Java

## 一、初始java

1. 程序：指完成某些事物的一种既定方法和过程。（有序的指令集合）

2. C/S 客户端/服务器：桌面应用程序（超市收银员、银行）

B/S 浏览器/服务端：淘宝、京东、门户网站

3. java特点：面向对象、安全、高性能、跨平台（可移植性）

4. java三大平台：javaSE：java开发桌面应用程序平台

JavaEE：开发B/S平台

javaME（已被淘汰 诺基亚：塞班系统 多普达：HTC）



5. jdk环境变量配置：

JAVA\_HOME：jdk安装的根目录

path：%JAVA\_HOME%\bin;

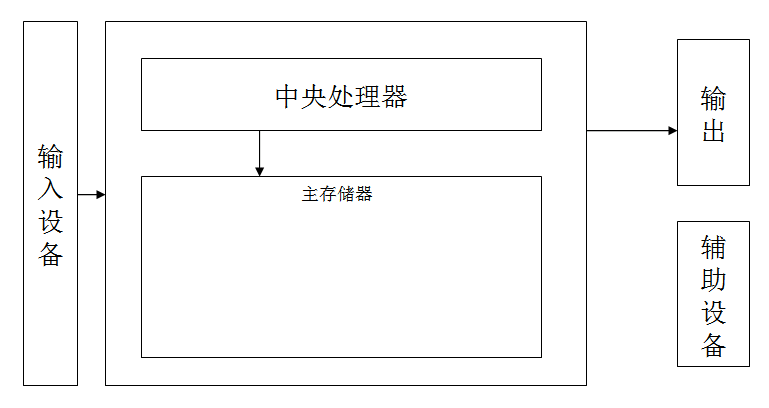
目的：使计算机在任意目录下都能识别jdk里的命令

6. EditPlus下载地址：<https://www.editplus.com/latest4.html>

EditPlus注册地址：https://www.jb51.net/tools/editplus/

## 二、面向过程编程

1. 计算机硬件：



2. 基本数据类型（8个）：

数值型：整形： byte short（短整型） int（整型） long（长整型）

非整形：double（双精度） float（单精度，后必须加f）

非数值型：字符型 char

布尔型 boolean

3. 命名规则：

(1)变量命名规则

首字母：字母、下划线、$

其余部分：任意多的数字、字母、下划线、$

不能是Java中的关键字：goto

(2)类命名规则：同变量命名规则，但首字母大写（驼峰式）

4. 隐式类型转换：int a = ‘A’;

强制类型转换：int b = (int)1.02;

|  |
| --- |
| public class Caculate{  public static void main(String[] args){  char a = 'A';  char b = 'B';  System.out.println("" + a + b); //输出AB  System.out.println(a + b); //输出131  }  } |

5. 自加自减运算

|  |
| --- |
| public class Caculate{  static char a;  public static void main(String[] args){  int a = 53,b = 6;  int c;  c = a + b++;  System.out.println(c); //59  c = a + ++b;  System.out.println(c); //61  c = a + --b;  System.out.println(c); //60  }  } |

算数运算符：+ - \* / %

关系运算符：> >= < <= == !=

逻辑运算符：&& ||（短路运算符） &、|（一直都会被执行） !

运算符优先级：

() 左→右

! ++ -- **右→左**

\* / 左→右

+ - 左→右

> < 左→右

== != 左→右

&& 左→右

|| 左→右

= \*= /= += -= **右→左**

三、条件结构：

1. if – else语句

2. switch – case语句（default放于最后时可省略break）

jdk1.7前switch参数只能是byte、char、short、int

jdk1.7及之后switch也可接受String、boolean、enum

2. 若两个引用对象相等（==）指两个对象指向的内存地址相同。

最好是 常量.equals(变量) 可避免不必要的空指针异常。

3. nextInt()方法在扫描到空白符的时候会将前面的数据读取走，但会丢下空白符“\r”在缓冲区中。但是，nextLine()方法在扫描的时候会将扫描到的空白符一同清理掉。

四、循环结构：

1. while() {}

2. do{}while();

3. for(;;)

|  |
| --- |
| int[] num = new int[]{1,2,3,4};  /\*\*foreach循环  用来循环集合 每一次循环从集合中拿出一个值赋值给声明的变量  供循环体使用  \*\*/  for(int temp:num){  System.out.print(temp+"\t");  } |

实例：

|  |
| --- |
| for(int i = 1 ; i <= 10 ; i++){    for(int j = 10 ; j > i ; j--){  System.out.print(" ");  }  for(int j = 1 ; j <= 2\*i-1 ; j++){  System.out.print("\*");  }  System.out.println();  } |

