一个特定的数据存储。DAO模式实际上包含了两个模式，一是Data Accessor（数据访问器），二是Data Object（数据对象），前者要解决如何访问数据的问题，而后者要解决的是如何用对象封装数据。

1. **事务的ACID是指什么？**

- 原子性(Atomic)：事务中各项操作，要么全做要么全不做，任何一项操作的失败都会导致整个事务的失败；

- 一致性(Consistent)：事务结束后系统状态是一致的；

- 隔离性(Isolated)：并发执行的事务彼此无法看到对方的中间状态；

- 持久性(Durable)：事务完成后所做的改动都会被持久化，即使发生灾难性的失败。通过日志和同步备份可以在故障发生后重建数据。

关于事务，在面试中被问到的概率是很高的，可以问的问题也是很多的。首先需要知道的是，只有存在并发数据访问时才需要事务。当多个事务访问同一数据时，可能会存在5类问题，包括3类数据读取问题（脏读、不可重复读和幻读）和2类数据更新问题（第1类丢失更新和第2类丢失更新）。

脏读（Dirty Read）：A事务读取B事务尚未提交的数据并在此基础上操作，而B事务执行回滚，那么A读取到的数据就是脏数据。

不可重复读（Unrepeatable Read）：事务A重新读取前面读取过的数据，发现该数据已经被另一个已提交的事务B修改过了。

幻读（Phantom Read）：事务A重新执行一个查询，返回一系列符合查询条件的行，发现其中插入了被事务B提交的行

1. **JDBC中如何进行事务处理？**

答：Connection提供了事务处理的方法，通过调用setAutoCommit(false)可以设置手动提交事务；当事务完成后用commit()显式提交事务；如果在事务处理过程中发生异常则通过rollback()进行事务回滚。除此之外还引入了Savepoint（保存点）的概念，允许通过代码设置保存点并让事务回滚到指定的保存点。

1. **JDBC能否处理Blob和Clob？**

答： Blob是指二进制大对象（Binary Large Object），而Clob是指大字符对象（Character Large Objec），因此其中Blob是为存储大的二进制数据而设计的，而Clob是为存储大的文本数据而设计的。JDBC的PreparedStatement和ResultSet都提供了相应的方法来支持Blob和Clob操作。

1. **servlet**
2. **HttpServlet中的方法**

**doGet()/doPost()方法**

通过覆盖HttpServlet类中的doGet()方法，可以处理浏览器端发送过来的GET请求

post是一种邮寄的方式，在浏览器的地址栏不显示提交的信息，但是这种方式传送的数据是没有限制的；

get是一种手把手的提交到服务器上，最大的信息量是1K，而且提交的信息显示在浏览器上。

**init()**

在第一次调用servlet时，会创建servlet实例，在创建这个实例时，会调用HttpServlet中的init()方法（这个方法只会被调用一次）

**service()**

当客户端有一个对servlet的请求发送过来，则服务器会产生一个新的线程，并让它调用servlet的service()方法，service()会根据收到的客户端请求类型，决定调用doGet()还是doPost()方法

**destroy()**

在删除servlet实例之前，服务器会先调用destroy()方法

1. **什么是ServletContext？它由谁创建？**

当Servlet容器启动时，它会部署并加载所有的web应用。当web应用被加载时，Servlet容器会一次性为每个应用创建Servlet上下文（ServletContext）并把它保存在内存里。Servlet容器会处理web应用的web.xml文件，并且一次性创建在web.xml里定义的Servlet、Filter和Listener，同样也会把它们保存在内存里。当Servlet容器关闭时，它会卸载所有的web应用和ServletContext，所有的Servlet、Filter和Listner实例都会被销毁。

从Java文档可知，ServletContext定义了一组方法，Servlet使用这些方法来与它的Servlet容器进行通信。

1. **Servlet生命周期**

 当客户端第一次向web服务器提出一个对servlet的请求时，web服务器会创建一个该servlet的实例，并调用init()方法。如果当web服务器中已存在一个servlet实例，则直接使用此实例，然后再调用service()方法，service()方法根据客户端的请求方式决定调用对应的doXXX()方法，当servlet从web服务器中删除时，web服务器会调用servlet的destroy()方法

1. **Forward与sendRedirect区别**

forward是服务器端的转向而redirect是客户端的跳转。

使用forward浏览器的地址不会发生改变。而redirect会发生改变。

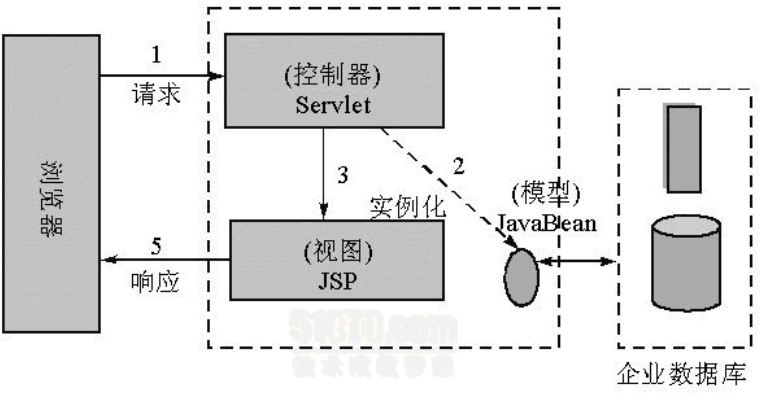
Forward是一次请求中完成。而redirect是重新发起请求。

Forward是在服务器端完成，而不用客户端重新发起请求，效率较高。

1. **MVC设计模式**

MVC全名是Model View Controller，是模型(model)－视图(view)－控制器(controller)的缩写，一种软件设计典范，用一种业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。

最简单的、最经典就是Jsp(view) +Servlet(controller) + JavaBean(model)



1、当控制器收到来自用户的请求

2、控制器调用JavaBean完成业务

3、完成业务后通过控制器跳转JSP页面的方式给用户反馈信息

**传统MVC模式问题**

1、所有的Servlet和Servlet映射都要配置在web.xml中，如果项目太大，web.xml就太庞大，并且不能实现模块化管理。

2、Servlet的主要功能就是接受参数、调用逻辑、跳转页面，比如像其他字符编码、文件上传等功能也要写在Servlet中， 不适合。

3、接受参数比较麻烦(String name = request.getParameter(“name”),User user=new User user.setName(name))，不能通过model接收，只能单个接收，接收完成后转换封装model.

4、跳转页面方式比较单一(forword,redirect),并且当我的页面名称发生改变时需要修改Servlet源代码.

**现在比较常用的MVC框架有：**

Struts2

Spring MVC

SpringBoot(推荐)

1. **Filter与listener**
2. **Filter接口中的方法**

* void **init**([FilterConfig](mk:@MSITStore:C:\\Documents%20and%20Settings\\tty\\桌面\\java-ee-5.chm::/j2ee5/api/javax/servlet/FilterConfig.html) filterConfig) throws [ServletException](mk:@MSITStore:C:\Documents%20and%20Settings\tty\桌面\java-ee-5.chm::/j2ee5/api/javax/servlet/ServletException.html)

过滤器初始化是在启动容器(Tomcat)时自动初始化

* void **doFilter**([ServletRequest](mk:@MSITStore:C:\\Documents%20and%20Settings\\tty\\桌面\\java-ee-5.chm::/j2ee5/api/javax/servlet/ServletRequest.html) request, [ServletResponse](mk:@MSITStore:C:\Documents%20and%20Settings\tty\桌面\java-ee-5.chm::/j2ee5/api/javax/servlet/ServletResponse.html) response, [FilterChain](mk:@MSITStore:C:\Documents%20and%20Settings\tty\桌面\java-ee-5.chm::/j2ee5/api/javax/servlet/FilterChain.html) chain) throws [IOException](http://java.sun.com/j2se/1.5/docs/api/java/io/IOException.html), [ServletException](mk:@MSITStore:C:\Documents%20and%20Settings\tty\桌面\java-ee-5.chm::/j2ee5/api/javax/servlet/ServletException.html)

如果过滤器要将内容传递到目的地，则需要FilterChain,将请求继续向下转发。

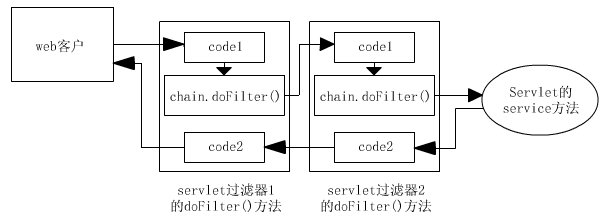
chain.doFilter(request,response) ;

过滤器会执行两次:FilterChain之前执行一次，之后再执行一次

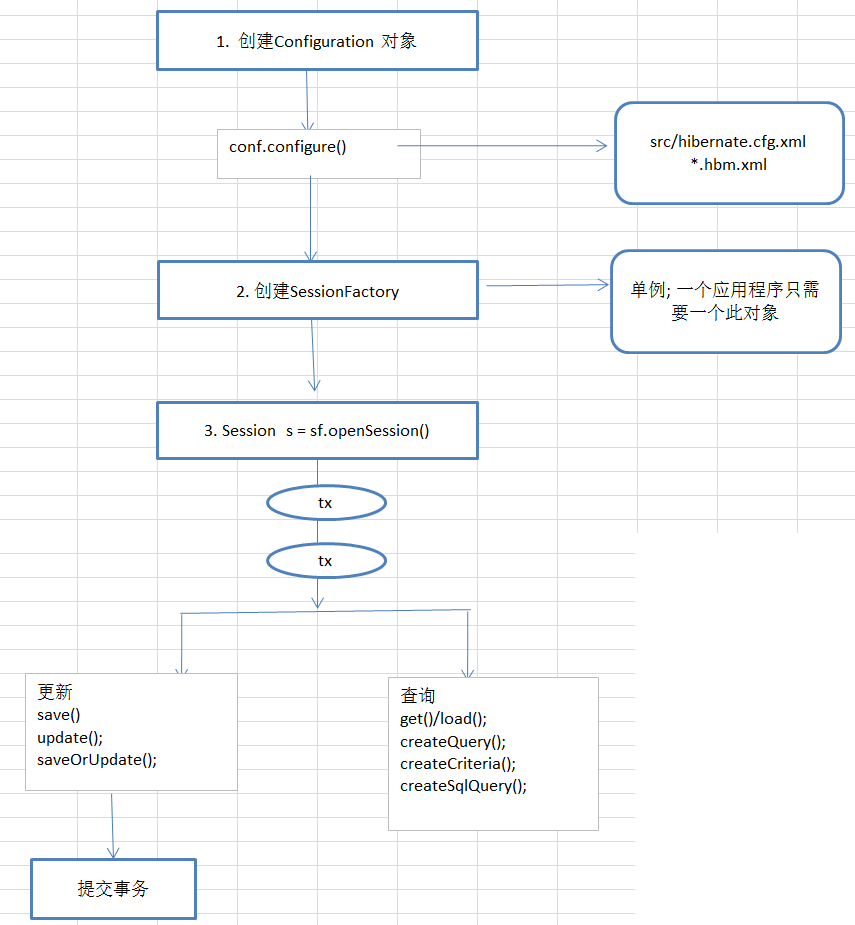
* void **destroy**()

在标准Servlet中的doGet或doPost方法中，传递的参数为HttpServletRequest、HttpServletResponse

在过滤Servlet中的doFilter中传递的是ServletRequest、ServletResponse



1. D
2. **Hibernate**
3. **Hibernate基础**
4. **Hibernate工作原理**



1. 通过Configuration对象读取并解析配置文件
2. 读取并解析映射信息，创建SessionFactory对象
3. 打开session
4. 创建事务Transaction
5. 持久化操作，对对象进行CRUD操作
6. 提交事务
7. 关闭session和SessionFactory对象
8. **Hibernate优缺点**

优点：

对 JDBC 访问数据库的代码做了封装，简化了数据访问层繁琐的重复性代码

映射的灵活性, 它支持各种关系数据库, 从一对一到多对多的各种复杂关系.

非侵入性、移植性会好

缓存机制: 提供一级缓存和二级缓存

缺点：

无法对 SQL 进行优化

框架中使用 ORM 原则, 导致配置过于复杂

执行效率和原生的 JDBC 相比偏差: 特别是在批量数据处理的时候

不支持批量修改、删除

1. **描述使用 Hibernate 进行大批量更新的经验. .**

直接通过 JDBC API 执行相关的 SQl 语句或调用相关的存储过程是最佳的

方式

1. **在 Hibernate 中Java 对象的状态有哪些**

临时状态（transient）：不处于 Session 的缓存中，OID为null

持久化状态（persistent）：加入到 Session 的缓存中。

游离状态（detached）：已经被持久化，但不再处于 Session 的缓存中。

1. **Session中一些重要方法的区别**

session.save(obj); 保存一个对象

session.update(emp); 更新一个对象

session.saveOrUpdate(emp); 保存或者更新的方法：

🡪没有设置主键，执行保存；

🡪有设置主键，执行更新操作;

🡪如果设置主键不存在报错！

主键查询：

session.get(Employee.class, 1); 主键查询

session.load(Employee.class, 1); 主键查询 (支持懒加载)

1. **Session的清理和清空有什么区别**

清理缓存调用的是session.flush() 方法 .

清空调用的是session.clear()方法.

Session 清理缓存是指按照缓存中对象的状态的变化来同步更新数据库，但

不清空缓存；清空是把 Session 的缓存置空, 但不同步更新数据库；

1. **Hibernate 的检索方式有哪些 ?**

导航对象图检索 HQL 检索 QBC 检索 本地 SQL 检索

1. **主键的生成策略**

native 自增长【会根据底层数据库自增长的方式选择identity或sequence】

如果是mysql数据库, 采用的自增长方式是identity

如果是oracle数据库， 使用sequence序列的方式实现自增长  
increment 自增长(会有并发访问的问题，一般在服务器集群环境使用会存在问题。)

assigned 指定主键生成策略为手动指定主键的值

uuid 指定uuid随机生成的唯一的值

foreign (外键的方式， one-to-one)

1. **Inverse属性**

Inverse属性，是在维护关联关系的时候起作用的。

表示控制权是否转移。(在一的一方起作用)

Inverse , 控制反转。

Inverse = false 不反转； 当前方有控制权

True 控制反转； 当前方没有控制权

1. **cascade 属性**

cascade 表示级联操作 【可以设置到一的一方或多的一方】

none 不级联操作， 默认值

save-update 级联保存或更新

delete 级联删除

save-update,delete 级联保存、更新、删除

all 同上。级联保存、更新、删除

1. **list与iterator查询的区别？**

list() 一次把所有的记录都查询出来，

**会放入缓存，但不会从缓存中获取数据**

Iterator N+1查询； N表示所有的记录总数

即会先发送一条语句查询所有记录的主键（1），

再根据每一个主键再去数据库查询（N）！

**会放入缓存，也会从缓存中取数据！**

1. **Hibernate的OpenSessionView问题**

①. 用于解决懒加载异常, 主要功能就是把 Hibernate Session和一个请求的线程绑定在一起, 直到页面完整输出, 这样就可以保证页面读取数据的时

候 Session一直是开启的状态, 如果去获取延迟加载对象也不会报错。

②. 问题: 如果在业务处理阶段大批量处理数据, 有可能导致一级缓存里

的对象占用内存过多导致内存溢出, 另外一个是连接问题: Session 和数据库Connection 是绑定在一起的, 如果业务处理缓慢也会导致数据库连接得不到及时的释放, 造成连接池连接不够. 所以在并发量较大的项目中不建议使用此种方式, 可以考虑使用迫切左外连接 (LEFT OUTER JOIN FETCH) 或手工对关联的对象进行初始化.

