## （注：答案写在答题纸上，标注以下内容 做题人 做题时间 结束时间 答题时间 1 ~1.5小时）

## 概念题

1. **main方法是Java Application程序执行的入口点，关于main方法的方法头以下哪项是合法的（C）**

　　A public static void main （）　B public static void main （String args）

　　C public static int main （String [] arg）　D public void main （String args[]）

1. **设float x=1，y=2，z=3，则表达式y+=z--/++x的值是（B）**

　　A 3.5 B 3

　　C 4 D 5

1. 说出 Servlet 的生命周期，并说出 Servlet 和 CGI 的区别。

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

与 cgi 的区别在于 servlet 处于服务器进程中，它通过多线程方式运行其 service 方法，一个 实例可以服务于多个请求，并且其实例一般不会销毁，而 CGI 对每个请求都产生新的进程， 服务完成后就销毁，所以效率上低于 servlet。

1. **以下数值相等否？不相等说明原因？**

Integer.valueOf("12");

Integer.parseInt("12");

1. **表达式12&9的运算结果是？**

1110 & 1011 = 1010 8

1. **执行以下代码int x,a=4,b=6,c=8 ; x=++a+b+++c++; 结果是：**

a:5,b:7,c:9,x:19

1. **对于1.0/0.0 的结果是：**

Double.POSITIVE\_INFINITY

1. switch语句能否作用在byte上,能否作用在long上,能否作用在String上

switch可作用于char byte short int

switch可作用于char byte short int对应的包装类

switch不可作用于long double float boolean，包括他们的包装类

switch中可以是字符串类型,String(jdk1.7之后才可以作用在string上)

switch中可以是枚举类型

1. 方法重载Overload和方法的重写Overriding的区别。Overload的方法是否可以改变返回值的类型？

方法的重写Overriding和重载Overloading是Java多态性的不同表现。重写Overriding是父类与子类之间多态性的一种表现，重载Overloading是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写（Overriding）。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽”了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载（Overloading）。 Overloaded的方法是可以改变返回值的类型。

1. 谈谈HashMap和Hashtable的区别（6分）

答：

1.Hashtable是Dictionary的子类，HashMap是Map接口的一个实现类；

2.Hashtable中的方法是同步的，而HashMap中的方法在缺省情况下是非同步的。即是说，在多线程应用程序中，不用专门的操作就安全地可以使用Hashtable了；而对于HashMap，则需要额外的同步机制。但HashMap的同步问题可通过Collections的一个静态方法得到解决：

Map Collections.synchronizedMap(Map m)

这个方法返回一个同步的Map，这个Map封装了底层的HashMap的所有方法，使得底层的HashMap即使是在多线程的环境中也是安全的。

3. 在HashMap中，null可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为null。当get()方法返回null值时，即可以表示 HashMap中没有该键，也可以表示该键所对应的值为null。

因此，在HashMap中不能由get()方法来判断HashMap中是否存在某个键，而应该用containsKey()方法来判断。

4.其底层的实现机制不同，hashmap的访问速度要快于hashtable，因为它不需要进行同步检验，建议在非多线程环境中使用hashmap代替hashtable .

1. String是最基本的数据类型吗？基本数据类型还有？

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

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类

1. **以下表达式中返回true的有？（ADEF） 为何？**

String a = “Hello”; String b= “Hello”;

String c= new String(“Hello”); String d= new String(“Hello”);

A. a==b B. a==c C. c==d D. a.equals(b) E. a.equals(c) F.c.equals(d)

1. int 和 Integer 有什么区别? 原始类型封装类都有?

Java 提供两种不同的类型：引用类型和原始类型（或内置类型）。Int是java的原始数据类型，Integer是java为int提供的封装类。Java为每个原始类型提供了封装类。Interger 是JAVA提供的封装类。  
4 i! }/ ~5 G, Y4 J6 T' y6 b7 \_http://www.itshaik.com原始类型封装类：boolean Boolean    char Character    byte Byte    short Short int Integer                  long Long             float Float    double Double

1. **解释String s=”abc”; s+=”word”;代码的过程，并说明String 和StringBuffer的区别？StringBuilder呢？他们各自的执行效率？**JAVA平台提供了两个类：String和StringBuffer，它们可以储存和操作字符串，即包含多个字符的字符数据。这个String类提供了数值不可改变的字符串。而这个StringBuffer类提供的字符串进行修改。当你知道字符数据要改变的时候你就可以使用StringBuffer。典型地，你可以使用StringBuffers来动态构造字符数据。
2. 说出ArrayList,Vector, LinkedList的存储性能和特性？