五、数组

1. 存放相同类型数据的容器(占用一块连续的内存地址)

2. 定义语法：int[] score = new int[10];(int未赋值，则默认值为0)

int[] score = {1,2,3,4};

3. 二维数组：

|  |
| --- |
| int [][] m = {{1,2,3},{4,5,6},{7,8,9}};  for(int temp[]:m){  for(int t:temp){  System.out.print(t+" ");  }  System.out.println();  } |

## 三、面向对象编程

1. 类：类是具有相同特征和行为的一组对象的集合。

相同特征：属性；相同行为：方法

类是一个概念，不对应具体的实体

2. 封装：将属性和方法写进类的过程，隐藏内部信息。

3. 方法的重载：在同一个类中，方法名相同，参数列表不同（跟返回值类型无关）

|  |
| --- |
| **public** **class** OverLoad{  //参数不定长度的方法,str变为一个数组  //参数类型必须一致，jdk1.5及以后才可使用  **public** **static** **void** talk(String ... str){  System.***out***.println(str.length);  **for**(String s:str){  System.***out***.print(s+"\t");  }  System.***out***.println();  System.***out***.println();  }  **public** **static** **void** main(String[] args){  OverLoad.*talk*();  OverLoad.*talk*("a","b");  OverLoad.*talk*("aa","bb","cc");  }  } |

4. 消息的传递：对象之间的相互调用

5. 类之间的关系：

依赖：相对独立的两个类A和B，A使用B，形成依赖关系

java中代码表现形式：局部变量，方法的参数

关联：相对独立的两个类A和B，当A持有B时，形成关联关系

java中代码表现形式：B作为A的属性

泛化：

6. 构造方法：

|  |
| --- |
| **public** Hunter(String name) {  //此种情况this不可省略  **this**.name = name;  } |

7. static

static修饰方法时，方法里不能有this

static修饰方法或属性时，可以直接通过方法名+“.”来调用

8. 垃圾回收(gc)：finalize();

9. 继承：

Java中所有类都直接或间接继承自Object类（java.lang包）

子类构造方法的第一行必须是父类的构造方法。

java中的继承是单继承，只能拥有一个父类。

10. 访问修饰符：

private：本类内访问。

默认：同一包中的类。

protected：本类、子类中及同一个包下的类。

public：所有类。

11. 方法的重写（覆写）：Override

子类继承父类时，子类中的方法声明与父类中的方法声明相同，实现不同。

12. 父类的引用指向了子类的对象（Animal animal = new Tiger(“华南虎”);）

产生了两种对象；

animal有两种类型，此时animal在编译阶段是Animal类型，在执行阶段是Tiger类型；

13. 抽象类（abstract）：

普通类中不能有抽象方法，抽象类中可以没有抽象方法。

抽象类不能实例化，只能被继承。

抽象类里可以有有实现的方法。

14. 接口：

接口不能实例化；

在接口里定义的所有变量都默认是**public static final 数据类型 变量名**（即常量）；

jdk1.8以前接口里的方法都不能有方法体；

jdk1.8及以后接口里可以有带实现的方法；

|  |
| --- |
| **public** **interface** Ianimal {  **public** **void** dead() ;  //default只能通过实现接口的类来调用  **public** **default** **void** eat1() {  System.***out***.println("吃东西1");  }  //static只能通过接口名调用  **public** **static** **void** eat2() {  System.***out***.println("吃东西2");  }  } |

类可以实现多个接口，中间用逗号隔开。

接口可以由普通的类来实现，但这些类需实现接口的所有方法。

接口可以由抽象类来实现，抽象类可选择实现接口的部分方法。

15. final:

修饰属性，则变量变为常量，一旦被赋值，则不能被修改。（常量定义时变量字母全大写）

修饰方法，则该方法不能被重写。

修饰类，则该类不能被继承。

16. 内部类（内嵌类）：需要隐藏一个不为人知的类

* 内部类可以很好的实现隐藏，一般的类是不允许用private和protected修饰的
* 内部类拥有外部类所有的元素访问修饰符
* 内部类可实现多重继承
* 内部类可以避免修改接口，实现自己方法与接口中同名的方法。
* 静态内部类：

|  |
| --- |
| **public** **class** InnerClassTest {  **static** **class** StaticInnerClass{  **public** **void** Test() {  System.***out***.println("静态内部类");  }  }  **public** **static** **void** main(String[] args) {  InnerClassTest.StaticInnerClass sic = **new** InnerClassTest.StaticInnerClass();  sic.Test();  }  } |