1. **Hibernate的缓存**

Hibernate 缓存包括两大类 ：Hibernate一级缓存和Hibernate 二级缓存：

1）Hibernate 一级缓存又称为“Session 的缓存”，它是内置的，不能被

卸载。由于 Session 对象的生命周期通常对应一个数据库事务或者一个应用事务，因此它的缓存是事务范围的缓存。在第一级缓存中，持久化类的每个实例都具有唯一的 OID。

2）Hibernate 二级缓存又称为“SessionFactory 的缓存”，由于

SessionFactory 对象的生命周期和应用程序的整个过程对应，因此 Hibernate二级缓存是进程范围或者集群范围的缓存，有可能出现并发问题，因此需要采用适当的并发访问策略，该策略为被缓存的数据提供了事务隔离级别。第二级缓存是可选的，是一个可配置的插件，在默认情况下，SessionFactory 不会启用这个插件。当Hibernate根据ID访问数据对象的时候，首先从Session一级缓存中查；查不到，如果配置了二级缓存，那么从二级缓存中查；如果都查不到，再查询数据库，把结果按照 ID 放入到缓存删除、更新、增加数据的时候，同时更新缓存。HashtableCacheProvider

1. 映射关系
2. **一对多映射**

部门方(一方)

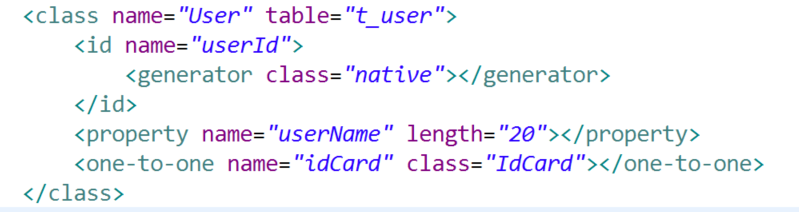


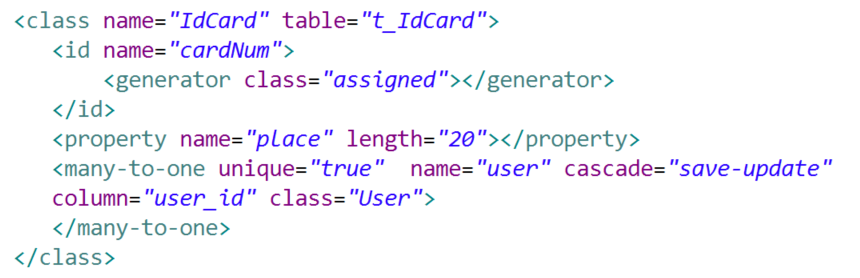
员工方（多方）



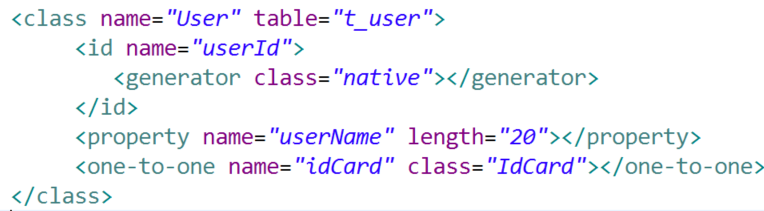
1. **一对一映射**

外键





主键



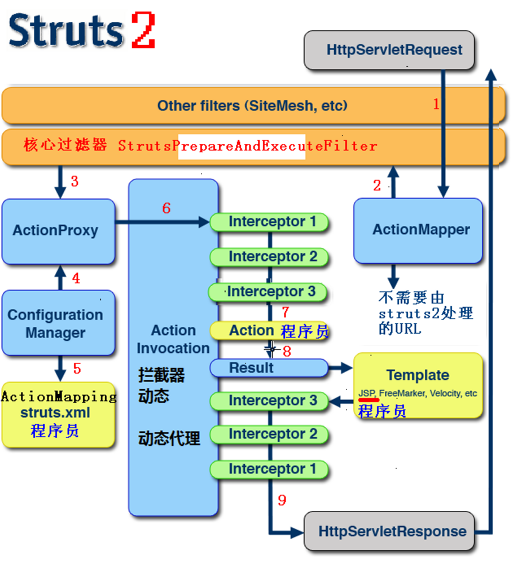


1. **多对多映射**





1. s
2. s
3. **Struts2**
4. **Struts工作原理**



①请求发送给 StrutsPrepareAndExecuteFilter

②. StrutsPrepareAndExecuteFilter判定该请求是否是一个Struts2请求

③. 若该请求是一个 Struts2 请求，则 StrutsPrepareAndExecuteFilter

把请求的处理交给 ActionProxy

④. ActionProxy 创建一个 ActionInvocation 的实例，并进行初始化

⑤. ActionInvocation 实例在调用 Action 的过程前后，涉及到相关拦截

器（Intercepter）的调用。

⑥. Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置

找到对应的返回结果。调用结果的 execute 方法，渲染结果。

⑦. 执行各个拦截器 invocation.invoke() 之后的代码

⑧. 把结果发送到客户端Struts执行流程

1. **Struts有什么优缺点**

优点:

1. 实现MVC模式，结构清晰；

2. 丰富的struts的标记库，利用好能大大提高开发效率；

3. 全局结果与声明式异常；

4. 可使用OGNL进行参数传递

5. 各个类方便使用的拦截器

缺点:

1. 转到表现层时，需要配置结果页面；页面多了比较繁杂；

2. 对Servlet的依赖性过强

3. struts标签稍微比el表达式繁重

1. **常用拦截器**

conversionError:类型转换错误拦截器

exception:异常拦截器

fileUpload:文件上传拦截器

i18n:国际化拦截器

logger:日志拦截器

params:解析请求参数拦截器

validation:校验拦截器

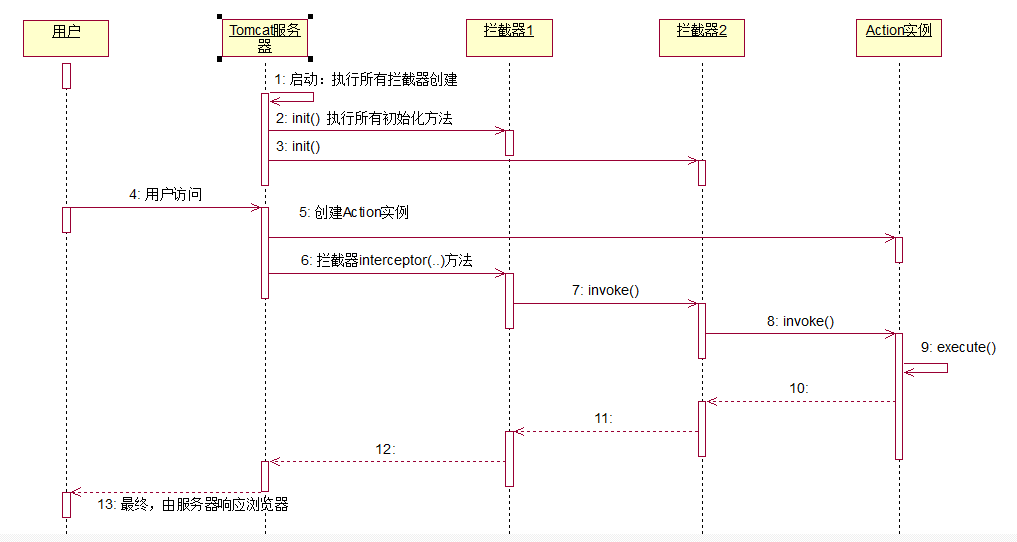
timer:这个拦截器负责输出Action的执行时间，以分析该Action的性能瓶颈。

token：避免重复提交的校验拦截器。

modelDriven：Action执行该拦截器时候可以将getModel方法得到的result值放 入值栈中

roles：进行权限配置的拦截器，如果登录用户拥有相应权限才去执行某一特 定Action。

1. **拦截器执行流程**



每个拦截器都是需要实现 Interceptor接口

> init()：在拦截器被创建后立即被调用, 它在拦截器的生命周期内只

被调用一次. 可以在该方法中对相关资源进行必要的初始化；

> intercept(ActionInvocation invocation)：每拦截一个动作请求，

该方法就会被调用一次；

> destroy：该方法将在拦截器被销毁之前被调用, 它在拦截器的生命周

期内也只被调用一次；

1. **数据封装方式**

jsp表单数据填充到action中的属性

jsp表单数据填充到action的对象中的属性

ModelDriven模型驱动(推荐)

1. **OGNL表达式**

OGNL是Object Graphic Navigation Language（对象图导航语言）的缩写，它是一 个开源项目。 Struts2框架使用OGNL作为默认的表达式语言。

它是一个功能强大的表达式语言，用来获取和设置Java对象的属性，它旨在提 供一个更高的更抽象的层次来对Java对象图进行导航。

Ognl的类中包含的很方便的方法实现OGNL表达式的赋值。实现这个功能需要 两步，解析一个表达式使之称为一种内部的形式然后再用这种内部的形式对属 性赋值或取值；

Struts框架默认就支持Ognl表达式语言。(struts必须引用包ognl.jar)

作用：页面取值用。

El表达式语言，用于页面取值，jsp页面取值的标准。(默认直接可以使用)

Ognl表达式语言， struts标签默认支持的表达式语言。

缺点：必须配置struts标签用，不能离开struts标签直接用。

1. a
2. s
3. **Spring**
4. **Spring特点**
5. **为何使用Spring**

Spring是一个轻量级J2EE框架。它的主要功能有控制反转（IoC）、面向切面编程 AOP）、面向接口开发、事务管理、还可以包容其它框架,使系统中用到的其它框架耦合程度大大降低，拓展性强、简单易用好管理。

1. **IOC**
2. **Spring中，如何给对象的属性赋值?**

1. 通过构造函数

2. 通过set方法给属性注入值

3. 自动装配(了解)

**byName** 通过参数名自动装配，如果一个bean的name 和另外一个bean的 property 相同就自动装配。

**byType** 通过参数的数据类型自动自动装配，如果一个bean的数据类型和另 外一个bean的property属性的数据类型兼容，就自动装配必须确保 该类型在IOC容器中只有一个对象；否则报错。

4.注解

使用注解步骤：

1）先引入context名称空间

xmlns:context="http://www.springframework.org/schema/context"

2）开启注解扫描

<context:component-scan base-package="cn.itcast.e\_anno2"></context:component-scan>

3）使用注解

通过注解的方式，把对象加入ioc容器。

创建对象以及处理对象依赖关系，相关的注解：

@Component 指定把一个对象加入IOC容器

@Repository 作用同@Component； 在持久层使用

@Service 作用同@Component； 在业务逻辑层使用

@Controller 作用同@Component； 在控制层使用

@Resource 属性注入

总结：

1） 使用注解，可以简化配置，且可以把对象加入IOC容器,及处理依赖关系(DI)  
 2） 注解可以和XML配置一起使用。

3） @Resource的作用相当于@Autowired，只不过@Autowired按byType自动 注入，而@Resource默认按 byName自动注入罢了

1. **bean对象创建的细节**

1) 对象创建： 单例/多例

scope="singleton", 默认值， 即 默认是单例 【service/dao/工具类】

scope="prototype", 多例； 【Action对象】

Spring的单例bean是线程安全的

2) 什么时候创建?

scope="prototype" 在用到对象的时候，才创建对象。

scope="singleton" 在启动(容器初始化之前)， 就已经创建了bean，且整个应用只有一个。

3)是否延迟创建

lazy-init="false" 默认为false, 不延迟创建，即在启动时候就创建对象

lazy-init="true" 延迟初始化， 在用到对象的时候才创建对象（只对单例有效）

4) 创建对象之后，初始化/销毁

init-method="init\_user" 【对应对象的init\_user方法，在对象创建爱之后执行】

destroy-method="destroy\_user" 【在调用容器对象的destriy方法时候执行，(容器用实现类)】

1. **Bean的作用域？**

1.singleton：这种bean范围是默认的，这种范围确保不管接受到多少个请求，每个容器中只有一个bean的实例，单例的模式由bean factory自身来维护。

2. prototype：原形范围与单例范围相反，为每一个bean请求提供一个实例。3.request：在请求bean范围内会每一个来自客户端的网络请求创建一个实例，在请求完成以后，bean会失效并被垃圾回收器回收。

4.Session：与请求范围类似，确保每个session中有一个bean的实例，在session过期后，bean会随之失效。

1. **IOC 容器对 Bean 的生命周期**
2. 通过构造器或工厂方法创建 Bean 实例
3. 为 Bean 的属性设置值和对其他 Bean 的引用
4. 将 Bean实例传递 给 Bean 后 置 处 理 器 的

postProcessBeforeInitialization 方法

1. 调用 Bean 的初始化方法(init-method)将 Bean 实 例 传 递 给 Bean 后 置 处 理 器 的postProcessAfterInitialization 方法,Bean 可以使用了
2. 当容器关闭时, 调用 Bean 的销毁方法(destroy-method)
3. **讲下BeanFactory和ApplicationContext的区别？**

BeanFactory是Spring容器顶级核心接口,比较早,但功能比较少,getBean就是BeanFactory定义的。

ApplicationContext是Spring里面的另外一个容器顶级接口,它继承于BeanFactory,但是提供的功能譬如校验,国际化,监听,对Bean的管理功能比较多,一般使用ApplicationContext。

1. **AOP**
2. **代理模式**

代理（Proxy）是一种设计模式， 提供了对目标对象另外的访问方式；即通过 代理访问目标对象。 这样好处： 可以在目标对象实现的基础上，增强额外的 功能操作。(扩展目标对象的功能)。