ArrayList和Vector都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector由于使用了synchronized方法（线程安全），通常性能上较ArrayList差，而LinkedList使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

1. final, finally, finalize的区别

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。

1. **abstract class和interface有什么区别? 接口是否可继承接口? 抽象类是否可实现(implements)接口? 抽象类是否可继承实体类(concrete class)?**

声明方法的存在而不去实现它的类被叫做抽象类（abstract class），它用于要创建一个体现某些基本行为的类，并为该类声明方法，但不能在该类中实现该类的情况。不能创建abstract 类的实例。然而可以创建一个变量，其类型是一个抽象类，并让它指向具体子类的一个实例。不能有抽象构造函数或抽象静态方法。Abstract 类的子类为它们父类中的所有抽象方法提供实现，否则它们也是抽象类。取而代之，在子类中实现该方法。知道其行为的其它类可以在类中实现这些方法。

接口（interface）是抽象类的变体。在接口中，所有方法都是抽象的。多继承性可通过实现这样的接口而获得。接口中的所有方法都是抽象的，没有一个有程序体。接口只可以定义static final成员变量。接口的实现与子类相似，除了该实现类不能从接口定义中继承行为。当类实现特殊接口时，它定义（即将程序体给予）所有这种接口的方法。然后，它可以在实现了该接口的类的任何对象上调用接口的方法。由于有抽象类，它允许使用接口名作为引用变量的类型。通常的动态联编将生效。引用可以转换到接口类型或从接口类型转换，instanceof 运算符可以用来决定某对象的类是否实现了接口。

接口可以继承接口。抽象类可以实现(implements)接口，抽象类可继承实体类

1. 给我几个你最常见到的runtime exception

ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, ClassCastException, CMMException, ConcurrentModificationException, DOMException, EmptyStackException, IllegalArgumentException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, ImagingOpException, IndexOutOfBoundsException, MissingResourceException, NegativeArraySizeException, NoSuchElementException, NullPointerException, ProfileDataException, ProviderException, RasterFormatException, SecurityException, SystemException, UndeclaredThrowableException, UnmodifiableSetException, UnsupportedOperationException

1. new 一个对象的过程和 clone 一个对象的过程区别？clone 对象的使用？深拷贝和浅拷贝

new 操作符的本意是分配内存。程序执行到 new 操作符时，首先去看 new 操作符后面的类型，因为知道了类型， 才能知道要分配多大的内存空间。分配完内存之后，再调用构造函数，填充对象的各个域，这一步叫做对象的初始化， 构造方法返回后，一个对象创建完毕，可以把他的引用(地址)发布到外部，在外部就可以使用这个引用操纵这个对象。

clone 在第一步是和 new 相似的，都是分配内存，调用 clone 方法时，分配的内存和原对象(即调用 clone 方法 的对象)相同，然后再使用原对象中对应的各个域，填充新对象的域，填充完成之后，clone 方法返回，一个新的相同 的对象被创建，同样可以把这个新对象的引用发布到外部。

clone 方法执行的是浅拷贝， 在编写程序时要注意这个细节。

1. 举例三个元注解并解释用途。解释一下springBoot中注解：@SpringBootApplication

@Retention 定义了该Annotation被保留的时间长短

@Target 所修饰的对象范围，取值(ElementType)有

**CONSTRUCTOR:用于描述构造器**

**FIELD:用于描述域**

**LOCAL\_VARIABLE:用于描述局部变量**

**METHOD:用于描述方法**

**PACKAGE:用于描述包**

**PARAMETER:用于描述参数**

**TYPE:用于描述类、接口(包括注解类型) 或enum声明**

@Documented 将此注解包含在 javadoc 中 ，它代表着此注解会被javadoc工具提取成文档。

@Inherited 说明子类可以继承父类中的该注解

@SpringBootApplication

@Target、@Retention、@Documented、@Inherited、@SpringBootConfiguration、@EnableAutoConfiguration、@ComponentScan

1. 描述一下JVM加载class文件的原理机制?