* 非静态内部类：
* 成员内部类：在类里边声明一个普通的类

如果要使用成员内部类，必须有外部类的实例

|  |
| --- |
| **public** **class** InnerClassTest1 {  **public** **class** MemberInnerClass{  **public** **void** Test() {  System.***out***.println("成员内部类");  }  }  **public** **static** **void** main(String[] args) {  InnerClassTest1 ic = **new** InnerClassTest1();  InnerClassTest1.MemberInnerClass mic = ic.**new** MemberInnerClass();  mic.Test();  }  } |

* 区域内部类：在方法里声明一个类

目的：分离一些算法、业务

|  |
| --- |
| **public** **class** InnerClass2 {  **public** **int** add(**int** a,**int** b) {  **class** Add{  **int** a;  **int** b;  **public** Add() {}  **public** Add(**int** a,**int** b) {  **this**.a = a; **this**.b = b;  }  **public** **int** add() {  **return** **this**.a + **this**.b;  }  }  Add tempAdd = **new** Add(a,b);  **return** tempAdd.add();  }    **public** **static** **void** main(String[] args) {  InnerClass2 ic = **new** InnerClass2();  System.***out***.println(ic.add(1, 2));  }  } |

* 匿名内部类：

|  |
| --- |
| **public** **class** Test {  **public** **static** **void** main(String[] args) {  Hunter hunter = **new** Hunter("武松");  hunter.shoot(**new** Iarm() {  @Override  **public** Animal use(Animal animal) {  System.***out***.println(hunter.getName()+"发现了"+animal.getName());  System.***out***.println(hunter.getName()+"出拳重击了"+animal.getName());  animal.dead();  **return** animal;  }  }, **new** Animal("吊睛白额大虎","活的") {  @Override  **public** **void** dead() {  System.***out***.println("我是虎族，然而我惨遭不幸，英年早逝");  **this**.setState("死的");  }  });  System.***out***.println(hunter);  }  } |

17. 包装类（java中为基本数据类型提供了封装类）

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| byte | char | short | int | float | double | long | boolean |
| Byte | Character | Short | Integer | Float | Double | Long | Boolean |

jdk1.5以前：Integer a = new Integer(2);

jdk1.5及以后支持自动装箱：Integer a = 2；

拆箱：int f = a; 自动装箱和拆箱时注意空指针异常

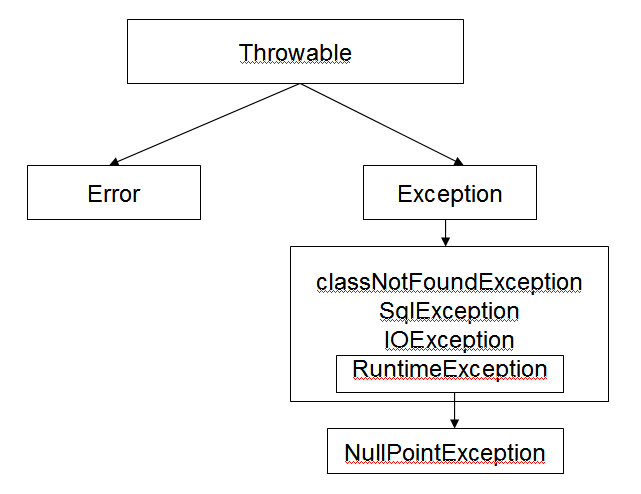
|  |
| --- |
| //如果用类自动装箱在 -128~127(包括端点 )之间是不再新开辟内存空间的。  Integer a = 2;  Integer b = 2;  System.***out***.println(a == b); //true    Integer e = **new** Integer(2);  Integer f = **new** Integer(2);  System.***out***.println(e == f); //false    Integer c = 200;  Integer d = 200;  System.***out***.println(c == d); //false |

18. String

* String类被final修饰，不能被继承
* String是一个字符数组，每创建一个都去开辟新的内存空间。
* 常用方法：

|  |
| --- |
| String strNum = "123";  String str = "aBcBd";  //获取字符串长度  System.***out***.println(strNum.length()); //输出3  //转化为大、小写字母  System.***out***.println(str.toUpperCase()); //ABCBD  System.***out***.println(str.toLowerCase()); //abcbd  //转化为数值类型  **int** a = Integer.*parseInt*(strNum);  System.***out***.println(a); //123(int)  //字符串连接  System.***out***.println(strNum.concat(str)); //123aBcBd  //字符串替换  System.***out***.println(str.replace('B', 'b')); //abcbd  System.***out***.println(str.replaceFirst("B", "b")); //abcBd  System.***out***.println(str.replaceAll("B","b")); //abcbd  //查找索引  System.***out***.println(str.indexOf('c')); //2,若找不到则返回-1  //获得子串  System.***out***.println(str.substring(2)); //cBd 获得索引由2开始的子串（包括端点）  System.***out***.println(str.substring(1, 3)); //Bc 获得索引由[1,3)的子串  //切分字符串  String[] newStr = str.split("B");  **for** (String string : newStr) {  System.***out***.print(string+" "); //a c d  } |