静态代理：要实现与目标对象一样的接口

总结静态代理：

1）可以做到在不修改目标对象的功能前提下，对目标对象功能扩展。

2）缺点：

--》 因为代理对象，需要与目标对象实现一样的接口。所以会有很多代理类，类 太多。

--》 一旦接口增加方法，目标对象与代理对象都要维护。

解决：可以使用动态代理。

动态代理

1）代理对象，不需要实现接口；

2）代理对象的生成，是利用JDKAPI， 动态的在内存中构建代理对象

动态代理总结：代理对象不需要实现接口，但是目标对象一定要实现接口；否则不 能用动态代理！

Cglib代理

也叫做子类代理。在内存中构建一个子类对象从而实现对目标对象功能的扩展。

JDK的动态代理有一个限制，就是使用动态代理的对象必须实现一个或多个接口。 如果想代理没有实现接口的类，就可以使用CGLIB实现。

CGLIB是一个强大的高性能的代码生成包，它可以在运行期扩展Java类与实现Java 接口。它广泛的被许多AOP的框架使用，例如Spring AOP和dynaop，为他们提供 方法的interception（拦截）。

CGLIB包的底层是通过使用一个小而快的字节码处理框架ASM，来转换字节码并生 成新的类。不鼓励直接使用ASM，因为它要求你必须对JVM内部结构包括class文 件的格式和指令集都很熟悉。

1. **AOP**

**Aop，** aspect object programming 面向切面编程

功能： 让关注点代码与业务代码分离！

**关注点,** 重复代码就叫做关注点；

**切面，**关注点形成的类，就叫切面(类)！

面向切面编程，就是指对很多功能都有的重复的代码抽取，再在运行的时候往 网业务方法上动态植入“切面类代码”。

**切入点，**执行目标对象方法，动态植入切面代码。

可以通过切入点表达式，指定拦截哪些类的哪些方法； 给指定的类在运行的 时候植入切面类代码。

分析总结：

关注点代码，就是指重复执行的代码。

**业务代码与关注点代码分离，好处？**

--🡪 关注点代码写一次即可；

-🡪开发者只需要关注核心业务；

-🡪运行时期，执行核心业务代码时候动态植入关注点代码； 【代理】

AOP一般的使用场景？

缓存、权限、错误处理、持久化、资源池、同步、事物、日志跟踪等

1. **事务**
2. **描述在系统中如何使用了Spring的事务控制**

Spring事务包括编程式事务和声明式事务。在系统中使用了声明式的事务管理是用Spring的AOP来实现的；配置了只读事务和回滚事务（传播行为为REQUIRED）当出现错误后进行回滚操作。在项目中通过aop切入事务到serivce层，这样做能使一次业务逻辑操作如果包括几个数据库操作都控制在一个事务中。

1. **如何管理事务，事务是只读的还是读写的，对于查询的find()是只读，对于保存的save()是读写？**

如果一次执行单条查询语句，则没有必要启用事务支持，数据库默认支持[SQL](https://www.baidu.com/s?wd=SQL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d-myckPHNBnHN-rH7hnhf0IAYqnWm3PW64rj0d0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtznWmdrj6zn1R1nHfsnjDkPjc3) 执行期间的读一致性；   
 如果一次执行多条查询语句，例如统计查询，报表查询，在这种场景下，多条 查询[SQL](https://www.baidu.com/s?wd=SQL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d-myckPHNBnHN-rH7hnhf0IAYqnWm3PW64rj0d0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtznWmdrj6zn1R1nHfsnjDkPjc3" \t "_blank)必须保证整体的读一致性，否则，在前条[SQL](https://www.baidu.com/s?wd=SQL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d-myckPHNBnHN-rH7hnhf0IAYqnWm3PW64rj0d0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtznWmdrj6zn1R1nHfsnjDkPjc3" \t "_blank)查询之后，后条SQL 查询之前，数据被其他用户改变，则该次整体的统计查询将会出现读数据不一 致的状态，此时，应该启用事务支持  
 read-only="true"表示该事务为只读事务，比如上面说的多条查询的这种情况可 以使用只读事务，由于只读事务不存在数据的修改，因此数据库将会为只读事 务提供一些优化手段，例如[Oracle](https://www.baidu.com/s?wd=Oracle&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d-myckPHNBnHN-rH7hnhf0IAYqnWm3PW64rj0d0AP8IA3qPjfsn1bkrjKxmLKz0ZNzUjdCIZwsrBtEXh9GuA7EQhF9pywdQhPEUiqkIyN1IA-EUBtznWmdrj6zn1R1nHfsnjDkPjc3" \t "_blank)对于只读事务，不启动回滚段，不记录回 滚log。

指定只读事务的办法为：

bean配置文件中，prop属性增加“read-Only”

或者用注解方式@Transactional(readOnly=true)  
 Spring中设置只读事务是利用上面两种方式（根据实际情况）  
 在将事务设置成只读后，相当于将数据库设置成只读数据库，此时若要进行写 的操作，会出现错误。

1. **Spring的事务是如何配置的？**

先配置事务管理器TransactionManager,不同的框架有不同属性。

再配置事务通知和属性,通过tx:advice。

配置<aop:config>,设置那些方法或者类需要加入事务。

1. **事务传播行为**

Propagation.REQUIRED（spring默认的事务）

指定当前的方法必须在事务的环境下执行；

如果当前运行的方法，已经存在事务， 就会加入当前的事务；

Propagation.SUPPORTS 如果有事务则加入事务，没有则无事务运行。

Propagation.REQUIRED\_NEW

指定当前的方法必须在事务的环境下执行；

如果当前运行的方法，已经存在事务： 事务会挂起； 会始终开启一个新的 事务，执行完后； 刚才挂起的事务才继续运行。

1. **事务并发会引起什么问题,怎么解决？**

事务并发会引起脏读,幻读,不可重复读等问题,设定事务的隔离级别可以解决。

1. D
2. d
3. **Mybatis**
4. **Mybatis架构或工作流程**



1. mybatis配置

SqlMapConfig.xml，此文件作为mybatis的全局配置文件，配置了mybatis的运行环 境等信息。

mapper.xml文件即sql映射文件，文件中配置了操作数据库的sql语句。此文件需 要在SqlMapConfig.xml中加载。

1. 通过mybatis环境等配置信息构造SqlSessionFactory即会话工厂
2. 由会话工厂创建sqlSession即会话，操作数据库需要通过sqlSession进行。
3. mybatis底层自定义了Executor执行器接口操作数据库，Executor接口有两个实现，一个是基本执行器、一个是缓存执行器。
4. Mapped Statement也是mybatis一个底层封装对象，它包装了mybatis配置信息及sql映射信息等。mapper.xml文件中一个sql对应一个Mapped Statement对象，sql的id即是Mapped statement的id。
5. Mapped Statement对sql执行输入参数进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql前将输入的java对象映射至sql中，输入参数映射就是jdbc编程中对preparedStatement设置参数。
6. Mapped Statement对sql执行输出结果进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql后将输出结果映射至java对象中，输出结果映射过程相当于jdbc编程中对结果的解析处理过程。
7. **Mapper.xml中statement中属性含义**
8. id:sql语句唯一标识
9. parameterType：指定输入参数类型，mybatis通过ognl从输入对象中获取参数值拼 接在sql中。
10. resultType：指定输出结果类型，mybatis将sql查询结果的一行记录数据映射为resultType指定类型的对象。
11. resultMap：resultType可以指定pojo将查询结果映射为pojo，但需要pojo的属性名和sql查询的列名一致方可映射成功。如果sql查询字段名和pojo的属性名不一致，可以通过resultMap将字段名和属性名作一个对应关系 resultMap实质上还需要将查询结果映射到pojo对象中。

resultMap可以实现将查询结果映射为复杂类型的pojo，比如在查询结果映射对象 中包括pojo和list实现一对一查询和一对多查询。

1. #{}表示一个占位符号，通过#{}可以实现preparedStatement向占位符中设置值，自动进行java类型和jdbc类型转换，#{}可以有效防止sql注入。 #{}可以接收简单类型值或pojo属性值。 如果parameterType传输单个简单类型值，#{}括号中可以是value或其它名称。
2. ${}表示拼接sql串，通过${}可以将parameterType 传入的内容拼接在sql中且不进 行jdbc类型转换， ${}可以接收简单类型值或pojo属性值，如果parameterType传输单个简单类型值，${}括号中只能是value。
3. **SqlSessionFactory**

SqlSessionFactory是一个接口，接口中定义了openSession的不同重载方法，SqlSessionFactory的最佳使用范围是整个应用运行期间，一旦创建后可以重复使用，通常以单例模式管理SqlSessionFactory。

1. **SqlSession**

SqlSession是一个面向用户的接口， sqlSession中定义了数据库操作方法。

每个线程都应该有它自己的SqlSession实例。SqlSession的实例不能共享使用，它也是线程不安全的。因此最佳的范围是请求或方法范围。绝对不能将SqlSession实例的引用放在一个类的静态字段或实例字段中。

private static ThreadLocal<SqlSession> threadLocal = new ThreadLocal<SqlSession>();

1. **Mybatis中自主主键如何获取**

#{}:如果传入的是pojo类型,那么#{}中的变量名称必须是pojo中对应的属性.属 性.属性..... 如果要返回数据库自增主键:可以使用select LAST\_INSERT\_ID()

执行 select LAST\_INSERT\_ID()数据库函数,返回自增的主键 keyProperty:将返回 的主键放入传入参数的Id中保存.

order:当前函数相对于insert语句的执行顺序,在insert前执行是before,在insert后 执行是AFTER resultType:id的类型,也就是keyproperties中属性的类型

1. **Mybatis中uuid主键如何获取**

需要增加通过select uuid()得到uuid值

要将User中的id改成String类型，并且将User表中的id字段改为varchar(36)

1. **Mapper接口开发方法**

通常**Mybatis开发Dao方法**有两种。即原始Dao开发方法和Mapper接口开发方法。

Mapper接口开发方法只需要程序员编写Mapper接口（相当于Dao接口），由Mybatis框架根据接口定义创建接口的动态代理对象，代理对象的方法体同上边Dao接口实现类方法。

Mapper接口开发需要遵循以下规范：

1. Mapper.xml文件中的namespace与mapper接口的类路径相同。
2. Mapper接口方法名和Mapper.xml中定义的每个statement的id相同
3. Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同
4. Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同
5. **selectOne和selectList**

动态代理对象调用sqlSession.selectOne()和sqlSession.selectList()是根据mapper接口方法的返回值决定，如果返回list则调用selectList方法，如果返回单个对象则调用selectOne方法。

1. **动态sql**

通过mybatis提供的各种标签方法实现动态拼接sql。

foreach标签:循环传入的集合参数

collection:传入的集合的变量名称

item:每次循环将循环出的数据放入这个变量中

open:循环开始拼接的字符串

close:循环结束拼接的字符串

separator:循环中拼接的分隔符

where标签作用:会自动向sql语句中添加where关键字,会去掉第一个条件的and关键字。

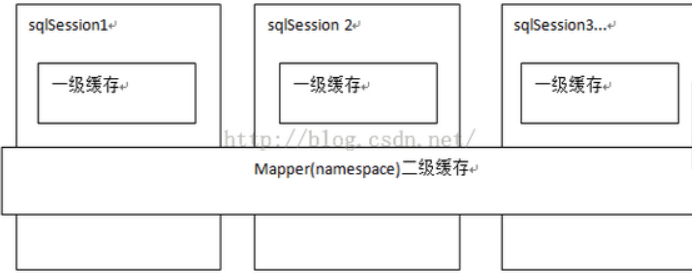
include标签：调用sql条件

1. **Mybatis缓存机制**

将用户经常查询的数据放在缓存（内存）中，用户去查询数据就不用从磁盘上(关 系型数据库数据文件)查询，从缓存中查询，从而提高查询效率，解决了高并发系 统的性能问题。

mybatis提供查询缓存，用于减轻数据压力，提高数据库性能。

mybaits提供一级缓存，和二级缓存。



**一级缓存**是SqlSession级别的缓存。在操作数据库时需要构造 sqlSession对象，在 对象中有一个(内存区域)数据结构（HashMap）用于存储缓存数据。不同的sqlSession 之间的缓存数据区域（HashMap）是互相不影响的。

一级缓存的作用域是同一个SqlSession，在同一个sqlSession中两次执行相同的sql 语句，第一次执行完毕会将数据库中查询的数据写到缓存（内存），第二次会从缓 存中获取数据将不再从数据库查询，从而提高查询效率。当一个sqlSession结束后 该sqlSession中的一级缓存也就不存在了。Mybatis默认开启一级缓存。

如果sqlSession去执行commit操作（执行插入、更新、删除），清空SqlSession中 的一级缓存，这样做的目的为了让缓存中存储的是最新的信息，避免脏读。

**二级缓存**是mapper级别的缓存，多个SqlSession去操作同一个Mapper的sql语句， 多个SqlSession去操作数据库得到数据会存在二级缓存区域，多个SqlSession可以 共用二级缓存，二级缓存是跨SqlSession的。

    二级缓存是多个SqlSession共享的，其作用域是mapper的同一个namespace，不 同的sqlSession两次执行相同namespace下的sql语句且向sql中传递参数也相同 即最终执行相同的sql语句，第一次执行完毕会将数据库中查询的数据写到缓存（内 存），第二次会从缓存中获取数据将不再从数据库查询，从而提高查询效率。Mybatis 默认没有开启二级缓存需要在setting全局参数中配置开启二级缓存。

如果缓存中有数据就不用从数据库中获取，大大提高系统性能。

**二级缓存的应用场景**

对查询频率高，变化频率低的数据建议使用二级缓存。

对于访问多的查询请求且用户对查询结果实时性要求不高，此时可采用 mybatis二级缓存技术降低数据库访问量，提高访问速度，业务场景比如：耗时较 高的统计分析sql、电话账单查询sql等。

实现方法如下：通过设置刷新间隔时间，由mybatis每隔一段时间自动清空缓 存，根据数据变化频率设置缓存刷新间隔flushInterval，比如设置为30分钟、60 分钟、24小时等，根据需求而定。

**Mybatis缓存的局限性**

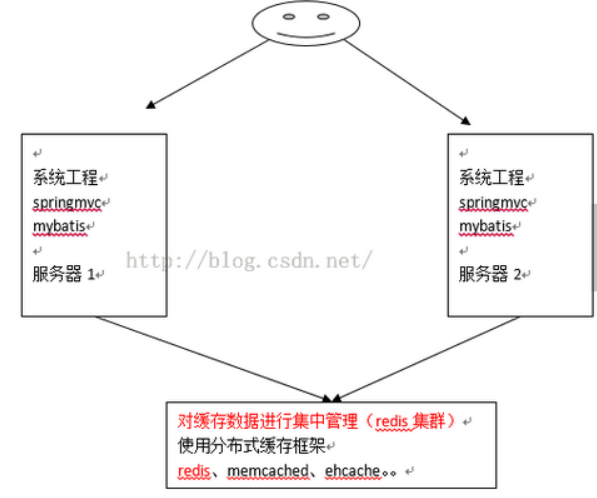
mybatis二级缓存对细粒度的数据级别的缓存实现不好，比如如下需求：对商品信 息进行缓存，由于商品信息查询访问量大，但是要求用户每次都能查询最新的商品 信息，此时如果使用mybatis的二级缓存就无法实现当一个商品变化时只刷新该商 品的缓存信息而不刷新其它商品的信息，因为mybaits的二级缓存区域以mapper 为单位划分，当一个商品信息变化会将所有商品信息的缓存数据全部清空。解决此 类问题需要在业务层根据需求对数据有针对性缓存。

1. **Mybatis实现分布式缓存**

ehcache是一个分布式缓存框架。

EhCache 是一个纯Java的进程内缓存框架，是一种广泛使用的开源Java分布式缓 存，具有快速、精干等特点，是Hibernate中默认的CacheProvider。

我们系统为了提高系统并发，性能、一般对系统进行分布式部署（集群部署方式）



不使用分布缓存，缓存的数据在各各服务单独存储，不方便系统开发。所以要使用 分布式缓存对缓存数据进行集中管理。

mybatis无法实现分布式缓存，需要和其它分布式缓存框架进行整合。

1. **Mybatis逆向工程**

使用官方网站的mapper自动生成工具mybatis-generator-core-1.3.2来生成po类和 mapper映射文件。

作用:mybatis官方提供逆向工程,可以使用它通过数据库中的表来自动生成Mapper 接口和映射文件(单表增删改查)和Po类.