JVM中类的装载是由ClassLoader和它的子类来实现的,Java ClassLoader 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的类。

Class Loader的类结构层次:

*1.启动类加载器(Bootstrap Class Loader)*:用C++实现的类加载器

*2.拓展类加载器(Extension Class Loader)*:负责加载\lib\ext目录中的

*3.应用程序加载器(App Class Loader)*:由于这个类加载器是Class Loader的getSystemClassLoder()方法的返回值，所以又称*系统加载器* ,它负责*加载classPath*路径上的指定的类库

1. **char型变量中能不能存贮一个中文汉字?为什么?**

能够定义成为一个中文的，因为java中以unicode编码，一个char占2个字节（16比特）

，所以放一个中文是没问题的

1. **j2ee常用的设计模式？说明工厂模式。**

Java中的23种设计模式：

Factory（工厂模式）， Builder（建造模式）， Factory Method（工厂方法模式），

Prototype（原始模型模式），Singleton（单例模式）， Facade（门面模式），

Adapter（适配器模式）， Bridge（桥梁模式）， Composite（合成模式），

Decorator（装饰模式）， Flyweight（享元模式）， Proxy（代理模式），

Command（命令模式）， Interpreter（解释器模式）， Visitor（访问者模式），

Iterator（迭代子模式）， Mediator（调停者模式）， Memento（备忘录模式），

Observer（观察者模式）， State（状态模式）， Strategy（策略模式），

Template Method（模板方法模式）， Chain Of Responsibleity（责任链模式）

1. **内部类有几种？分别是？给出相应的代码**

成员内部类、局部内部类、匿名内部类、静态内部类

public class OuterClass {

private String sex;

public static String name = "chenssy";

/\*\*

\*静态内部类

\*/

static class InnerClass1{

/\* 在静态内部类中可以存在静态成员 \*/

public static String \_name1 = "chenssy\_static";

public void display(){

/\*

\* 静态内部类只能访问外围类的静态成员变量和方法

\* 不能访问外围类的非静态成员变量和方法

\*/

System.out.println("OutClass name :" + name);

}

}

/\*\*

\* 非静态内部类

\*/

class InnerClass2{

/\* 非静态内部类中不能存在静态成员 \*/

public String \_name2 = "chenssy\_inner";

/\* 非静态内部类中可以调用外围类的任何成员,不管是静态的还是非静态的 \*/

public void display(){

System.out.println("OuterClass name：" + name);

}

}

/\*\*

\* @desc 外围类方法

\* @author chenssy

\* @data 2013-10-25

\* @return void

\*/

public void display(){

/\* 外围类访问静态内部类：内部类. \*/

System.out.println(InnerClass1.\_name1);

/\* 静态内部类 可以直接创建实例不需要依赖于外围类 \*/

new InnerClass1().display();

/\* 非静态内部的创建需要依赖于外围类 \*/

OuterClass.InnerClass2 inner2 = new OuterClass().new InnerClass2();

/\* 方位非静态内部类的成员需要使用非静态内部类的实例 \*/

System.out.println(inner2.\_name2);

inner2.display();

}

public static void main(String[] args) {

OuterClass outer = new OuterClass();

outer.display();

}

}

----------------

Output:

chenssy\_static

OutClass name :chenssy

chenssy\_inner

OuterClass name：chenssy

1. 可以不通过构造方法创建对象？如果可以，有哪些方法？

通过clone 的方式

通过反射机制

### 26）****Collection 和 Collections 的区别****

Collection 和 Collections 都是位于 java.util 包下的类

Collection 是集合类的父类，它是一个顶级接口，大部分抽象类比如说 AbstractList、AbstractSet 都继承了 Collection 类，Collection 类只定义一节标准方法比如说 add、remove、set、equals 等，具体的方法由抽象类或者实现类去实现。

Collections 是集合类的工具类，Collections 提供了一些工具类的基本使用

sort 方法，对当前集合进行排序, 实现 Comparable 接口的类，只能使用一种排序方案，这种方案叫做自然比较

比如实现线程安全的容器 Collections.synchronizedList、 Collections.synchronizedMap 等

reverse 反转，使用 reverse 方法可以根据元素的自然顺序 对指定列表按降序进行排序。

fill，使用指定元素替换指定列表中的所有元素。

有很多用法，读者可以翻阅 Collections 的源码查看，Collections 不能进行实例化，所以 Collections 中的方法都是由 Collections.方法 直接调用。

### 27）ArrayList、LinkedList 和 Vector 的区别

ArrayList、LinkedList、Vector 都是位于 java.util 包下的工具类，它们都实现了 List 接口。

ArrayList 的底层是动态数组，它是基于数组的特性而演变出来的，所以ArrayList 遍历访问非常快，但是增删比较慢，因为会涉及到数组的拷贝。ArrayList 是一个非线程安全的容器，在并发场景下会造成问题，如果想使用线程安全的容器的话，推荐使用 Collections.synchronizedList；ArrayList 在扩容时会增加 50% 的容量。