19. 异常处理：



* 异常不是编译错误，异常发生在运行期。
* 异常在java中代表一个错误的对象
* 有些可能发生的异常在程序中必须处理，有些则不必须处理
* java中的异常分为checked异常和Runtime异常。

Runtime异常jvm都能为我们处理，无需开发人员手动添加try……catch……语句

checked异常（受查异常）必须自行添加try……catch……语句

* 常见异常

空指针异常NullPointException：一个null对象去调用方法。

运算时异常：ArithmeticException

类型转化错误异常：ClassCastException

数组越界异常ArrayIndexOutOfBoundsException

* 捕获异常时catch、finally是可以省略的，同时他们是可以嵌套的。

20. 泛型：（jdk1.5及以后开始支持泛型）

简单的理解为声明占位符，在实例化时需将占位符填上所需类型（不能是基本数据类型）

泛型继承时，父类的泛型应在子类中声明。

21. Data类：

|  |
| --- |
| //获得当前日期对象  Date date = **new** Date();  SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  System.***out***.println(sdf.format(date)); //2018-08-06 14:16:07  //获取今天几号,几月，几年  **int** dataDay = Calendar.*getInstance*().get(Calendar.***DAY\_OF\_MONTH***);  **int** dataMonth = Calendar.*getInstance*().get(Calendar.***MONTH***)+1;  **int** dataYear = Calendar.*getInstance*().get(Calendar.***YEAR***);  System.***out***.println(dataDay); //6  System.***out***.println(dataMonth); //8  System.***out***.println(dataYear); //2018 |

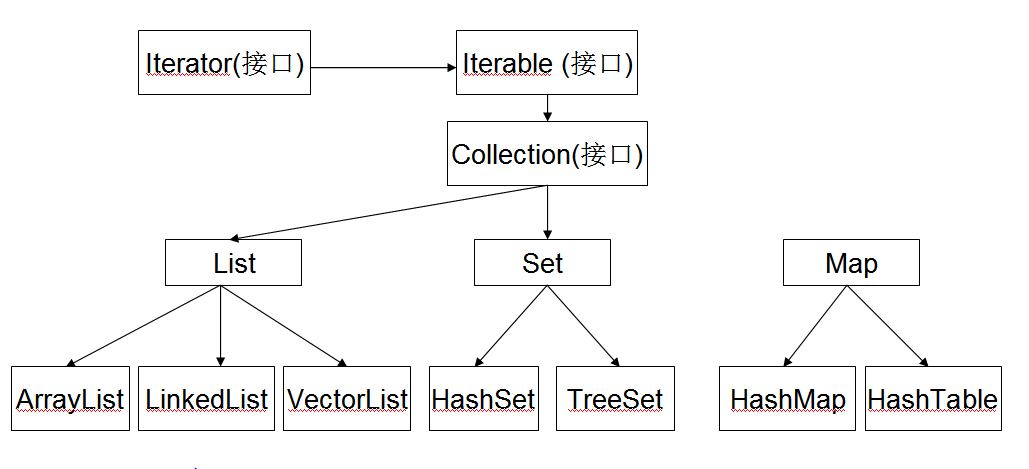
22. Random类：

|  |
| --- |
| //可传入一个种子,若传入一个种子，每次产生的随机数固定  Random random = **new** Random();  System.***out***.println(random.nextInt());  //产生[0,1)之间的随机小数  System.***out***.println(random.nextDouble()); |

23. Math类：

|  |
| --- |
| //产生[0,1)之间的随机小数  **double** ran = Math.*random*();  System.***out***.println(ran); |

24. 集合：



List按对象被放置容器的先后顺序排列对象。

Set不接受重复对象，有自己的顺序。

从jdk1.5以后开始支持集合中使用泛型。

|  |
| --- |
| /\* ArrayList对象是可变长度的  \* 好处：访问和遍历对象的时候，能够提供很好的性能  \* 弊端：删除和插入速度很慢，是基于数组实现的，底层容量默认是10  \* **ArrayList是非线程安全的，VectorList是线程安全的**  \* \*/  Scanner sc = **new** Scanner(System.***in***);  //通过泛型控制放入集合的对象类型  ArrayList<String> arrayList = **new** ArrayList<String>();  //添加函数  arrayList.add("张三");  //在指定索引处插入对象  arrayList.add(0,"李四");  /\*arrayList.size()显示Arraylist中的总数  \* arrayList.get(i)得到索引为i的对象  \* arrayList.remove(i)移除索引为i的对象  \* arrayList.set(i,"")替换索引为i的对象  \* \*/  **for**(**int** i = 0;i< arrayList.size();i++) {  System.***out***.println(arrayList.get(i));  }  /\* LinkedList是以链表的形式存储的  \* 好处：LinkedList删除和修改速度快  \* 弊端：LinkedList查询速度慢  \* addFirst(),addLast()  \* getFirst(),getLast()  \* \*/ |