1. **Mybatis解决jdbc编程的问题**
2. 数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。

解决：在SqlMapConfig.xml中配置数据链接池，使用连接池管理数据库链接。

1. Sql语句写在代码中造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

解决：将Sql语句配置在XXXXmapper.xml文件中与java代码分离。

1. 向sql语句传参数麻烦，因为sql语句的where条件不一定，可能多也可能少，占位符需要和参数一一对应。

解决：Mybatis自动将java对象映射至sql语句，通过statement中的 parameterType定义输入参数的类型。

1. 对结果集解析麻烦，sql变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成pojo对象解析比较方便。
2. **Hibernate与Mybatis对比**

**首先简单介绍下两者的概念**

Hibernate ：Hibernate 是当前最流行的ORM框架，对数据库结构提供了较为完整 的封装。

Mybatis：Mybatis同样也是非常流行的ORM框架，主要着力点在于POJO 与SQL 之间的映射关系。

其次具体从几个方面说一下两者的区别：

**1.两者最大的区别**

针对简单逻辑，Hibernate和MyBatis都有相应的代码生成工具，可以生成简单基 本的DAO层方法。

针对高级查询，Mybatis需要手动编写SQL语句，以及ResultMap。而Hibernate有 良好的映射机制，开发者无需关心SQL生成与结果映射，可更专注于业务流程。

**2.开发难度对比**

Hibernate的开发难度要大于Mybatis。主要由于Hibernate比较复杂、庞大，学习 周期较长。

而Mybatis则相对简单一些，并且Mybatis主要依赖于sql的书写，让开发者感觉 更熟悉。

**3.sql书写比较**

Mybatis的SQL是手动编写的，所以可以按需求指定查询的字段。不过没有自己的 日志统计，所以要借助log4j来记录日志。

Hibernate也可以自己写SQL来指定需要查询的字段，但这样就破坏了Hibernate开 发的简洁性。不过Hibernate具有自己的日志统计。

**4.数据库扩展性比较**

Mybatis由于所有SQL都是依赖数据库书写的，所以扩展性，迁移性比较差。

Hibernate与数据库具体的关联都在XML中，所以HQL对具体是用什么数据库并不 是很关心。

**5.缓存机制比较**

相同点：Hibernate和Mybatis的二级缓存除了采用系统默认的缓存机制外，都可 以通过实现自己的缓存或为其他第三方缓存方案，创建适配器来覆盖缓存行为。

不同点：Hibernate的二级缓存配置在SessionFactory生成的配置文件中进行详细配 置，然后再在具体的表-对象映射中配置是那种缓存。

MyBatis的二级缓存配置都是在每个具体的表-对象映射中进行详细配置，这样针对 不同的表可以自定义不同的缓存机制。并且Mybatis可以在命名空间中共享相同的 缓存配置和实例，通过Cache-ref来实现。

两者比较：因为Hibernate对查询对象有着良好的管理机制，用户无需关心SQL。 所以在使用二级缓存时如果出现脏数据，系统会报出错误并提示。

而MyBatis在这一方面，使用二级缓存时需要特别小心。如果不能完全确定数据更 新操作的波及范围，避免Cache的盲目使用。否则，脏数据的出现会给系统的正常 运行带来很大的隐患。

**6.总结**

Hibernate与MyBatis都可以是通过SessionFactoryBuider由XML配置文件生成 SessionFactory，然后由SessionFactory 生成Session，最后由Session来开启执行事 务和SQL语句。

而MyBatis的优势是MyBatis可以进行更为细致的SQL优化，可以减少查询字段， 并且容易掌握。

Hibernate的优势是DAO层开发比MyBatis简单，Mybatis需要维护SQL和结果映射。 数据库移植性很好，MyBatis的数据库移植性不好，不同的数据库需要写不同SQL。 有更好的二级缓存机制，可以使用第三方缓存。MyBatis本身提供的缓存机制不佳。

1. D
2. d
3. **SpringMVC**
4. **Spring MVC比较 Struts2**
5. springmvc的入口是一个servlet即前端控制器，而struts2入口是一个filter过虑器。
6. springmvc是基于方法开发(一个url对应一个方法)，请求参数传递到方法的形参，可以设计为单例或多例(建议单例)，struts2是基于类开发，传递参数是通过类的属性，只能设计为多例。
7. Struts采用值栈存储请求和响应的数据，通过OGNL存取数据， springmvc通过参数解析器是将request请求内容解析，并给方法形参赋值，将数据和视图封装成ModelAndView对象，最后又将ModelAndView中的模型数据通过reques域传输到页面。Jsp视图解析器默认使用jstl。
8. **SpringMVC处理流程**



1. 向服务器发送HTTP请求，请求被前端控制器 DispatcherServlet 捕获。
2. DispatcherServlet 根据 <servlet-name>-servlet.xml 中的配置对请求的URL进行解析，得到请求资源标识符（URI）。 然后根据该URI，调用 HandlerMapping 获得该Handler配置的所有相关的对象（包括Handler对象以及Handler对象对应的拦截器），最后以 HandlerExecutionChain 对象的形式返回。
3. DispatcherServlet 根据获得的Handler，选择一个合适的 HandlerAdapter。
4. 提取Request中的模型数据，填充Handler入参，开始执行Handler（Controller)。 在填充Handler的入参过程中，根据你的配置，Spring将帮你做一些额外的工作：

HttpMessageConveter： 将请求消息（如Json、xml等数据）转换成一个对象，将对象转换为指定的响应信息。

数据转换：对请求消息进行数据转换。如String转换成Integer、Double等。

数据格式化：对请求消息进行数据格式化。 如将字符串转换成格式化数字或格式化日期等。

数据验证： 验证数据的有效性（长度、格式等），验证结果存储到BindingResult或Error中。

1. Handler(Controller)执行完成后，向 DispatcherServlet 返回一个 ModelAndView 对象
2. 根据返回的ModelAndView，选择一个适合的 ViewResolver（必须是已经注册到Spring容器中的ViewResolver)返回给DispatcherServlet。
3. ViewResolver 结合Model和View，来渲染视图。
4. 视图负责将渲染结果返回给客户端
5. **SpringMVC中的组件及各个组件的作用**

1.DispatherServlet：前置控制器，负责接收并处理所有的web请求，根据handlerMapping找到具体的Controller，由controller完成具体的处理逻辑。

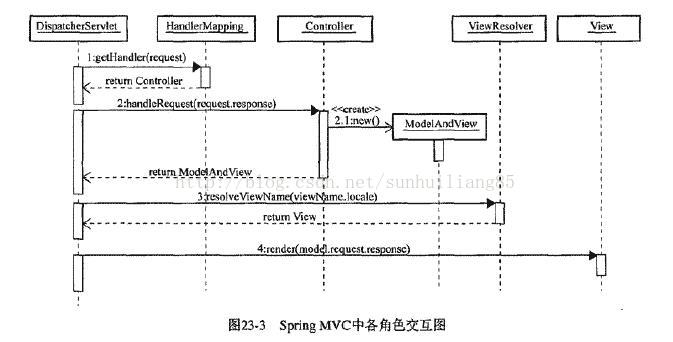
2.HandlerMapping：负责处理web请求和具体的Controller之间的映射关系匹配。

3.Controller：DispatherServlet的次级控制器，web请求的具体处理者。DispatherServlet获得handlerMapping的返回结果后，调用controller的处理方法处理当前的业务请求，处理完成后返回ModelAndView对象。

4.ViewResolver：用来处理视图名与具体的view实例之间的映射对应关系。根据ModelAndView中的视图名查找相应的View实现类，然后将查找的结果返回给DispatcherServlet，DispatcherServlet最终会将ModelAndView中的模型数据交给返回的View处理最终的视图渲染工作。

5.View：为支持多种视图技术,统一抽象视图的生成策略，根据模型数据输出具体的视图。

而存在，统一抽象视图的生成策略，根据模型数据输出具体的视图。



1. **Spring MVC 常用的注解**

@RequestMapping设置请求映射url

@RequestParam常用于处理简单类型的绑定

@RequestBody注解用于读取http请求的内容(字符串)，通过springmvc提供的 HttpMessageConverter接口将读到的内容转换为json、xml等格式的数据并绑 定到controller方法的参数上。

@ResponseBody该注解用于将Controller的方法返回的对象，通过 HttpMessageConverter接口转换为指定格式的数据如：json,xml等，通过 Response响应给客户端

1. **如何使用SpringMVC 完成 JSON 操作**

配置 MappingJacksonHttpMessageConverter

如果使用mvc:annotation-driven就不需要配置。

使用 @RequestBody 注解作为请求参数或 ResponseBody作为返回值

1. **mvc:annotation-driven**

使用mvc:annotation-driven代替注解映射器和注解适配器配置器

mvc:annotation-driven默认加载很多的参数绑定方法，比如json转换解析器就默认加载了，如果使用mvc:annotation-driven就不用配置

RequestMappingHandlerMapping和RequestMappingHandlerAdapter

1. **restful**

一种软件架构风格，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。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。（强调以资源为导向）

所谓"资源"，就是网络上的一个实体，或者说是网络上的一个具体信息。它可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的实在。你可以用一个URI（统一资源定位符）指向它，每种资源对应一个特定的URI。要获取这个资源，访问它的URI就可以，因此URI就成了每一个资源的地址或独一无二的识别符。

资源定位：互联网所有的事物都是资源，要求url中没有动词，只有名词。没有参数

Url格式：http://blog.csdn.net/beat\_the\_world/article/details/45621673

（使用restful后，utl中不能用？传参或.action）

资源操作：使用put、delete、post、get，使用不同方法对资源进行操作。分别对应添加、删除、修改、查询。一般使用时还是post和get。Put和Delete几乎不使用。

客户端用到的手段，只能是HTTP协议。具体来说，就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。

它们分别对应四种基本操作：GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源。

URI 的设计只要负责把资源通过合理方式暴露出来就可以了。对资源的操作与它无关，操作是通过 HTTP动词来体现，所以REST 通过 URI 暴露资源时，会强调不要在 URI 中出现动词。

比如：左边是错误的设计，而右边是正确的

GET /rest/api/getDogs --> GET /rest/api/dogs 获取所有小狗狗

GET /rest/api/addDogs --> PUT /rest/api/dog 添加一个小狗狗

GET /rest/api/editDogs/:dog\_id --> POST /rest/api/dogs/:dog\_id 修改一个小 狗狗

GET /rest/api/deleteDogs/:dog\_id --> DELETE /rest/api/dogs/:dog\_id 删除一 个小狗狗

左边的这种设计，很明显不符合REST风格，上面已经说了，URI 只负责准确无误的暴露资源，而 getDogs/addDogs...已经包含了对资源的操作，这是不对的。相反右边却满足了，它的操作是使用标准的HTTP动词来体现。

1. **SpringMvc里面拦截器是怎么实现的？**

有两种方式,一种是实现接口,另外一种是继承适配器类,然后在SpringMvc的配置文件中配置拦截器即可。



**使用场景：**

用户登录判断，在执行Action的前面判断是否已经登录，如果没有登录的跳转到登录页面。

用户权限判断，在执行Action的前面判断是否具有，如果没有权限就给出提示信息。

操作日志

1. D
2. **SpringBoot**
3. **Springboot和springcloud？**

现在很多公司已经开始使用微服务架构(SpringCloud或Dubbo)，而SpringBoot对于微服务的环境支持很好。

* 微架构，与Spring4一起诞生。比如@RestController
* 可以快速上手，整合一些子项目(开源框架或第三方开源库)
* 可以依赖很少的配置就可以快速搭建并运行项目
* 基于spring，开发者快速入门，门槛很低。
* Springboot可以创建独立运行的应用而不依赖容器
* 不需要打成war包，可以放入tomcat中直接运行。
* 提供maven极简配置，缺点是会引入些项目不需要的包。
* 提供可视化的相关功能，方便监控，比如性能、应用的健康程度。
* 简化配置，不需要过多的xml
* 为微服务springClouod铺路，springBoot可以整合各种架构来构建微服务，比如dubbo、thrift等。

Spring Cloud是一系列框架的有序集合。它利用Spring Boot的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用Spring Boot的开发风格做到一键启动和部署。Spring Cloud并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过Spring Boot风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。

微服务架构需要的功能或使用场景

1：我们把整个系统根据业务拆分成几个子系统。

2：每个子系统可以部署多个应用，多个应用之间使用负载均衡。

3：需要一个服务注册中心，所有的服务都在注册中心注册，负载均衡也是通过在注册中心注册的服务来使用一定策略来实现。

4：所有的客户端都通过同一个网关地址访问后台的服务，通过路由配置，网关来判断一个URL请求由哪个服务处理。请求转发到服务上的时候也使用负载均衡。

5：服务之间有时候也需要相互访问。例如有一个用户模块，其他服务在处理一些业务的时候，要获取用户服务的用户数据。

6：需要一个断路器，及时处理服务调用时的超时和错误，防止由于其中一个服务的问题而导致整体系统的瘫痪。

7：还需要一个监控功能，监控每个服务调用花费的时间等。

目前主流的微服务框架：Dubbo、 SpringCloud、thrift、Hessian等，目前国内的中小企业用的大多数都是Dubbo，SpringCloud估计很少，也许有些开发同学都没听说过。

　springCloud是基于SpringBoot的一整套实现微服务的框架。他提供了微服务开发所需的配置管理、服务发现、断路器、智能路由、微代理、控制总线、全局锁、决策竞选、分布式会话和集群状态管理等组件。最重要的是，跟spring boot框架一起使用的话，会让你开发微服务架构的云服务非常好的方便。