LinkedList 的底层是双向链表，所以 LinkedList 的增加和删除非常快，只需把元素删除，把各自的指针指向新的元素即可。但是 LinkedList 遍历比较慢，因为只有每次访问一个元素才能知道下一个元素的值。LinkedList 也是一个非线程安全的容器，推荐使用 Collections.synchronizedList

* Vector 向量是最早出现的集合容器，Vector 是一个线程安全的容器，它的每个方法都粗暴的加上了 synchronized 锁，所以它的增删、遍历效率都很低。Vector 在扩容时，它的容量会增加一倍。

### 27）String、StringBuilder 和 StringBuffer 有什么区别

String 特指的是 Java 中的字符串，String 类位于 java.lang 包下，String 类是由 final 修饰的，String 字符串一旦创建就不能被修改，任何对 String 进行修改的操作都相当于重新创建了一个字符串。String 字符串的底层使用 StringBuilder 来实现的

StringBuilder 位于 java.util 包下，StringBuilder 是一非线程安全的容器，StringBuilder 的 append 方法常用于字符串拼接，它的拼接效率要比 String 中 + 号的拼接效率高。StringBuilder 一般不用于并发环境

StringBuffer 位于 java.util 包下，StringBuffer 是一个线程安全的容器，多线程场景下一般使用 StringBuffer 用作字符串的拼接

StringBuilder 和 StringBuffer 都是继承于AbstractStringBuilder 类，AbstractStringBuilder 类实现了 StringBuffer 和 StringBuilder 的常规操作

### 28）String、StringBuilder 和 StringBuffer 有什么区别

String 特指的是 Java 中的字符串，String 类位于 java.lang 包下，String 类是由 final 修饰的，String 字符串一旦创建就不能被修改，任何对 String 进行修改的操作都相当于重新创建了一个字符串。String 字符串的底层使用 StringBuilder 来实现的

StringBuilder 位于 java.util 包下，StringBuilder 是一非线程安全的容器，StringBuilder 的 append 方法常用于字符串拼接，它的拼接效率要比 String 中 + 号的拼接效率高。StringBuilder 一般不用于并发环境

StringBuffer 位于 java.util 包下，StringBuffer 是一个线程安全的容器，多线程场景下一般使用 StringBuffer 用作字符串的拼接

StringBuilder 和 StringBuffer 都是继承于AbstractStringBuilder 类，AbstractStringBuilder 类实现了 StringBuffer 和 StringBuilder 的常规操作

**29）强引用、若引用、虚引用和幻象引用的区别**

我们说的不同的引用类型其实都是逻辑上的，而对于虚拟机来说，主要体现的是对象的不同的可达性(reachable) 状态和对垃圾收集(garbage collector)的影响

图片包含 游戏机, 钟表, 画, 标志

描述已自动生成

对象被创建并初始化，对象在运行时被使用，然后离开对象的作用域，对象会变成不可达并会被垃圾收集器回收。图中用红色标明的区域表示对象处于强可达阶段。

JDK1.2 介绍了 java.lang.ref 包，对象的生命周期有四个阶段：􏲧强可达􏰛(Strongly Reachable􏰜)、软可达(Soft Reachable􏰜)、弱可达(Weak Reachable􏰜)、 幻象可达(Phantom Reachable􏰜)

如果只讨论符合垃圾回收条件的对象，那么只有三种：软可达、弱可达和幻象可达。

* 软可达：软可达就是􏱬我们只能通过软引用􏳂才能访问的状态，软可达的对象是由 SoftReference 引用的对象，并且没有强引用的对象。软引用是用来描述一些还有用但是非必须的对象。垃圾收集器会尽可能长时间的保留软引用的对象，但是会在发生 OutOfMemoryError 之前，回收软引用的对象。如果回收完软引用的对象，内存还是不够分配的话，就会直接抛出 OutOfMemoryError。
* 弱可达：弱可达的对象是 WeakReference 引用的对象。垃圾收集器可以随时收集弱引用的对象，不会尝试保留软引用的对象。
* 幻象可达：幻象可达是由 PhantomReference 引用的对象，幻象可达就是没有强、软、弱引用进行关联，并且已经被 finalize 过了，只有幻象引用指向这个对象的时候。