Set：

|  |
| --- |
| /\* List允许存重复的值，Set不允许存重复的值  \* Set没有索引，Set有自身的排序方法  \* \*/  Set<String> set = **new** HashSet<String>();  **for**(**int** i = 0;i < 10;i++) {  set.add("abc"+i);  }  //Set没有get方法，需要用迭代器  Iterator it = set.iterator();  **while**(it.hasNext()) {  System.***out***.println(it.next());  } |

HashMap在1.8以前是用数组+链表实现，在1.8及以后用数组+红黑树实现。

|  |
| --- |
| //Map的key不允许重名  Map<String,String> map = **new** HashMap<String,String>();  map.put("key1", "value1");  System.***out***.println(map.size());  //获取Map中所有的key，返回一个Set集合  Set<String> keySet = map.keySet();  **for** (String string : keySet) {  System.***out***.println(map.get(string));  } |

25. 输入输出

File对象：

|  |
| --- |
| **public** **static** **void** main(String[] args) {  /\* java.io包约83个类和接口组成  \* 主要分为字节流方向和字符流方向  \* File类对应的是一个文件或一个文件夹,IO读写是受查时异常  \* \*/  //File类对应一个文件  File file = **new** File("F:\\MyJavaWorkSpace\\workspace\\Util\\Test\\test.txt");  //得到文件长度  System.***out***.println(file.length());  //得到文件名字  System.***out***.println(file.getName());    //File类对应一个文件夹  File file2 = **new** File("F:\\MyJavaWorkSpace\\workspace\\Util\\Test");  //判断文件是否存在  System.***out***.println(file2.exists());  //判断是否是一个目录  System.***out***.println(file2.isDirectory());  //判断是否是一个文件  System.***out***.println(file2.isFile());  //得到绝对路径  System.***out***.println(file2.getAbsolutePath());  } |

FileInputStream、FileOutputStream类：

|  |
| --- |
| **public** **static** **void** copyFile() {  FileInputStream fis = **null** ;  FileOutputStream fos = **null**;  **try** {  //1. 将硬盘上的物理文件读取到java系统中，建立读入的通道  fis = **new** FileInputStream("F:\\MyJavaWorkSpace\\workspace\\Util\\Test\\test.txt");  //2. 从java系统中向物理硬盘上写入相应内容，建立写出的通道  fos = **new** FileOutputStream("F:\\MyJavaWorkSpace\\workspace\\Util\\Test\\testcopy.txt");    //缓冲区  **byte**[] b = **new** **byte**[1024];  //fis.read(b) 从读入通道里读入缓存区  **while**(fis.read(b) != -1) {  //从缓冲区读出写入物理磁盘上  fos.write(b);  }  }**catch**(IOException e) {  e.printStackTrace();  }**finally** {  //关闭读入、写出通道  **try** {  fis.close();  fos.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

BufferedInputStream、BufferedOutputStream类

|  |
| --- |
| **public** **static** **void** copyFile() {  //速度比FileInputTest快  BufferedInputStream bis = **null**;  BufferedOutputStream bos = **null**;  **try** {  FileInputStream fis = **new** FileInputStream("F:\\MyJavaWorkSpace\\workspace\\Util\\Test\\test.txt");  bis = **new** BufferedInputStream(fis);    FileOutputStream fos = **new** FileOutputStream("F:\\MyJavaWorkSpace\\workspace\\Util\\Test\\testcopy2.txt");  bos = **new** BufferedOutputStream(fos);    **byte**[] b = **new** **byte**[1024];  **while**(bis.read(b) != -1) {  bos.write(b);  }  } **catch** (IOException e) {  e.printStackTrace();  }**finally** {  **try** {  bis.close();  bos.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