SpringBoot旨在简化创建产品级的 Spring 应用和服务，简化了配置文件，使用嵌入式web服务器，含有诸多开箱即用微服务功能

1. S
2. s
3. **数据库**
4. **基础**
5. **SQL概念**

SQL（Structured Query Language 结构查询语言）是一个功能强大的数 据库语言。SQL是关系数据库管理系统的标准语言。

DML（数据操作语言）: 用于检索或修改数据

DDL（数据定义语言）:用于定义数据的结构，创建、修改、删除数据库对象

DCL（数据控制语言）: 用于定义数据库用户的权限

1. **常用函数**

**字符函数**

UPPER: 将输入的字符串变为大写返回

Select upper(‘hello’) from dual; //dual为数据库提供的一张虚拟表

LOWER: 将输入的字符串变为小写返回

Select lower(ename) from emp;

INITCAP: 开头首字母大写

Select initcap(ename) from emp;

LENGTH: 求出字符串的长度

Select ename, length(ename) from emp;

查询出姓名长度为5的雇员信息

Select ename,length(ename) from emp where length(ename)=5;

REPLACE: 字符串进行替换

使用”\_”替换姓名中所有字母”A”

Select replace(ename,’A’,’\_’) from emp;

SUBSTR: 字符串截取

Select ename,substr(ename,0,3) from emp ;

截取每个雇员姓名的后三个字母

Select ename,substr(ename,length(ename)-2) from emp;

Select ename,substr(ename,-3) from emp;

**数字函数**

ROUND: 四舍五入的操作

TRUNC: 截取指定位置的内容

MOD: 取模或取余

**通用函数**

NVL() 处理null

查询出每个雇员的全部年薪

Select ename,sal,comm,(sal+comm)\*12 from emp; //此时comm字段有null值

//此时要将null变为0

Select ename,sal,comm,(sal+nvl(comm,0))\*12,nvl(comm,0) from emp;

DECODE() 多数值判断

类似于if…else语，不同的是decode()函数判断的是数值，而不是逻辑条件。

显示全部雇员的职位，但是这些职位要求替换为中文显示

Select empno,ename,job,decode(job,’CLERK’,’办事员’,’SALESMAN’,’销售人员’,’MANAGER’,’经理’ ,’ANALYST’,’分析员’ ,’PRESIDENT’,’总裁’) from emp

1. **常用的数据字段(oracle)**

每一张数据表实际是由若干字段组成，而每一个字段会有其对应的数据类型。

|  |  |  |
| --- | --- | --- |
| **数据类型** | **关键字** | **描述** |
| 字符串 | VARCHAR2(n) | n表示字符串所能保存的最大长度200字符左右 |
| 整数 | NUMBER(n) | 最多为n位的整数，有时可也可以用INT代替 |
| 小数 | NUMBER(n,m) | m为小数位，n-m为整数位，可以用FLOAT代替 |
| 日期 | DATE | 存放日期时间 |
| 大文本 | CLOB | 存储海量文字(4G) |
| 大对象 | BLOB | 存放二进制数据,如:电影、mp3、图片、文字 |

1. **集合操作**

在数学的操作中存在交、差、并、补的概念，而在数据的查询中也存在此概念。

UNION 连接两个，相同部分不显示。

UNION ALL连接两个，相同部分显示。

INTERSECT 返回两个查询中相同部分

MINUS 返回两个查询中不同部分

1. **视图和游标**

视图时一种虚拟表，具有与物理表相同的功能，可对视图增删改查。视图通常是一个或多个表的行或列的子集。对视图的修改不影响基本表。使获取数据更容易，相比多表查询。

游标: 对查询出来的结果集做一个单元来有效处理。游标可定在该单元中的特定行，从结果集的当前行检索一行或多行。可对结果集当前行做修改。

一般不使用游标,但当需逐条处理数据时,游标显得很重要。

1. **数据库视图和表的区别，什么是视图？怎么使用视图？物化视图？**

视图是种虚表，不是真正的物理表，只是为业务查询方便 ，将几张物理表虚拟组成一个视图。

第一点：

使用视图，可以定制用户数据，聚焦特定的数据。

解释：

在实际过程中，公司有不同角色的工作人员，我们以销售公司为例的话，

采购人员，可以需要一些与其有关的数据而与他无关的数据，对他没

有任何意义，我们可以根据这一实际情况，专门为采购人员创建一个视

图，以后他在查询数据时，只需select \* from view\_caigou 就可以啦。

第二点：使用视图，可以简化数据操作。

解释：我们在使用查询时，在很多时候我们要使用聚合函数，同时还要

显示其它字段的信息，可能还会需要关联到其它表，这时写的语句可能

会很长，如果这个动作频繁发生的话，我们可以创建视图，这以后，我

们只需要select \* from view1就可以

第三点：使用视图，基表中的数据就有了一定的安全性

因为视图是虚拟的，物理上是不存在的，只是存储了数据的集合，我们可以

将基表中重要的字段信息，可以不通过视图给用户，视图是动态的数据的集

合，数据是随着基表的更新而更新。同时，用户对视图，不可以随意的更改

和删除，可以保证数据的安全性。

第四点：可以合并分离的数据，创建分区视图

随着社会的发展，公司的业务量的不断的扩大，一个大公司，下属都设有很

多的分公司，为了管理方便，我们需要统一表的结构，定期查看各公司业务

情况，而分别看各个公司的数据很不方便，没有很好的可比性，如果将这些

数据合并为一个表格里，就方便多啦，这时我们就可以使用union关键字，

将各分公司的数据合并为一个视图。

1. **数据库的索引有什么用，带来的问题是什么?**

**1、索引定义**

数据库[索引](http://baike.baidu.com/view/262241.htm)好比是一本书前面的目录，能加快数据库的查询速度。[索引](http://baike.baidu.com/view/262241.htm)是对数据库表中一个或多个列（例如，employee 表的姓氏 (lname) 列）的值进行排序的结构。如果想按特定职员的姓来查找他或她，则与在表中搜索所有的行相比，索引有助于更快地获取信息。

**2、建立索引的优缺点**

**优点：** 1.大大加快数据的检索速度;   
 2.创建唯一性索引，保证数据库表中每一行数据的唯一性;   
 3.加速表和表之间的连接;   
 4.在使用分组和排序子句进行数据检索时，可以显著减少查询中分组和排 序的时间。

**缺点：**  
　　1.索引需要占用数据表以外的物理存储空间  
　　2.创建索引和维护索引要花费一定的时间  
　　3.当对表进行更新操作时，索引需要被重建，这样降低了数据的维护速度。

**3、索引类型**  
 根据数据库的功能，可以在数据库设计器中创建索引：**唯一索引、主键索引和聚集索引**。 尽管唯一索引有助于定位信息，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： UNIQUE

例如：create unique index stusno on student（sno）；  
表明此索引的每一个索引值只对应唯一的数据记录，对于单列惟一性索引，这保证单列不包含重复的值。对于多列惟一性索引，保证多个值的组合不重复。

主键索引： primary key  
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](http://www.cnblogs.com/aspnethot/articles/1504082.html)（也叫聚簇索引）：cluster   
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。

**4、索引的实现方式**

**1、** [**B+树**](http://blog.csdn.net/kennyrose/article/details/7532032)

经常听到B+树就是这个概念，用这个树的目的和红黑树差不多，也是为了尽量保持树的平衡，当然红黑树是二叉树，但B+树就不是二叉树了，节点下面可以有多个子节点，数据库开发商会设置子节点数的一个最大值，这个值不会太小，所以B+树一般来说比较矮胖，而红黑树就比较瘦高了。  
关于B+树的插入，删除，会涉及到一些算法以保持树的平衡，这里就不详述了。ORACLE的默认索引就是这种结构的。  
如果经常需要同时对两个字段进行AND查询,那么使用两个单独索引不如建立一个复合索引，因为两个单独索引通常数据库只能使用其中一个，而使用复合索引因为索引本身就对应到两个字段上的，效率会有很大提高。  
**2、散列索引** 第二种索引叫做散列索引，就是通过散列函数来定位的一种索引，不过很少有单独使用散列索引的，反而是散列文件组织用的比较多。  
散列文件组织就是根据一个键通过散列计算把对应的记录都放到同一个槽中，这样的话相同的键值对应的记录就一定是放在同一个文件里了，也就减少了文件读取的次数，提高了效率。  
散列索引呢就是根据对应键的散列码来找到最终的索引项的技术，其实和B树就差不多了，也就是一种索引之上的二级辅助索引，我理解散列索引都是二级或更高级的稀疏索引，否则桶就太多了，效率也不会很高。  
**3、位图索引**  
 位图索引是一种针对多个字段的简单查询设计一种特殊的索引，适用范围比较小，只适用于字段值固定并且值的种类很少的情况，比如性别，只能有男和女，或者级别，状态等等，并且只有在同时对多个这样的字段查询时才能体现出位图的优势。  
位图的基本思想就是对每一个条件都用0或者1来表示，如有5条记录，性别分别是男，女，男，男，女，那么如果使用位图索引就会建立两个位图，对应男的10110和对应女的01001,这样做有什么好处呢，就是如果同时对多个这种类型的字段进行and或or查询时，可以使用按位与和按位或来直接得到结果了。  
B+树最常用，性能也不差，用于范围查询和单值查询都可以。特别是范围查询，非得用B+树这种顺序的才可以了。  
HASH的如果只是对单值查询的话速度会比B+树快一点，但是ORACLE好像不支持HASH索引，只支持HASH表空间。  
位图的使用情况很局限，只有很少的情况才能用，一定要确定真正适合使用这种索引才用（值的类型很少并且需要复合查询），否则建立一大堆位图就一点意义都没有了。

1. **索引失效**

1.如果条件中有or，即使其中有条件带索引也不会使用(这也是为什么尽量少 用or的原因) 注意：要想使用or，又想让索引生效，只能将or条件中的每 个列都加上索引

2.对于多列索引，不是使用的第一部分，则不会使用索引

3.like查询是以%开头

4.如果列类型是字符串，那一定要在条件中将数据使用引号引用起来,否则不 使用索引

5.如果mysql估计使用全表扫描要比使用索引快,则不使用索引

此外，查看索引的使用情况

show status like ‘Handler\_read%’;

大家可以注意：

handler\_read\_key:这个值越高越好，越高表示使用索引查询到的次数

1. **怎么用索引?建立索引的思想？**

索引是建立在数据库表中的某些列的上面。在创建索引的时候，应该考虑在哪些列上可以创建索引，在哪些列上不能创建索引。一般来说，应该在这些列上创建索引：

在经常需要搜索的列上，可以加快搜索的速度；

在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；

在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；在经常需要根据范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的；

在经常需要排序的列上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；

在经常使用在WHERE子句中的列上面创建索引，加快条件的判断速度。

同样，对于有些列不应该创建索引。一般来说，不应该创建索引的这些列具有下列特点：

第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

第三，对于那些定义为text, image和bit数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少,不利于使用索引。

第四，当修改性能远远大于检索性能时，不应该创建索引。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改操作远远多于检索操作时，不应该创建索引。

1. **如何防止sql注入?**

1）简单又有效的方法是使用PreparedStatement

采用预编译语句集，它内置了处理SQL注入的能力，只要使用它的setXXX方法传值即可。

使用好处：

(1).代码的可读性和可维护性.

(2).PreparedStatement尽最大可能提高性能.

(3).最重要的一点是极大地提高了安全性.

原理：

sql注入只对sql语句的准备(编译)过程有破坏作用而PreparedStatement已经准备好了,执行阶段只是把输入串作为数据处理,而不再对sql语句进行解析,准备,因此也就避免了sql注入问题.

2）使用正则表达式过滤传入的参数

3）使用javascript在客户端进行不安全字符屏蔽

功能介绍：检查是否含有”‘”,”\\”,”/”

参数说明：要检查的字符串

返回值：0：是1：不是

1. D
2. dd
3. **事务**
4. **数据库引擎种类及区别**

**数据库引擎**是用于存储、处理和保护数据的核心服务。利用数据库引擎可控制访问权限并快速处理事务，从而满足企业内大多数需要处理大量数据的应用程序的要求。 使用数据库引擎创建用于联机事务处理或联机分析处理数据的关系数据库。这包括创建用于存储数据的表和用于查看、管理和保护数据安全的数据库对象（如索引、视图和存储过程）。

Innodb引擎: Innodb引擎提供了对数据库ACID事务的支持，并且实现了SQL标准的四种隔离级别，该引擎还提供了行级锁和外键约束，它的设计目标是处理大容量数据库系统，它本身其实就是基于MySQL后台的完整数据库系统，MySQL运行时Innodb会在内存中建立缓冲池，用于缓冲数据和索引。但是该引擎不支持FULLTEXT类型的索引，而且它没有保存表的行数，当SELECT COUNT(\*) FROM TABLE时需要扫描全表。当需要使用数据库事务时，该引擎当然是首选。由于锁的粒度更小，写操作不会锁定全表，所以在并发较高时，使用Innodb引擎会提升效率。但是使用行级锁也不是绝对的，如果在执行一个SQL语句时MySQL不能确定要扫描的范围，InnoDB表同样会锁全表。

InnoDB是一个事务型的存储引擎，有行级锁定和外键约束，适用于以下的场合：

1. 更新多的表，适合处理多重并发的更新请求。

2. 支持事务。

3. 可以从灾难中恢复（通过bin-log日志等）。

4. 外键约束。只有他支持外键。

5. 支持自动增加列属性auto\_increment。

ACID

A 事务的原子性(Atomicity)：指一个事务要么全部执行,要么不执行.也就是说一个事务不可能只执行了一半就停止了.比如你从取款机取钱,这个事务可以分成两个步骤:1划卡,2出钱.不可能划了卡,而钱却没出来.这两步必须同时完成.要么就不完成.

C 事务的一致性(Consistency)：指事务的运行并不改变数据库中数据的一致性.例如,完整性约束了a+b=10,一个事务改变了a,那么b也应该随之改变.

I 独立性(Isolation）:事务的独立性也有称作隔离性,是指两个以上的事务不会出现交错执行的状态.因为这样可能会导致数据不一致.