除此之外，还有强可达和不可达的两种可达性判断条件

* 强可达：就是一个对象刚被创建、初始化、使用中的对象都是处于强可达的状态
* 不可达(unreachable)：处于不可达的对象就意味着对象可以被清除了

图片包含 游戏机, 文字

描述已自动生成

判断可达性条件，也是 JVM 垃圾收集器决定如何处理对象的一部分考虑因素。

所有的对象可达性引用都是 java.lang.ref.Reference 的子类，它里面有一个get() 方法，返回引用对象。如果已通过程序或垃圾收集器清除了此引用对象，则此方法返回 null 。也就是说，除了幻象引用外，软引用和弱引用都是可以得到对象的。而且这些对象可以人为拯救，变为强引用，例如把 this 关键字赋值给对象，只要重新和引用链上的任意一个对象建立关联即可

### ****30）静态绑定和动态绑定的区别****

一个Java 程序要经过编写、编译、运行三个步骤，其中编写代码不在我们讨论的范围之内，那么我们的重点自然就放在了编译 和 运行这两个阶段，由于编译和运行阶段过程相当繁琐，下面就我的理解来进行解释：

Java 程序从源文件创建到程序运行要经过两大步骤：

1、编译时期是由编译器将源文件编译成字节码的过程

2、字节码文件由Java虚拟机解释执行

绑定

绑定就是一个方法的调用与调用这个方法的类连接在一起的过程被称为绑定。

绑定主要分为两种：

静态绑定 和 动态绑定

绑定的其他叫法

静态绑定  == 前期绑定 == 编译时绑定

动态绑定  == 后期绑定 == 运行时绑定

为了方便区分：下面统一称呼为静态绑定和动态绑定

静态绑定

在程序运行前，也就是编译时期 JVM 就能够确定方法由谁调用，这种机制称为静态绑定

识别静态绑定的三个关键字以及各自的理解

如果一个方法由 private、static、final 任意一个关键字所修饰，那么这个方法是前期绑定的

构造方法也是前期绑定

private：private 关键字是私有的意思，如果被 private 修饰的方法是无法由本类之外的其他类所调用的，也就是本类所特有的方法，所以也就由编译器识别此方法是属于哪个类的

动态绑定

在运行时根据具体对象的类型进行绑定

除了由 private、final、static 所修饰的方法和构造方法外，JVM 在运行期间决定方法由哪个对象调用的过程称为动态绑定

如果把编译、运行看成一条时间线的话，在运行前必须要进行程序的编译过程，那么在编译期进行的绑定是前期绑定，在程序运行了，发生的绑定就是后期绑定

public class Father {  
 void drinkMilk(){

System.out.println("父亲喜欢喝牛奶");

}

}  
public class Son extends Father{  
 @Override void drinkMilk()

{ System.out.println("儿子喜欢喝牛奶"); }

public static void main(String[] args)

{

Father son = new Son();

son.drinkMilk();

}

}

Son 类继承 Father 类，并重写了父类的 dringMilk() 方法，在输出结果得出的是儿子喜欢喝牛奶。那么上面的绑定方式是什么呢？

上面的绑定方式称之为动态绑定，因为在你编写 Father son = new Son() 的时候，编译器并不知道 son 对象真正引用的是谁，在程序运行时期才知道，这个 son 是一个 Father 类的对象，但是却指向了 Son 的引用，这种概念称之为多态，那么我们就能够整理出来多态的三个原则：

继承

重写

父类对象指向子类引用

也就是说，在 Father son = new Son() ，触发了动态绑定机制

**动态绑定的过程**

虚拟机提取对象的实际类型的方法表；

虚拟机搜索方法签名；

调用方法。

动态绑定和静态绑定的特点

**静态绑定**

静态绑定在编译时期触发，那么它的主要特点是

1、编译期触发，能够提早知道代码错误

2、提高程序运行效率

动态绑定

1、使用动态绑定的前提条件能够提高代码的可用性，使代码更加灵活。

2、多态是设计模式的基础，能够降低耦合性。

## Mybatis

1. #{}和${}的区别是什么？

#{}是预编译处理，${}是字符串替换。

Mybatis在处理#{}时，会将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；

Mybatis在处理${}时，就是把${}替换成变量的值。

使用#{}可以有效的防止SQL注入，提高系统安全性。

1. 当实体类中的属性名和表中的字段名不一样 ，怎么办 ？

第1种： 通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致

<select id=”selectorder” parametertype=”int” resultetype=”me.gacl.domain.order”>

select order\_id id, order\_no orderno ,order\_price price form orders where order\_id=#{id};

</select>

第2种： 通过<resultMap>来映射字段名和实体类属性名的一一对应的关系

<select id="getOrder" parameterType="int" resultMap="orderresultmap">

select \* from orders where order\_id=#{id}

</select>

<resultMap type=”me.gacl.domain.order” id=”orderresultmap”>

<!–用id属性来映射主键字段–>

<id property=”id” column=”order\_id”>

<!–用result属性来映射非主键字段，property为实体类属性名，column为数据表中的属性–>

<result property = “orderno” column =”order\_no”/>

<result property=”price” column=”order\_price” />

</reslutMap>

1. 模糊查询like语句该怎么写?