FileReader、FileWriter类：

|  |
| --- |
| **public** **static** **void** copyFile() {  FileReader fr = **null**;  FileWriter fw = **null**;  File file = **new** File("F:\\MyJavaWorkSpace\\workspace\\Util\\Test\\test.txt");  **try** {  fr = **new** FileReader(file);  fw = **new** FileWriter("F:\\MyJavaWorkSpace\\workspace\\Util\\Test\\test3.txt");    **char**[] ch = **new** **char**[100];  **while**(fr.read(ch) != -1) {  fw.write(ch);  }  } **catch** (IOException e) {  e.printStackTrace();  }**finally** {  **try** {  fr.close();  fw.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

BufferedReader、BufferedWriter类：

|  |
| --- |
| **public** **static** **void** copyFile() {  BufferedReader br = **null**;  FileReader fr;  BufferedWriter bw = **null**;  FileWriter fw;    **try** {  fr = **new** FileReader("F:\\MyJavaWorkSpace\\workspace\\Util\\Test\\test.txt");  br = **new** BufferedReader(fr);  fw = **new** FileWriter("F:\\MyJavaWorkSpace\\workspace\\Util\\Test\\test4.txt");  bw = **new** BufferedWriter(fw);    String str = "";  **while**((str = br.readLine()) != **null**) {  bw.write(str);  bw.newLine();  }  } **catch** (IOException e) {  e.printStackTrace();  }**finally** {  **try** {  bw.close();  br.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

## 三、Oracle数据库

1. 数据库：即数据仓库，由表、关系、操作对象组成，数据被储存在表中。

2. oracle是对象关系型数据库

3. oracle内置用户：sys、system、scott

4. 逻辑结构：表空间是最大的逻辑单元，一个oracle数据库至少包含一个表空间，即system的系统空间。

|  |
| --- |
| *--创建表空间，必须由拥有dba身份的用户创建*  *--创建表空间语法*  create tablespace java9space  *--指定文件的位置和名字*  datafile 'F:\oracle\app\oradata\userdata\java9space.dbf'  *--初始大小,10兆*  size 10m  *--开启自动扩展 每次5兆*  autoextend on next 5m;  *--创建用户*  create user java9  identified by java9  default tablespace java9space;  *--给用户赋予权限*  grant connect to java9; *--连接的权限*  grant resource to java9; *--操作表空间的权限*  *--回收权限*  revoke connect from java9;  create table testtable  (  tid varchar2(32) constraint PK\_KID PRIMARY KEY, *--创建字段，并设置其为主键,PK\_KID为主键名*  username varchar2(32),  age number(10,0),  address varchar2(32) *--最后一个不需要逗号*  );  *--查看表*  select \* from testtable;  *--插入内容*  insert into testtable(tid,username,age,address)  values('1','王大傻',18,'秦皇岛');  insert into testtable(tid,username,age,address)  values('2','王二叔',18,'秦皇岛');  *--更改*  update testtable set username = '王二傻',age = 16 where tid = 2;  *--删除数据*  delete from testtable where username = '王大傻' and age = 18;  *--排序（正序asc，倒叙desc）*  select \* from testtable order by tid asc;  *--修改表名*  alter table testtable1 rename to testtable;  *--修改列名*  alter table testtable rename column username to username1;  *--修改字段类型（列），修改字段类型的所在列时，该列必须为空*  alter table testtable modify address number(10,0);  *--添加字段*  alter table testtable add address varchar2(32);  *--删除字段（列）*  alter table testtable drop column address;  *--删除表*  drop table testtable;  *--删除用户*  drop user java9;  *--删除表空间*  drop tablespace java9space including contents and datafiles; |

5. 数据库设计：

* 数据库范式（规范）：

第一范式：确保每列的原子性

第二范式：在满足第一范式的基础上，并且除了主键以外的其他列，都依赖于该主键，即每个表只描述一件事情

第三范式：在满足第一、第二范式上，表中所有的列都不传递依赖主键列

主键不能有任何的业务意义。

* 软件项目开发周期：

(1)需求阶段：跟客户沟通，需求文档

(2)概要设计阶段：设计数据库模型：powerdesigner

分析名词：

实体→表

属性→字段

关系→主外键

(3)详细设计阶段：出数据库文档，对数据库中的每一个表进行详细的描述

出详细设计文档 接口、类、方法描述等

(4)开发阶段：根据详细设计文档里面设计的类、接口、方法进行编写代码

(5)测试阶段：测试

(6)实施阶段：安装部署给客户用

* 表与表之间的关系：

一对一：

一对多：由多的一方来记关系

多对一：由多的一方来记关系

多对多：由另一张弱实体表记录关系

* B表中有一个字段（n）存的值是A表中的主键，则n称为A表的外键

|  |
| --- |
| alter table STU\_COURSE  add constraint FK\_COURSEID foreign key (COURSEID)  references COURSE (COURSEID); |