D 持久性(Durability）:事务的持久性是指事务执行成功以后,该事务所对数据库所作的更改便是持久的保存在数据库之中，不会无缘无故的回滚.

MyIASM引擎

　　MyIASM是MySQL默认的引擎，但是它没有提供对数据库事务的支持，也不支持行级锁和外键，因此当INSERT(插入)或UPDATE(更新)数据时即写操作需要锁定整个表，效率便会低一些。不过和Innodb不同，MyIASM中存储了表的行数，于是SELECT COUNT(\*) FROM TABLE时只需要直接读取已经保存好的值而不需要进行全表扫描。如果表的读操作远远多于写操作且不需要数据库事务的支持，那么MyIASM也是很好的选择。

1.  不支持事务，但是并不代表着有事务操作的项目不能用MyIsam存储引擎， 可以在service层进行根据自己的业务需求进行相应的控制。

2.  不支持外键。

3.  查询速度很快，如果数据库insert和update的操作比较多的话比较适用。

4.  对表进行加锁。

两种引擎的选择

　　大尺寸的数据集趋向于选择InnoDB引擎，因为它支持事务处理和故障恢复。数据库的大小决定了故障恢复的时间长短，InnoDB可以利用事务日志进行数据恢复，这会比较快。主键查询在InnoDB引擎下也会相当快，不过需要注意的是如果主键太长也会导致性能问题。大批的INSERT语句(在每个INSERT语句中写入多行，批量插入)在MyISAM下会快一些，但是UPDATE语句在InnoDB下则会更快一些，尤其是在并发量大的时候。

1. **事务**

[数据库事务](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E4%BA%8B%E5%8A%A1)(Database Transaction) ，是指作为单个逻辑工作单元执行的一系列[操作](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C/33052" \t "_blank)，要么完全地执行，要么完全地不执行。一个逻辑工作单元要成为事务，必须满足所谓的ACID（原子性、一致性、隔离性和持久性）属性。

1. **事务和锁**

事务是绑定到一起作为一个逻辑工作单元的sql语句分组，如任一个语句操作失败就视作整个操作失败。操作将回滚到操作前状态。

锁: 实现事务的关键。可以保证事务的完整性和并发性。可以使某些数据的拥有者在某段时间内不能使用某些数据或数据结构。

1. D
2. dd
3. **存储过程和触发器**
4. **存储过程和函数**

存储过程重在处理数据，函数可以返回值

存储过程是procedure用户定义的一系列sql语句的集合，涉及特定表或其它对象 的任务，用户可以调用存储过程

函数通常是数据库已定义的方法，它接收参数并返回某种类型的值并且不涉及特定 用户表。

    存储过程和函数存在以下几个区别：

1. 一般来说，存储过程实现的功能要复杂一点，而函数的实现的功能针对性比较强。

存储过程，功能强大，可以执行包括修改表等一系列数据库操作；用户定义函数不能用于执行一组修改全局数据库状态的操作。

    2）对于存储过程来说可以返回参数，如记录集，而函数只能返回值或者表对象。函数只 能返回一个变量；而存储过程可以返回多个。存储过程的参数可以有IN,OUT,INOUT三 种类型，而函数只能有IN类~~存储过程声明时不需要返回类型，而函数声明时需要描 述返回类型，且函数体中必须包含一个有效的RETURN语句。

    3）存储过程一般是作为一个独立的部分来执行（ EXECUTE 语句执行），而函数可以作为 查询语句的一个部分来调用（SELECT调用），由于函数可以返回一个表对象，因此它可 以在查询语句中位于FROM关键字的后面。 SQL语句中不可用存储过程，而可以使用 函数。

1. **存储过程的概念、优点（或特点），写一个简单的存储过程**

存储过程: 是一组为了完成特定功能的SQL语句集,利用SQL Server所提供的T-SQL语言所编写的程序,经 编译后存储在数据库中。

优点:

1.执行速度快。存储过程只在创造时进行编译,以后每次执行不需再重新编译,,一般SQL语句每执行一次就编译一次

2.存储过程可重复使用

3.安全性高。(可设定只有某些用户才具有对指定存储过程的使用权)

4.当对数据库进行复杂操作时,可完成复杂的判断和较复杂的运算,可用存储过程封装起来

5.易于维护和集中控制。当企业规则变化时在服务器中改变存储过程即可,无须修改应用程序

简单的存储过程:

create proc select\_query @year int

as

select \* from tmp where year=@year

1. **触发器**
2. 触发器:触发器可以看成一个特殊的存储过程，存储过程是要显式调用去完成，而触发器可以自动完成。比如:当数据库中的表发生增删改操作时，对应的触发器就可以执行对应的PL/SQL语句块。
3. 作用:维护表的完整性，记录表的修改来审计表的相关信息。

分为:

DML触发器: 当数据库服务器中发生数据操作语言事件时执行的存储过程。分为:After触发器和Instead Of触发器

DDL触发器 特殊的触发器,在响应数据定义语言(DDL)语句时触发。一般用于数据库中执行管理任务

DDL触发器是响应Create、 Alter或Drop开头的语句而激活

1. **D**
2. **d**
3. **Sql**
4. **Sql语法**

SELECT[DISTINCT] \* |分组字段1 [别名] [,分组字段2 [别名],…] | 统计函数

FROM表名　［表名］，［表名，［别名］，．．．］

[WHERE 条件]

[GROUP BY 分组字段1 [,分组字2,…]]

[HAVING 分组后的过滤条件]

[ORDER BY 排序字段 ASC｜DESC [,排序字段 ASC |DESC]];

1. **左、右连接**

**1、内联接**（典型的联接运算，使用像 =  或 <> 之类的比较运算符）。包括相等联接和自然联接。       
内联接使用比较运算符根据每个表共有的列的值匹配两个表中的行。例如，检索 students和courses表中学生标识号相同的所有行。     
**2、外联接。**外联接可以是左向外联接、右向外联接或完整外部联接。       
在 FROM子句中指定外联接时，可以由下列几组关键字中的一组指定：       
1）LEFT  JOIN或LEFT OUTER JOIN       
左向外联接的结果集包括  LEFT OUTER子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值。         
2）RIGHT  JOIN 或 RIGHT  OUTER  JOIN       
右向外联接是左向外联接的反向联接。将返回右表的所有行。如果右表的某行在左表中没有匹配行，则将为左表返回空值。         
3）FULL  JOIN 或 FULL OUTER JOIN  
完整外部联接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值。如果表之间有匹配行，则整个结果集行包含基表的数据值。     
  
**3、交叉联接**交叉联接返回左表中的所有行，左表中的每一行与右表中的所有行组合。交叉联接也称作笛卡尔积。      
FROM 子句中的表或视图可通过内联接或完整外部联接按任意顺序指定；但是，用左或右向外联接指定表或视图时，表或视图的顺序很重要。有关使用左或右向外联接排列表的更多信息，请参见使用外联接。

1. **数据库设计的三范式**

**第一范式1NF:做到每列不可拆分**

中国北京可以拆分成两个字段中国和北京

**第二范式2NF:确保一个表只做一件事情**

id name sex address grade

1 tom male beijin 80

对于grade可以专门定义一张成绩表

**第三范式3NF:在满足2NF下，消除表中的传递依赖**

推理规则：A->B,B->C,则A->C 如果这种情况存在，就要消除这种关系。目的是为 了使空间最省。

id name number price totalprice

1 风扇 20 200 4000

id是A , number和price是B totalprice是C

id ->number,price->totalprice

可以将totalprice字段删除，节省空间。

**反三范式（用空间换时间）：**在进行数据库设计时，首先要确保表结构要达到第三 范式的要求。但在实际开发时，为满足业务的需要，会在符合三范式的表中加入冗 余字段，结果导致不符合第三范式的要求。

因为存在冗余字段需要维护它，如购销合同中的合同总金额就是一个冗余字段（由 于在页面上要显示总金额，不将计算的操作放在页面上进行，而是在数据库中计算 好直接获取到页面上。），就需要在添加货物或添加附件时，都要去更新购销合同中 的总金额。

1. **分页**

**Mysql**

0表示第一条记录的索引号，索引号从0开始, 2表示最多选取二个记录

select \* from users limit 0,2或 select \* from users limit 2;

查询出users第2条到第4条记录

select \* from users limit 1,3;

**Oracle**

方式一使用集合减运算: 显示emp表中3-8条记录

select rownum "伪列",emp.\* from emp where rownum<=8

minus

select rownum,emp.\* from emp where rownum<=2;

方式二：使用子查询，在from子句中使用，重点

select xx.\*

from (select rownum ids,emp.\* from emp where rownum<=8) xx

where ids>=3;

其中的xx是子查询结果表的别名

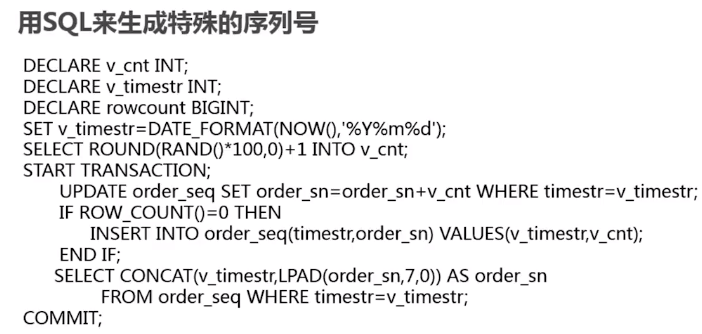
1. **唯一序列号**

Mysql: auto\_increment

Oracle:sequence

分布式系统：snowflake 雪花算法，适用于订单号

用sql来成生特殊的序列号



1. **常用SQL语句**
2. 创建数据库   
   CREATE DATABASE database-name;
3. 删除数据库   
   DROP DATABASE database-name;
4. 创建新表   
   create table depart (dept\_id int(11) NOT NULL AUTO\_INCREMENT,   
   dept\_name varchar(255) DEFAULT NULL, PRIMARY KEY (dept\_id));   
   根据已有的表创建新表：   
   create table tab\_new like tab\_old (使用旧表B创建新表A)   
   备注：此种方式在将表B复制到A时候会将表B完整的字段结构和索引复制到表A中来   
   create table tab\_new as select col1,col2… from tab\_old definition only   
   备注：此种方式只会将表B的字段结构复制到表A中来，但不会复制表B中的索引到表A中来。这种方式比较灵活可以在复制原表表结构的同时指定要复制哪些字段，并且自身复制表也可以根据需要增加字段结构。   
   create table as select 会将原表中的数据完整复制一份，但表结构中的索引会丢失。   
   create table like 只会完整复制原表的建表语句，但不会复制数据。
5. 删除新表   
   drop table tabname；
6. 增加一个列   
   alter table tabname add column column\_name type
7. 添加主键： Alter table tabname add primary key(col)   
   删除主键： Alter table tabname drop primary key   
   一个数据表只可以有一个主键，所以不存在删除某一列的主键.
8. 创建索引：create [unique] index idxname on tabname(col….)   
   删除索引：drop index idxname   
   注：索引是不可更改的，想更改必须删除重新建。
9. 创建视图：create view viewname as select statement   
   删除视图：drop view viewname
10. 几个简单的基本的sql语句   
    选择：select \* from table1 where 范围   
    插入：insert into table1(field1,field2) values(value1,value2)   
    删除：delete from table1 where 范围   
    更新：update table1 set field1=value1 where 范围   
    查找：select \* from table1 where field1 like ’%value1%’   
    排序：select \* from table1 order by field1,field2 [desc]   
    desc：降序，asc：升序   
    总数：select count as totalcount from table1   
    求和：select sum(field1) as sumvalue from table1   
    平均：select avg(field1) as avgvalue from table1   
    最大：select max(field1) as maxvalue from table1   
    最小：select min(field1) as minvalue from table1
11. 分组:Group by:   
    一张表，一旦分组完成后，查询后只能得到组相关的信息。   
    组相关的信息：（统计信息） count,sum,max,min,avg 分组的标准)
12. 复制表select \* into b from a where 1<>1（仅用于SQlServer）
13. 子查询(表名1：a 表名2：b) select a,b,c from a where a IN (select d from b )
14. 显示文章、提交人和最后回复时间   
    select a.title,a.username,b.adddate from table a,(select max(adddate) adddate from table where table.title=a.title) b
15. 视图查询(表名1：a )   
    select \* from (SELECT a,b,c FROM a) T where t.a > 1;
16. between的用法,between限制查询数据范围时包括了边界值,not between不包括   
    select \* from table1 where time between time1 and time2   
    select a,b,c, from table1 where a not between 数值1 and 数值2
17. in 的使用方法   
    select \* from table1 where a [not] in (‘值1’,’值2’,’值4’,’值6’)
18. 两张关联表，删除主表中已经在副表中没有的信息   
    delete from table1 where not exists ( select \* from table2 where table1.field1=table2.field1 )
19. 日程安排提前五分钟提醒   
    SQL: select \* from 日程安排 where datediff(‘minute’,f开始时间,getdate())>5
20. 前10条记录   
    select top 10 \* form table1 where 范围
21. 选择在每一组b值相同的数据中对应的a最大的记录的所有信息(类似这样的用法可以用于论坛每月排行榜,每月热销产品分析,按科目成绩排名,等等.)   
    select a,b,c from tablename ta where a=(select max(a) from tablename tb where tb.b=ta.b)
22. 包括所有在 TableA 中但不在 TableB和TableC 中的行并消除所有重复行而派生出一个结果表   
    (select a from tableA ) except (select a from tableB) except (select a from tableC)
23. 随机取出10条数据  select top 10 \* from tablename order by newid()
24. **D**
25. **d**
26. **优化**
27. **维护数据库完整性和一致性**

尽可能使用约束：check、主外键、非空字段等，方便效率高；其次使用触发器。最后是自写业务逻辑

1. **连接池优点**

JDBC数据库连接池的必要性 没使用数据库连接池之前

1. 使用时连接数据库，使用后断开数据库。
2. 对于每一次数据库连接，使用完后都得断开。否则，如果程序出现异常而未能关闭，将会导致数据库系统中的内存泄漏，最终将导致重启数据库
3. 如果成千上万的用户同时使用数据库，频繁的进行数据库连接操作将占用很多的系统资源，严重的甚至会造成服务器的崩溃

新的资源分配手段对于多应用共享同一数据库的系统而言，可在应用层通过数据库连接池的配置实现某一应用最大可用数据库连接数的限制避免某一应用独占所有的数据库资源.