第1种：在Java代码中添加sql通配符。

string wildcardname = “%smi%”;

list<name> names = mapper.selectlike(wildcardname);

<select id=”selectlike”>

select \* from foo where bar like #{value}

</select>

第2种：在sql语句中拼接通配符，会引起sql注入

string wildcardname = “smi”;

list<name> names = mapper.selectlike(wildcardname);

<select id=”selectlike”>

select \* from foo where bar like "%"#{value}"%"

</select>

1. 通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

Dao接口，就是人们常说的Mapper接口，接口的全限名，就是映射文件中的namespace的值，接口的方法名，就是映射文件中MappedStatement的id值，接口方法内的参数，就是传递给sql的参数。Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MappedStatement，举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到namespace为com.mybatis3.mappers.StudentDao下面id = findStudentById的MappedStatement。在Mybatis中，每一个<select>、<insert>、<update>、<delete>标签，都会被解析为一个MappedStatement对象。

Dao接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。

Dao接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Dao接口生成代理proxy对象，代理对象proxy会拦截接口方法，转而执行MappedStatement所代表的sql，然后将sql执行结果返回。

1. Mybatis是如何进行分页的？分页插件的原理是什么？

Mybatis使用RowBounds对象进行分页，它是针对ResultSet结果集执行的内存分页，而非物理分页，可以在sql内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用Mybatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。

1. Mybatis是如何将sql执行结果封装为目标对象并返回的？都有哪些映射形式？

第一种是使用<resultMap>标签，逐一定义列名和对象属性名之间的映射关系。

第二种是使用sql列的别名功能，将列别名书写为对象属性名，比如T\_NAME AS NAME，对象属性名一般是name，小写，但是列名不区分大小写，Mybatis会忽略列名大小写，智能找到与之对应对象属性名，你甚至可以写成T\_NAME AS NaMe，Mybatis一样可以正常工作。

有了列名与属性名的映射关系后，Mybatis通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

1. 如何执行批量插入?

首先,创建一个简单的insert语句:

<insert id=”insertname”>

insert into names (name) values (#{value})

</insert>

然后在java代码中像下面这样执行批处理插入:

list<string> names = new arraylist();

names.add(“fred”);

names.add(“barney”);

names.add(“betty”);

names.add(“wilma”);

// 注意这里 executortype.batch

sqlsession sqlsession = sqlsessionfactory.opensession(executortype.batch);

try {

namemapper mapper = sqlsession.getmapper(namemapper.class);

for (string name : names) {

mapper.insertname(name);

}

sqlsession.commit();

} finally {

sqlsession.close();

}

1. 如在mapper中如何传递多个参数?

第1种：

//DAO层的函数

Public UserselectUser(String name,String area);

//对应的xml,#{0}代表接收的是dao层中的第一个参数，#{1}代表dao层中第二参数，更多参数一致往后加即可。

<select id="selectUser"resultMap="BaseResultMap">

select \* fromuser\_user\_t whereuser\_name = #{0} anduser\_area=#{1}

</select>

第2种： 使用 @param 注解:

import org.apache.ibatis.annotations.param;

public interface usermapper {

user selectuser(@param(“username”) string username,

@param(“hashedpassword”) string hashedpassword);

}

然后,就可以在xml像下面这样使用(推荐封装为一个map,作为单个参数传递给mapper):

<select id=”selectuser” resulttype=”user”>

select id, username, hashedpassword

from some\_table

where username = #{username}

and hashedpassword = #{hashedpassword}

</select>

1. Mybatis动态sql是做什么的？都有哪些动态sql？能简述一下动态sql的执行原理不？

Mybatis动态sql可以让我们在Xml映射文件内，以标签的形式编写动态sql，完成逻辑判断和动态拼接sql的功能。

Mybatis提供了9种动态sql标签：trim|where|set|foreach|if|choose|when|otherwise|bind。

其执行原理为，使用OGNL从sql参数对象中计算表达式的值，根据表达式的值动态拼接sql，以此来完成动态sql的功能。

## 数据结构和算法

1. 假设执行语句S的时间为O(1)，则执行下列程序短的时间为（B）

for(i=1;i<=n;i++)

for(j=i;j<=n;j++)