6. 连接查询

内连接：

|  |
| --- |
| *--内连接查询特点：能够关联上的数据全部关联上，并且显示出来*  *--关联不上的数据不显示，关联通过主外键设置*  *--查看有班级学生的信息*  select \* from student stu  inner join class c on stu.classid = c.classid; |

外连接：

|  |
| --- |
| *--外连接(左外连接left 右外连接right)*  *--查看多有学生的班级信息*  select \* from student stu  left join class c on stu.classid = c.classid; |

交叉连接：

|  |
| --- |
| *--交叉连接*  select \* from student stu,class c where stu.classid = c.classid; |

7. 数据库语言分类：

DDL(Data Definition Language):用于描述数据库中要存储的现实世界实体的语言 create、alter、drop、truncate

DCL(Data Control Language):数据控制语言，用于定义数据库用户的语言 grant、revoke

DQL(Data Query Language)：数据库查询语言 select

DML(Data Manage Language)：数据操作语言，用于修改表中的数据 delete、update、insert、

* truncate和delete区别：

truncate删除数据不能恢复；

truncate速度较delete快；

truncate不能用于外键约束应用的表；

truncate属于DDL，而delete属于DML

8. 数据的完整性：

约束

默认约束：alter table TEST1 modify testtage default 18;

检查约束：

外键约束：删除数据时注意有引用的时候不能删除主键记录

|  |
| --- |
| create table test1  (  pid varchar2(32) constraint PK\_pid PRIMARY KEY,  pname varchar2(32) not null,  psex varchar2(4) constraint ch\_sex check (psex='男' or psex='女'),  page number(10,2) default 18  )  *--删除约束*  alter table test1 drop constraint ch\_sex;  *--删除非空约束*  alter table test1 modify pname null; |

9. 查询

* select 字段名集合 from 表名

[where 查询的条件 多个条件之间用and or连接]

[group by 字段] 分组查询(分组查询时分组的字段必须显示，并且显示的字段只能是分组的字段)

[having 条件] 分组的条件

[order by 排序的字段] asc|desc

|  |
| --- |
| *--查询人数大于2的所有班级的人数*  *--多字段分组用“，”隔开，分组的顺序是字段排列的顺序*  select cla.classname,count(cla.classname) from student stu  right join classes cla on stu.classid = cla.classid  group by cla.classname  having count(cla.classname) >= 2; |

* 去除重复：distinct select distinct stu.name from students stu;
* 模糊查询：select \* from student stu where stu.name like '王%';

select \* from test1 where page between 13 and 19;

10. 函数

聚合函数：求和sum、均数avg、最大值max、最小值min、个数count

11. 子查询：在查询里面嵌套查询语句。

|  |
| --- |
| *--查询年龄在199，299,300的人*  select \* from test1 t where t.page in (199,299,300);  *--集合的概念*  *--使用union关键字的前提 两个查询语句的字段要保持完全一致*  *--union会去除重复数据（并集）*  *--union all为完全合并两个结果集*  *--intersect获取两个结果集的交集*  *--minus为差集*  select \* from test1 t where t.page < 199  union  select \* from test1 t where t.page > 299;  *--内置的日期函数，转换函数，数字函数* |

12.

|  |
| --- |
| *--创建索引*  create index 索引名字 on 表名（需要创建索引的列名）  *--导出*  exp userid = 用户名/密码 file= ‘导出的路径/文件名.dmp’ owner= （想要导出数据的用户名）  *--导入*  imp userid = 用户名/密码 file=‘导入文件的路径/文件名.dmp’ full = y;  *--为创建视图授权*  grant create view to 用户名  *--创建视图（视图是一个虚拟的表，不建议做修改、删除）*  create view 视图名as查询语句 |