统一的连接管理，避免数据库连接泄露在较为完善的数据库连接池实现中，可根据预先的占用超时设定，强制回收被占用连接，从而避免了常规数据库连接操作中可能出现的资源泄露。

数据库连接池技术的优点

(1)资源重用：由于数据库连接得以重用，避免了频繁创建，释放连接引起的大量性能开销。在减少系统消耗的基础上，另一方面也增加了系统运行环境的平稳性。

(2)更快的系统反应速度:数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于连接池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接避免了数据库连接初始化和释放过程的时间开销，从而减少了系统的响应时间

1. **数据库优化方面的经验**
2. 用PreparedStatement 一般来说比Statement性能高,而且可以防止SQL注入攻击
3. 有外键约束会影响插入和删除性能，如果程序能够保证数据的完整性，那在设计数据库时就去掉外键。在大型电商中尽量使用单表
4. 表中允许适当冗余，利用空间换时间，提高查询效率
5. 用户名和密码单独从用户表中独立出来。 在登录业务中只需要查询用户名和密码。
6. 使用索引。

select尽量不用\*，即使查询全部字段也要尽量全部写出来

能用连接查询绝不用子查询，因为子查询的的底层就是连接查询

合理使用索引提升效率（索引的本质是提高效率的机制，本质上是一棵树）

为经常出现在where子句中的列创建索引

为经常出现在order by，distinct中的字段建立索引

不要经常在DML（检索或修改）的表上建立索引（容易造成索引不准， 解决方法为rebuild重新建立，任何 DML操作都是更新index，这是代价）

不要在小表上简历索引（索引简历和维护也是要消耗资源的，小表上建立 索引会造成索引的维护时间大于数据查询时间）

限制表上的索引数目，索引并不是越多越好（索引会占用表空间，还要额 外的维护）

删除很少被使用的，不合理的索引

1. **数据库数据量过大，数据库接近崩溃的时候怎么办？**

1、索引优化和SQL语句优化是必须的，避免模糊查询和非索引查询，删改操作根据聚集索引进行，删改操作太频繁的话还是需要考虑分表

2、看需求，如果需求不限制，那就分表

3、一般都是把历史数据定期转存其他表（一样的表名后加年月例如TABLE201205）归档。这样该表本年度的查询的压力也小点（90%查询量集中在本年度）,即使查询历史数据也不影响性能，强力推荐！

4结合你的业务去优化表结构。有时候可以考虑用空间去换时间。

1. C
2. D
3. d
4. **练习**
5. **行列转换**

CREATE TABLE Scores(

ID INT,

Student varchar(10),

Subject varchar(10),

Score INT

);

INSERT INTO Scores VALUES(2, '张三', '语文', 93);

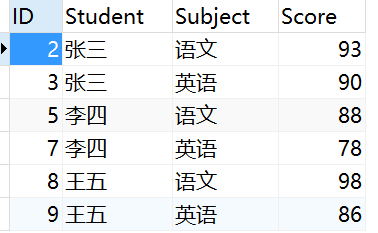
INSERT INTO Scores VALUES(3, '张三', '英语', 90);

INSERT INTO Scores VALUES(5, '李四', '语文', 88);

INSERT INTO Scores VALUES(7, '李四', '英语', 78);

INSERT INTO Scores VALUES(8, '王五', '语文', 98);

INSERT INTO Scores VALUES(9, '王五', '英语', 86);



SELECT student, max(yw) as '语文', max(yu) as '英语'

FROM(

SELECT student,

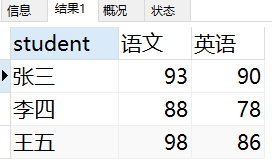
CASE subject WHEN '语文' THEN score ELSE 0 END as yw,

CASE subject WHEN '英语' THEN score ELSE 0 END as yu

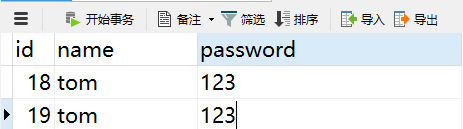
FROM Scores

) A

GROUP BY student



1. **删除重复数据**



DELETE FROM person WHERE id NOT IN (SELECT tab.id FROM

(SELECT MAX(id) AS id FROM person GROUP BY NAME ,PASSWORD

) AS tab )

这里面的第一个关键点是如何确定哪些数据是重复的呢？这里通过 group by name,password来确定。只要同时满足这两个条件，就可以归为相同一组数据

第二个关键点是通过max(id) 确定保留下的数据是id最大的。

第三点是 最内层的select语句的查询结果需要重启起名称为tab ，再从tab 中查出id .如果缺少这一步，语句会报错。

第四步是根据 id not in 删除除最大id外的所有数据

1. **省市县三级联动数据库是怎么设计的**

CREATE TABLE t\_area (

id int(11) NOT NULL auto\_increment,

area\_id varchar(50) default NULL,

area\_name varchar(60) default NULL,

area\_parentId varchar(6) default NULL,

primary key(id)

) ENGINE=MyISAM AUTO\_INCREMENT=3145 DEFAULT CHARSET=utf8;

INSERT INTO t\_area VALUES (1,'100','河北省','0');

INSERT INTO t\_area VALUES (2,'220999','石家庄市','100');

1. **Sql练习1**

Student(S#,Sname,Sage,Ssex) 学生表   
 Course(C#,Cname,T#) 课程表   
 SC(S#,C#,score) 成绩表   
 Teacher(T#,Tname) 教师表

问题：   
**1**、查询“**001**”课程比“**002**”课程成绩高的所有学生的学号；   
  select a.S# from (select s#,score from SC where C#='001') a,(select s#,score   
  from SC where C#='002') b   
  where a.score>b.score and a.s#=b.s#;   
**2**、查询平均成绩大于60分的同学的学号和平均成绩；   
    select S#,avg(score)   
    from sc   
    group by S# having avg(score) >**60**;   
**3**、查询所有同学的学号、姓名、选课数、总成绩；   
  select Student.S#,Student.Sname,count(SC.C#),sum(score)   
  from Student left Outer join SC on Student.S#=SC.S#   
  group by Student.S#,Sname   
**4**、查询姓“李”的老师的个数；   
  select count(distinct(Tname))   
  from Teacher   
  where Tname like '李%';   
**5**、查询没学过“叶平”老师课的同学的学号、姓名；

Student.S#,Student.Sname   
    from Student    
    where S# not in (select distinct( SC.S#) from SC,Course,Teacher where  SC.C#=Course.C# and Teacher.T#=Course.T# and Teacher.Tname='叶平');   
**6**、查询学过“**001**”并且也学过编号“**002**”课程的同学的学号、姓名；   
  select Student.S#,Student.Sname from Student,SC where Student.S#=SC.S# and SC.C#='001'and exists( Select \* from SC as SC\_2 where SC\_2.S#=SC.S# and SC\_2.C#='002');   
**7**、查询学过“叶平”老师所教的所有课的同学的学号、姓名；   
  select S#,Sname   
  from Student   
  where S# in (select S# from SC ,Course ,Teacher where SC.C#=Course.C# and Teacher.T#=Course.T# and Teacher.Tname='叶平' group by S# having count(SC.C#)=(select count(C#) from

Course,Teacher  where Teacher.T#=Course.T# and Tname='叶平'));   
**8**、查询课程编号“**002**”的成绩比课程编号“**001**”课程低的所有同学的学号、姓名；   
  Select S#,Sname from (select Student.S#,Student.Sname,score ,(select score from SC SC\_2 where SC\_2.S#=Student.S# and SC\_2.C#='002') score2   
  from Student,SC where Student.S#=SC.S# and C#='001') S\_2 where score2 <score;   
**9**、查询所有课程成绩小于60分的同学的学号、姓名；   
  select S#,Sname   
  from Student   
  where S# not in (select Student.S# from Student,SC where S.S#=SC.S# and score>**60**);   
**10**、查询没有学全所有课的同学的学号、姓名；   
    select Student.S#,Student.Sname   
    from Student,SC   
    where Student.S#=SC.S# group by  Student.S#,Student.Sname having count(C#) <(select count(C#) from Course);   
**11**、查询至少有一门课与学号为“**1001**”的同学所学相同的同学的学号和姓名；

select S#,Sname from Student,SC where Student.S#=SC.S# and C# in select C# from SC where S#='1001';   
**12**、查询至少学过学号为“**001**”同学所有一门课的其他同学学号和姓名；   
    select distinct SC.S#,Sname   
    from Student,SC   
    where Student.S#=SC.S# and C# in (select C# from SC where S#='001');   
**13**、把“SC”表中“叶平”老师教的课的成绩都更改为此课程的平均成绩；   
    update SC set score=(select avg(SC\_2.score)   
    from SC SC\_2   
    where SC\_2.C#=SC.C# ) from Course,Teacher where Course.C#=SC.C# and Course.T#=Teacher.T# and Teacher.Tname='叶平');   
**14**、查询和“**1002**”号的同学学习的课程完全相同的其他同学学号和姓名；   
    select S# from SC where C# in (select C# from SC where S#='1002')   
    group by

S# having count(\*)=(select count(\*) from SC where S#='1002');   
**15**、删除学习“叶平”老师课的SC表记录；   
    Delect SC   
    from course ,Teacher    
    where Course.C#=SC.C# and Course.T#= Teacher.T# and Tname='叶平';   
**16**、向SC表中插入一些记录，这些记录要求符合以下条件：没有上过编号“**003**”课程的同学学号、**2**、   
    号课的平均成绩；   
    Insert SC select S#,'002',(Select avg(score)   
    from SC where C#='002') from Student where S# not in (Select S# from SC where C#='002');   
**17**、按平均成绩从高到低显示所有学生的“数据库”、“企业管理”、“英语”三门的课程成绩，按如下形式显示： 学生ID,,数据库,企业管理,英语,有效课程数,有效平均分   
    SELECT S# as 学生ID   
        ,(SELECT score FROM SC WHERE SC.S#=t.S# AND C#='004') AS 数据库

,(SELECT score FROM SC WHERE SC.S#=t.S# AND C#='001') AS 企业管理   
        ,(SELECT score FROM SC WHERE SC.S#=t.S# AND C#='006') AS 英语   
        ,COUNT(\*) AS 有效课程数, AVG(t.score) AS 平均成绩   
    FROM SC AS t   
    GROUP BY S#   
    ORDER BY avg(t.score)    
**18**、查询各科成绩最高和最低的分：以如下形式显示：课程ID，最高分，最低分   
    SELECT L.C# As 课程ID,L.score AS 最高分,R.score AS 最低分   
    FROM SC L ,SC AS R   
    WHERE L.C# = R.C# and   
        L.score = (SELECT MAX(IL.score)   
                      FROM SC AS IL,Student AS IM   
                      WHERE L.C# = IL.C# and IM.S#=IL.S#   
                      GROUP BY IL.C#)   
        AND   
        R.Score

= (SELECT MIN(IR.score)   
                      FROM SC AS IR   
                      WHERE R.C# = IR.C#   
                  GROUP BY IR.C#   
                    );   
**19**、按各科平均成绩从低到高和及格率的百分数从高到低顺序   
    SELECT t.C# AS 课程号,max(course.Cname)AS 课程名,isnull(AVG(score),**0**) AS 平均成绩   
        ,**100** \* SUM(CASE WHEN  isnull(score,**0**)>=**60** THEN **1** ELSE **0** END)/COUNT(\*) AS 及格百分数   
    FROM SC T,Course   
    where t.C#=course.C#   
    GROUP BY t.C#   
    ORDER BY **100** \* SUM(CASE WHEN  isnull(score,**0**)>=**60** THEN **1** ELSE

**0** END)/COUNT(\*) DESC   
**20**、查询如下课程平均成绩和及格率的百分数(用"1行"显示): 企业管理（**001**），马克思（**002**），OO&UML （**003**），数据库（**004**）   
    SELECT SUM(CASE WHEN C# ='001' THEN score ELSE **0** END)/SUM(CASE C# WHEN '001' THEN **1** ELSE **0** END) AS 企业管理平均分   
        ,**100** \* SUM(CASE WHEN C# = '001' AND score >= **60** THEN **1** ELSE **0** END)/SUM(CASE WHEN C# = '001' THEN **1** ELSE **0** END

) AS 企业管理及格百分数   
        ,SUM(CASE WHEN C# = '002' THEN score ELSE **0** END)/SUM(CASE C# WHEN '002' THEN **1** ELSE **0** END) AS 马克思平均分   
        ,**100** \* SUM(CASE WHEN C# = '002' AND score >= **60** THEN **1** ELSE **0** END)/SUM(CASE WHEN C# = '002' THEN **1** ELSE **0** END) AS 马克思及格百分数   
        ,SUM(CASE WHEN C# = '003' THEN score ELSE **0** END)/

SUM(CASE C# WHEN '003' THEN **1** ELSE **0** END) AS UML平均分   
        ,**100** \* SUM(CASE WHEN C# = '003' AND score >= **60** THEN **1** ELSE **0** END)/SUM(CASE WHEN C# = '003' THEN **1** ELSE **0** END) AS UML及格百分数   
        ,SUM(CASE WHEN C# = '004' THEN score ELSE **0** END)/SUM(CASE C# WHEN '004' THEN **1** ELSE **0** END) AS 数据库平均分   
        ,**100**

\* SUM(CASE WHEN C# = '004' AND score >= **60** THEN **1** ELSE **0** END)/SUM(CASE WHEN C# = '004' THEN **1** ELSE **0** END) AS 数据库及格百分数   
  FROM SC   
**21**、查询不同老师所教不同课程平均分从高到低显示   
  SELECT max(Z.T#) AS 教师ID,MAX(Z.Tname) AS 教师姓名,C.C# AS 课程ＩＤ,MAX(C.Cname) AS 课程名称,AVG(Score) AS 平均成绩   
    FROM SC AS T,Course AS C ,Teacher AS Z   
    where T.C#=C.C# and C.T#=Z.T#   
  GROUP BY C.C#   
  ORDER BY AVG(Score) DESC   
**22**、查询如下课程成绩第 **3** 名到第 **6** 名的学生成绩单：企业管理（**001**），马克思（**002**），UML （**003**），数据库（**004**）   
    [学生ID],[学生姓名],企业管理,马克思,UML,数据库,平均成绩   
    SELECT  DISTINCT top **3**   
      SC.S# As 学生学号,   
        Student.Sname AS 学生姓名 ,   
      T1.score AS 企业管理,   
      T2.score AS 马克思,   
      T3.score AS UML,   
      T4.score AS 数据库,   
      ISNULL(T1.score,**0**) + ISNULL(T2.score,**0**) + ISNULL(T3.score,**0**) + ISNULL(T4.score,**0**) as 总分   
      FROM Student,SC  LEFT JOIN SC AS T1   
                      ON SC.S# = T1.S# AND T1.C# = '001'   
            LEFT JOIN SC AS T2   
                      ON SC.S# = T2.S# AND T2.C# = '002'   
            LEFT JOIN SC AS T3   
                      ON SC.S# =T3.S# AND T3.C# = '003'   
            LEFT JOIN SC AS T4   
                      ON SC.S# = T4.S# AND T4.C# = '004'   
      WHERE student.S#=SC.S# and   
      ISNULL(T1.score,**0**) + ISNULL(T2.score,**0**) + ISNULL(T3.score,**0**) + ISNULL(T4.score,**0**)   
      NOT IN   
      (SELECT   
            DISTINCT   
            TOP **15** WITH TIES   
            ISNULL(T1.score,**0**) + ISNULL(T2.score,**0**) + ISNULL(T3.score,**0**) + ISNULL(T4.score,**0**)   
      FROM sc   
            LEFT JOIN sc AS T1   
                      ON sc.S# = T1.S# AND T1.C#= 'k1'   
            LEFT JOIN sc AS T2   
                      ON sc.S# = T2.S# AND T2.C# = 'k2'   
            LEFT JOIN sc AS T3   
                      ON sc.S# = T3.S# AND T3.C# = 'k3'   
            LEFT JOIN sc AS T4   
                      ON sc.S# = T4.S# AND T4.C# = 'k4'   
      ORDER BY ISNULL(T1.score,**0**) + ISNULL(T2.score,**0**) + ISNULL(T3.score,**0**) + ISNULL(T4.score,**0**) DESC);   
  