S;

A. O(n) B. O(n2) C. O(n\*i) D. O(n+1)

1. 设栈最大长度为3，入栈序列为1，2，3，4，5，6，则不可能得出栈序列是（D）

A. 1，2，3，4，5，6 B. 2，1，3，4，5，6

C. 3，4，2，1，5，6 D. 4，3，2，1，5，6

1. 设有98个已排序列元素，采用二分法查找时，最大比较次数是（D）

A. 49 B. 15 C. 20 D. 7

1. Hash表示用于数据存储的一种有效的数据结构，Hash表等查找复杂度依赖于Hash值算法的有效性，在最好的情况下，Hash表的查找复杂度为（A）

A. O(1) B. O(logn) C. O(n) D. O(nlogn)

## 数据库

1. **事务是什么？**

事务是作为一个逻辑单元执行的一系列操作，一个逻辑工作单元必须有四个属性，称为 ACID（原子性、一致性、隔离性和持久性）属性，只有这样才能成为一个事务：

原子性

事务必须是原子工作单元；对于其数据修改，要么全都执行，要么全都不执行。

一致性

事务在完成时，必须使所有的数据都保持一致状态。在相关数据库中，所有规则都必须应用于事务的修改，以保持所有数据的完整性。事务结束时，所有的内部数据结构（如 B 树索引或双向链表）都必须是正确的。

隔离性

由并发事务所作的修改必须与任何其它并发事务所作的修改隔离。事务查看数据时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的数据。这称为可串行性，因为它能够重新装载起始数据，并且重播一系列事务，以使数据结束时的状态与原始事务执行的状态相同。

持久性

事务完成之后，它对于系统的影响是永久性的。该修改即使出现系统故障也将一直保持。

1. **数据库中的索引，主键，unique key 的区别和联系？索引的原理？索引的代价？MySQL的索引方式**

主键是表格里的(一个或多个)字段，只用来定义表格里的行;主键里的值总是唯一的。外键是一个用来建立两个表格之间关系的约束。这种关系一般都涉及一个表格里的主键字段与另外一个表格(尽管可能是同一个表格)里的一系列相连的字段。那么这些相连的字段就是外键。

1. **试设计一个商品相关数据库，要求包括商品类别及商品的基本信息，并完成下面的三个查询。**

1、请写出创建相关表的SQL语句。

2. 使用标准SQL，得到2008年1月份入库的所有商品信息。

3. 使用标准SQL，得到“Toy”分类下的所有商品信息。

4. 使用标准SQL，统计出所有包含商品价格低于100.00的分类。

1. **查询出选了2门课以上的学生的信息**

学生表   
student   
s\_id int   
s\_name varchar2   
  
课程表   
class   
c\_id int   
c\_name varchar2   
  
选课表   
xuanke   
id int   
s\_id int   
c\_id int

select \* from student where s\_id in ( select s\_id from xuanke group by s\_id having count(c\_id) >2)

## 编程题

1. **写一个Singleton出来**

第一种形式: 饿汉式

public class Singleton {

private Singleton(){}

　　 //在自己内部定义自己一个实例，是不是很奇怪

　　 //注意这是private 只供内部调用

　　 private static Singleton instance = new Singleton();

　　 //这里提供了一个供外部访问本class的静态方法，可以直接访问

　　 public static Singleton getInstance() {

　　　　 return instance;

　　 }

}

}

第二种形式 懒汉式单例模式（适合多线程安全）:

public class Singleton {

　　private static Singleton instance = null;

　　public static synchronized Singleton getInstance() {}

　　//这个方法比上面有所改进，不用每次都进行生成对象，只是第一次

　　//使用时生成实例，提高了效率

　　if (instance==null){

　　　 instance＝new Singleton();

return instance;

}

}

第三种：

由于synchronized的存在，效率很低，在单线程的情景下，完全可以去掉synchronized，为了兼顾效率与性能问题，改进后代码如下：

“双重检查锁”，主要在getSingleton()方法中，进行两次null检查。这样可以极大提升并发度，进而提升性能。

public class Singleton {

private static volatile Singleton singleton = null;

private Singleton(){}

public static Singleton getSingleton(){

if(singleton == null){

synchronized (Singleton.class){

if(singleton == null){

singleton = new Singleton();

}

}

}

return singleton;