**四、JDBC（Java DataBase Connectivity）**

## 五、HTML（Hyper Text Markup Language）

1. 历程：

1.0 1993

2.0 1995

3.2 1996（w3c组织发布定义成为一组标准）

4.0 1997

4.01 1999

h5 2014.09

2. 常用标签：

<pre></pre>：预格式标签

<ol><li></li></ol>：有序列表

<b></b>：加粗

<i></i>：倾斜

<del></del>：中间画横线删除

<sub></sub>：下标签

<sup></sup>：上标签

<img alt="hehe" />：当图片找不到时，显示alt中的信息

<a target="\_blank"></a>：打开新网页，默认为\_self

**html转移符**：&lt; 小于号 &gt; 大于号

<span></span>：行级标签（元素）

<div></div>：块级元素

<td colspan="2" rowspan="2">

3. get提交方式：明文提交，有长度限制

post提交方式：隐藏式提交

4. css样式：

|  |
| --- |
| body{  background-image: url(img/pic1.jpg);  /\* background-repeat默认为repeat\*/  background-repeat: initial;  } |
| .textClass{  text-align: center;  border: 1px solid red;  text-transform: uppercase; /\*让字母大写\*/  width: 800px;  height: 100px;  word-spacing: 20px; /\*单词间距\*/  letter-spacing: 4px; /\*字母标签\*/  text-decoration: underline; /\*下划线\*/  font-family: serif /\*cursive,serif,sans-serif,monospace,fantansy\*/  font-style: oblique; /\*italic和oblique都便是倾斜\*/  font-weight: bold; /\*字体加粗\*/  font-size:30px; /\*字体大小\*/  } |
| .aClass{  text-decoration: none;  }  a:link{ /\*未被访问过的\*/  color: #000000;  }  a:visited{ /\*已访问过的\*/  color: #FF0000;  }  a:hover{ /\*鼠标接触\*/  color: #0000ff;  }  a:active{ /\*鼠标点击\*/  color: #00ff00;  } |
| /\*子代选择器\*/  .ul>li{  list-style-type: none;  } |
| table{  border-collapse: collapse; /\*单线表格\*/  width: 700px;  text-align: center;  }  table tr td{  border:1px solid black;  } |
| span{  border: 1px solid #00FF00;  display: block;  /\*block转化为块级元素，inline-block转化为行内块级元素\*/  height: 200px;  } |

注：

* 盒子模型的左右边距叠加，上下边距不会叠加
* 行级元素是无法直接设置高的

5. javascript：基于对象，弱类型，解释性语言

java：面向对象，强类型，编译型语言

(1) 关系运算符

|  |
| --- |
| <script>  var i= 1;  if(i == "1"){ //ok,只判断内容，(!=)  alert("ok");  }else{  alert("no");  }  if(i === "1"){ //no，判断内容和类型，(!==)  alert("ok");  }else{  alert("no");  }  </script> |

(2) var修饰

|  |
| --- |
| var a = 1;  b = 1; //不加var修饰可以被delete掉  delete a;  delete b; |

(3) 对象

|  |
| --- |
| var obj = {name:"李四",age:100};  for(var s in obj){  alert(s + "==" + obj[s]);  } |

(4) 数组

|  |
| --- |
| var arrs = [1,2,"abc"];  var arrs = new Array(1,2,"abc");  var arrs = new Array(new Array(1,2,3),new Array(4,5,6));  alert(arrs.toString()); //1,2,3,4,5,6  arrs.join("-"); //给数组中的元素间增加“-”  arrs.push("bcd"); //给原数组结尾追加元素  var arrs2 = arrs.concat("aaa"); //给数组结尾追加元素,返回新数组  alert(arrs2.toString()); //1,2,3,4,5,6,bcd,aaa  delete arrs2[3]; //删除数组中指定位置的值，位置仍在  alert(arrs2.toString()); //1,2,3,4,5,6,bcd, |

(5) 正则表达式

|  |
| --- |
| function checkInput(){  var regex = /^[\u4e00-\u9fa5]{2,}$/;  var testValue = document.getElementById("inputtest").value;  var flag = regex.test(testValue);  alert(flag);  } |

(6) alert输入框

|  |
| --- |
| var text = prompt("请输入内容");  alert(text); |

(7) 执行

|  |
| --- |
| //延迟执行代码，注意引号  setTimeout("window.location.href='http://www.baidu.com'",5000); |
| var id;  function test(){  //循环执行代码  id = window.setInterval("alert(1)",5000);  }  function test2(){  //停止执行代码  window.clearInterval(id);  } |

(8) 事件

|  |
| --- |
| <input type="button" value="点击事件" onclick="" />  <input type="button" value="获得焦点事件" onfocus="" />  <input type="button" value="失去焦点事件" onblur="" />  <input type="text" value="value值被修改事件" onchange="" />  <input type="button" value="鼠标按下事件" onmousedown="" /> <!--操作对象不同-->  <input type="button" value="鼠标抬起事件" onmouseup="" />  <input type="button" value="鼠标移动到对象上" onmouseover="" />  <input type="button" value="鼠标移动到" onmousemove="" />  <input type="button" value="鼠标移走事件" onmouseout="" /> |

6. jQuery：jQuery是javascript的兼容多个浏览器的框架。

* 优点：轻量级；强大的选择器。
* 常用层次选择器：
  + 空格表示后代
  + > 表示子代
  + + 表示紧接在该元素后面的兄弟标签
  + ~ 表示选中该元素后面的所有标签