**23**、统计列印各科成绩,各分数段人数:课程ID,课程名称,[100-85],[85-70],[70-60],[ <60]   
    SELECT SC.C# as 课程ID, Cname

as 课程名称   
        ,SUM(CASE WHEN score BETWEEN **85** AND **100** THEN **1** ELSE **0** END) AS [100 - 85]   
        ,SUM(CASE WHEN score BETWEEN **70** AND **85** THEN **1** ELSE **0** END) AS [85 - 70]   
        ,SUM(CASE WHEN score BETWEEN **60** AND **70** THEN **1** ELSE **0** END) AS [70 - 60]   
        ,SUM(CASE WHEN score < **60** THEN **1** ELSE **0** END) AS [60 -]   
    FROM SC,Coursewhere SC.C#=Course.C#   
    GROUP BY SC.C#,Cname;   
  
**24**、查询学生平均成绩及其名次   
      SELECT **1**+(SELECT COUNT( distinct 平均成绩)   
              FROM (SELECT S#,AVG(score) AS 平均成绩   
                      FROM SC   
                  GROUP BY S#   
                  ) AS T1   
            WHERE 平均成绩 > T2.平均成绩) as 名次,   
      S# as 学生学号,平均成绩   
    FROM (SELECT S#,AVG(score) 平均成绩   
            FROM SC   
        GROUP BY S#   
        ) AS T2   
    ORDER BY 平均成绩 desc;   
    
**25**、查询各科成绩前三名的记录:(不考虑成绩并列情况)   
      SELECT t1.S# as 学生ID,t1.C# as 课程ID,Score as 分数   
      FROM SC t1   
      WHERE score IN (SELECT TOP **3** score   
              FROM SC   
              WHERE t1.C#= C#     ORDER BY score DESC   
              )   
      ORDER BY t1.C#;   
**26**、查询每门课程被选修的学生数   
  select c#,count(S#) from sc group by C#;   
**27**、查询出只选修了一门课程的全部学生的学号和姓名   
  select SC.S#,Student.Sname,count(C#) AS 选课数   
  from SC ,Student   
  where SC.S#=Student.S# group by SC.S# ,Student.Sname having count(C#)=**1**;   
**28**、查询男生、女生人数   
    Select count(Ssex) as 男生人数 from Student group by Ssex having Ssex='男';   
    Select count(Ssex) as 女生人数 from Student group by Ssex having Ssex='女'；   
**29**、查询姓“张”的学生名单   
    SELECT Sname FROM Student WHERE Sname like '张%';   
**30**、查询同名同性学生名单，并统计同名人数   
  select Sname,count(\*) from Studentgroup by Sname having  count(\*)>**1**;;   
**31**、1981年出生的学生名单(注：Student表中Sage列的类型是datetime)   
    select Sname,  CONVERT(char (**11**),DATEPART(year,Sage)) as age   
    from student   
    where  CONVERT(char(**11**),DATEPART(year,Sage))='1981';   
**32**、查询每门课程的平均成绩，结果按平均成绩升序排列，平均成绩相同时，按课程号降序排列   
    Select C#,Avg(score) from SC group by C# order by Avg(score),C# DESC ;   
**33**、查询平均成绩大于85的所有学生的学号、姓名和平均成绩   
    select Sname,SC.S# ,avg(score)   
    from Student,SC   
    where Student.S#=SC.S# group by SC.S#,Sname having    avg(score)>**85**;   
**34**、查询课程名称为“数据库”，且分数低于60的学生姓名和分数   
    Select Sname,isnull(score,**0**)   
    from Student,SC,Course   
    where SC.S#=Student.S# and SC.C#=Course.C#and  Course.Cname='数据库'and score <**60**;   
**35**、查询所有学生的选课情况；   
    SELECT SC.S#,SC.C#,Sname,Cname   
    FROM SC,Student,Course   
    where SC.S#=Student.S# and SC.C#=Course.C# ;   
**36**、查询任何一门课程成绩在70分以上的姓名、课程名称和分数；   
    SELECT  distinct student.S#,student.Sname,SC.C#,SC.score   
    FROM student,Sc   
    WHERE SC.score>=**70** AND SC.S#=student.S#;   
**37**、查询不及格的课程，并按课程号从大到小排列   
    select c# from sc where scor e <**60** order by C# ;   
**38**、查询课程编号为003且课程成绩在80分以上的学生的学号和姓名；   
    select SC.S#,Student.Sname from SC,Student where SC.S#=Student.S# and Score>**80** and C#='003';   
**39**、求选了课程的学生人数   
    select count(\*) from sc;   
**40**、查询选修“叶平”老师所授课程的学生中，成绩最高的学生姓名及其成绩   
    select Student.Sname,score   
    from Student,SC,Course C,Teacher   
    where Student.S#=SC.S# and SC.C#=C.C# and

C.T#=Teacher.T# and Teacher.Tname='叶平' and SC.score=(select max(score)from SC where C#=C.C# );   
**41**、查询各个课程及相应的选修人数   
    select count(\*) from sc group by C#;   
**42**、查询不同课程成绩相同的学生的学号、课程号、学生成绩   
  select distinct  A.S#,B.score from SC A  ,SC B where A.Score=B.Score and A.C# <>B.C# ;   
**43**、查询每门功成绩最好的前两名   
    SELECT t1.S# as 学生ID,t1.C# as 课程ID,Score as 分数   
      FROM SC t1   
      WHERE score IN (SELECT TOP **2** score   
              FROM SC   
              WHERE t1.C#= C#   
            ORDER BY score DESC   
              )   
      ORDER BY t1.C#;   
**44**、统计每门课程的学生选修人数（超过10人的课程才统计）。要求输出课程号和选修人数，查询结果按人数降序排列，查询结果按人数降序排列，若人数相同，按课程号升序排列    
    select  C# as 课程号,count(\*) as 人数

    from  sc    
    group  by  C#   
    order  by  count(\*) desc,c#    
**45**、检索至少选修两门课程的学生学号   
    select  S#    
    from  sc    
    group  by  s#   
    having  count(\*)  >  =  **2**   
**46**、查询全部学生都选修的课程的课程号和课程名   
    select  C#,Cname    
    from  Course    
    where  C#  in  (select  c#  from  sc group  by  c#)    
**47**、查询没学过“叶平”老师讲授的任一门课程的学生姓名   
    select Sname from Student where S# not in (select S# from Course,Teacher,SC where Course.T#=Teacher.T# and SC.C#=course.C# and Tname='叶平');   
**48**、查询两门以上不及格课程的同学的学号及其平均成绩   
    select S#,avg(isnull(score,**0**)) from SC where S# in (select S# from SC where

score <**60** group by S# having count(\*)>**2**)group by S#;   
**49**、检索“**004**”课程分数小于60，按分数降序排列的同学学号   
    select S# from SC where C#='004'and score <**60** order by score desc;   
**50**、删除“**002**”同学的“**001**”课程的成绩   
delete from Sc where S#='001'and C#='001';

1. Dd
2. D
3. d
4. ddd
5. **算法**
6. **冒泡排序**

/\*

冒泡排序。

比较方式：相邻两个元素进行比较。如果满足条件就进行位置置换。

原理：内循环结束一次，最值出现在尾角标位置。\*/

**public** **static** **void** bubbleSort(**int**[] arr)

{

**for**(**int** x=0; x<arr.length-1; x++)

{

**for**(**int** y=0; y<arr.length-x-1; y++)//-x:让每次参与比较的元减。

//-1:避免角标越界。

{

**if**(arr[y]>arr[y+1])

{

**int** temp = arr[y];

arr[y] = arr[y+1];

arr[y+1] = temp;

}

}

}

}

1. **折半查找(二分法)**

/\*

为了提高查找效率，可使用折半查找的方式，注意：这种查找只对有序的数组有效。

这种方式也成为二分查找法。

\*/

**public** **static** **int** halfSeach(**int**[] arr,**int** key)

{

**int** min,mid,max;

min = 0;

max = arr.length-1;

mid = (max+min)/2;

**while**(arr[mid]!=key)

{

**if**(key>arr[mid])

min = mid + 1;

**else** **if**(key<arr[mid])

max = mid - 1;

**if**(min>max)

**return** -1;

mid = (max+min)/2;

}

**return** mid;

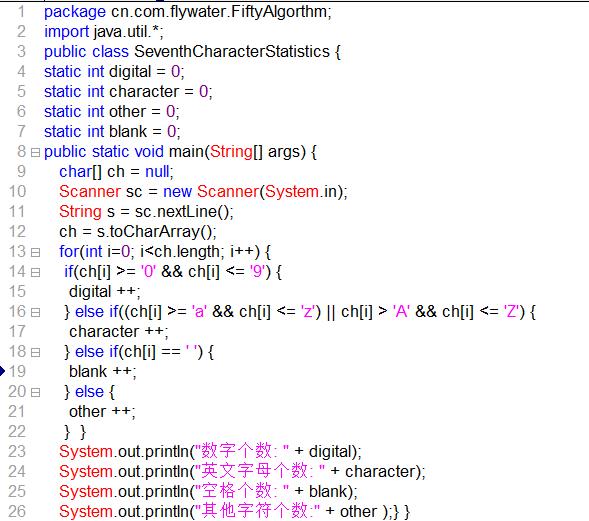
}

1. **递归算法题**

**第1个人10，第2个比第1个人大2岁，依次递推，请用递归方式计算出第8个人多大？**

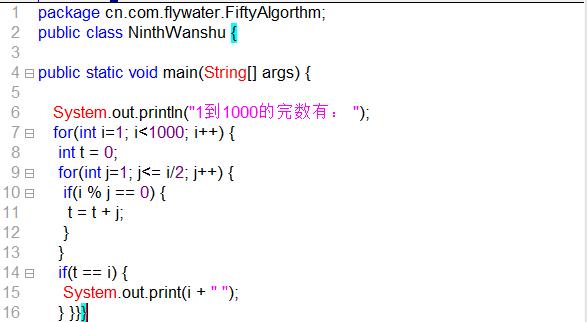
|  |
| --- |
| **package** dao;  **public** **class** A {  **public** **static** **void** main(String[] args) {  System.***out***.println(*computeAge*(8));  }  **public** **static** **int** computeAge(**int** n) {  **if** (n == 1)  **return** 10;  **return** *computeAge*(n - 1) + 2;  }  **public** **static** **void** toBinary(**int** n, StringBuffer result) {  **if** (n / 2 != 0)  *toBinary*(n / 2, result);  result.append(n % 2);  }  } |

1. **输入一行字符，请统计出这中间的空格、英文字母、数字以及其他字符的个数**

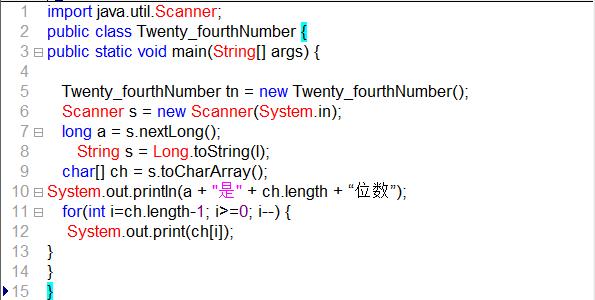


1. **完数**

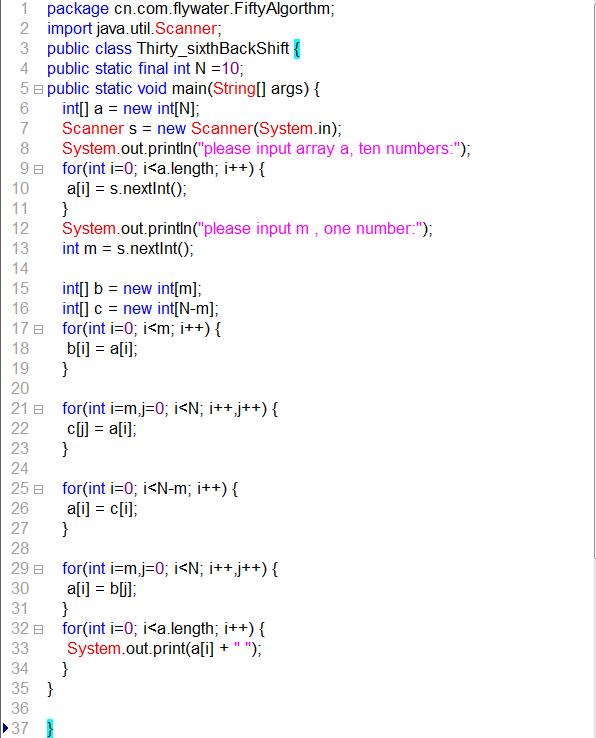
一个数如果恰好等于它的因子之和，这个数就称为 "完数 "。例如6=1＋2＋3.编程 找出1000以内的所有完数。



1. **给一个不多于5位的正整数，要求：一、求它是几位数，二、逆序打印出各位数字。**



1. **有n个整数，使其前面各数顺序向后移m个位置，最后m个数变成最前面的m个数**



1. **如何把一段逗号分割的字符串转换成一个数组?**

1)用正则表达式，代码大概为：String [] result = orgStr.split(“,”, -1);

2)用 StingTokenizer ,代码为：

StringTokenizer tokener = new StringTokenizer(s, ",");

String[] result = new String[tokener.countTokens()];

Integer i = 0;

while (tokener.hasMoreTokens()) {

result[i++] = tokener.nextToken();

}

1. D
2. D
3. d
4. Dd
5. D
6. D
7. D
8. D