}

}

第四种：静态内部类

public class SingletonInner {

private static class Holder {

private static SingletonInner singleton = new SingletonInner();

}

private SingletonInner(){}

public static SingletonInner getSingleton(){

return Holder.singleton;

}

}

以上存在的问题:http

1.序列化可能会破坏单例模式，比较每次反序列化一个序列化的对象实例时都会创建一个新的实例，解决方案如下：

第五种：

public class Singleton implements java.io.Serializable {

public static Singleton INSTANCE = new Singleton();

protected Singleton() {

}

//反序列时直接返回当前INSTANCE

private Object readResolve() {

return INSTANCE;

}

}

2. 使用反射强行调用私有构造器，解决方式可以修改构造器，让它在创建第二个实例的时候抛异常

public static Singleton INSTANCE = new Singleton();

private static volatile boolean flag = true;

private Singleton(){

if(flag){

flag = false;

}else{

throw new RuntimeException("The instance already exists ！");

}

}

还有问题：如果单例类维持了其他对象的状态时还需要使他们成为transient的对象，这种就更复杂了。解决：枚举单利

public enum SingletonEnum {

INSTANCE;

private String name;

public String getName(){

return name;

}

public void setName(String name){

this.name = name;

}

}

1. 有以下代码:

for (int i = 4; i > 0; i--){ int j = 0;

do {

j++;

if (j == 2) { break;

}

} while (j <= i);

System.out.print(j);

}

程序的运行结果是( 2 2 2 2)

1. **以下代码完成什么功能？如何实现的？**

public class ExtendSuper {  
 public static <T> void copy (List<? extends T> src , List<? super T> dest) {  
 for(int i=0;i<src.size();++i) {  
 dest.add(src.get(i));  
 }  
 }  
}

1. 以下代码的输出结果

public class FinallyReturn {

public static int testFinally() {

try {

return 1;

} catch (Exception e) {

return 0;

} finally {

System.out.println("execute finally!");

return 3;

}

}

public static void main(String[] args) {

int re = testFinally();

System.out.println(re);

}

}

程序的运行结果是

execute finally!

3

public class FinallyReturn {

public static int testFinally() {

try {

return 1;

} catch (Exception e) {

return 0;

} finally {

System.out.println("execute finally!");

return 3;

}

}

public static void main(String[] args) {

int re = testFinally();

System.out.println(re);

}

}

程序的运行结果是

execute finally!

1

1. **Java程序的初始化一般遵循**

1）静态对象（变量）优于非静态对象（变量）的初始化，其中，静态对象（变量）只初始化一次，非静态对象（变量）可能会初始化多次。（2）父类优先于子类进行初始化。（3）按照成员变量的定义顺序进行初始化。即使变量定义散布于方法定义之中，他们依然在任何方法（包括构造函数）被调用之前先初始化。

父类静态变量、父类静态代码块、子类静态变量、子类静态代码块、父类非静态变量、父类非静态代码块、父类构造函数、子类非静态变量、子类非静态代码块、子类构造函数

class A{

static {

System.out.println("A staic block");

}

{

System.out.println("A block");

}

public A(){

System.out.println("A constructor");

}

}

class B extends A{

static {

System.out.println("B staic block");

}

{

System.out.println("B block");

}

public B(){

System.out.println("B constructor");

}

}

public class staticTest {

public static void main(String[] args){

A a = new B();

System.out.println("--------------------");

B b = new B();

}

}

输出结果：

A staic block

B staic block

A block

A constructor

B block

B constructor

--------------------

A block

A constructor

B block

B constructor

1. **有以下代码，有错误么？否则，输出结果是:**

public class Ex {

public static void main(String[] args) {

Fx f = new Fx(5);

}

Ex() {

System.out.println("Ex, no args");

}

Ex(int i){

System.out.println("Ex int");

}

}

class Fx extends Ex {

Fx() {

super();

System.out.println("Fx,no-args");

}

Fx(int i) {

this();

System.out.println("i = [" + i + "],Fx int");

}

}

Ex, no args

Fx,no-args

i = [5],Fx int

1. **有以下代码，输出结果是:**

class Base {

public Base(){g();}

public void f() {

System.out.println("gase f()");

}

public void g(){

System.out.println("Base g()");

}

}

class Derived extends Base {

public void f() {

System.out.println("Derived f()");

}

public void g(){

System.out.println("Derived g()");

}

}

public class Test {

public static void main(String[] args) {

Base b = new Derived();

b.f();

b.g();

}

}

Derived g()

Derived f()

Derived g()