Java程序设计

(下)

2015.11.30

isszym　sysu.edu.cn

数据库设计

- 关系数据库

数据库的类型有层次数据库、网状数据库、关系数据库、面向对象数据库（对象持久化）、时态数据库等等，关系数据库是目前最普遍使用的数据库。

关系数据库是面向关系的。以行(row)和列(column,field)来存储数据，行和列组成二维表（table），很多二维表又组成一个数据库。其中表和表之间存在一定的关系。要设计好一个关系数据库要求符合三个设计规范:

（1）每个表的一列只能包含一个数据，不能包含多个数据，例如，产品统计表的第一季度的生产量包含了产品的1~3月的生产量，这就不符合规范，因为一列包含了三个数据。

（2）每个表可以用一个或多个列值确定唯一的一行，例如，学生名册(表)用学号、姓名、年龄、班级四列定义，用学号就确定一个学生（一行）。一般可以为每个表设定一个可以自动增长的id用于区分每一行，也用作主键。

（3）不要有冗余数据。例如，成绩单采用学号、姓名、成绩三列就冗余了，因为姓名可以用学号从学生名册查到。 这种冗余容易引起不一致性。

- 数据库语言
  - 数据库语言包括数据库定义语言DDL(Data Definition Language)和数据操纵语言DML(Data Manipulation Language)。
  - 数据库系统中要定义很多对象，包括数据库(Database)、表(table)、域(field)、视图(view)、存储过程(stored procedure)、触发器(trigger)、索引(index)等。
  - 数据定义语言是SQL语言集中负责建立(create)、修改(alter)和删除(drop)数据库对象的语言。
  - 数据操纵语言实现对数据库的基本操作，例如，在表(table)中插入(insert)数据(row)、删除(delete)和修改(update)表中的数据(fields)、查询和显示表中的数据(fields)等。

- MySQL数据操作语句
  - 基本的Sql查询语句：

```
SELECT [tablename1 .]`fieldname1` AS aliasname1,                    ! 字段列表
       [tablename2 .] `fieldname2` AS aliasname2,
       …
       FROM tablename1 taliasname1,          ! 要查询的表(可以有多个)
            tablename2 taliasname2,
            …
          WHERE condition                                           ! 查询条件
          ORDER BY fieldnamei [asc/desc], …                         ! 排序字段
          GROUP BY fieldnamej,…                                     ! 分组查询统计
          LIMIT [offset,] rows                                      ! 选择一页
```

- 如果字段名中包含汉字或者空格，必须加上反引号。
- SELECT * FROM tablename 可以获取一个表的所有内容，* 表示所有字段。
- 在字段名前面加上表名或者表的别名可以用来区分同名字段。
- 选择一页表示从查询结果中选择第offset个记录开始的rows个记录。只有一项时，表示选取从第1页开始的rows个记录。
- 与表(table)在查询语句里是一样的，可以使用表(table)的地方也可以使用视图(view) 。

假设有两个表：学籍表student，选课表 selCourse。 student有两列：学号stuNum、姓名stuName、专业major。 选课表有两列：学号stuNum和课程号courseNum。

```
! 查询专业为softeng的所有同学，显示他们的学号和姓名。
SELECT  stuNum As num, stuName As name          ! 学号，姓名，num和name为别名
 FROM student
      WHERE major="softeng"
      ORDER BY stuNum asc    ! 结果用学号升序排列

! 查询所有选了课程编号为2015001的课程的同学，显示他们的学号和姓名。
SELECT  A.stuNum, A.stuName          ! 学号，姓名
         FROM student A, selCourse  B   ! 要查询的表(学籍，选课)，A B为别名
         WHERE B.courseNum="20150001"  and A.stuNum=B.stuNum
              ORDER BY A.stuNum asc    ! 结果用学号升序排列
              LIMIT 20,10              ! 从结果的第20行开始，选择10行
```

WHERE 表达式中的运算符：

= != > < >= <=	等于，不等于, ...
[NOT] BETWEEN … AND …	WHERE age BETWEEN 20 AND 30
[NOT] IN (项1,项2,…)	WHERE city IN('beijing','shanghai') WHERE id IN(SELECT id FROM ItemA)
[NOT] LIKE	WHERE username LIKE '%user' % 0或多个任意字符，_ 单个字符，正则表达式
IS NULL	空值判断符
IS NOT NULL	非空判断符
NOT、AND、OR	否、与、或

*GROUP BY的例子(按部门统计最高薪水和人数，并且只选取平均薪水大于3000的部门)：

```
SELECT DEPT, AVG( SALARY ) AS AVG, COUNT( * ) AS COUNT FROM staff
        GROUP BY  DEPT
        HAVING   AVG( SALARY ) >3000
        ORDER BY  DEPT
```

函数：SUM(), COUNT(), MAX(), MIN(), AVG()

➢组合查询*

*表示选看

! UNION可以把多个表的查询结果合并到一起

```
SELECT type,name FROM ItemA
    UNION SELECT stype,sname FROM ItemB
```

! 把第一个SELECT查询结果作为一个表：

```
SELECT A.name,B.sname
   FROM (SELECT * FROM ItemA WHERE name LIKE "%A") A, ItemB B
   WHERE A.id = B.id
```

ItemA

id	type	name
1	X	AAA
2	Y	BBB
3	Z	CCC
4	X	DAA

ItemB

id	stype	sname
1	X	111
2	U	222
3	X	333
4	Z	444

UNION

type	name
X	AAA
Y	BBB
Z	CCC
X	DAA
X	111
U	222
X	333
Z	444

FROM (SELECT ...)

name	sname
AAA	111
DAA	444

表ItemA 和ItemB在test数据库中定义：

* id字段为自动增长的int类型，其它字段都为varchar类型，长度为24。id是key。

! JOIN语句: 只要分别来自两个表的记录满足条件，就组成一行：

LEFT JOIN	左边表所有行都要加入，即使右边表没有匹配的。
RIGHT JOIN	右边表所有行都要加入，即使左边表没有匹配的。
INNER JOIN	两边都要存在
LEFT OUTER JOIN	同LEFT JOIN。RIGHT OUTER JOIN 同RIGHT JOIN

例：
```
SELECT A.name,B.sname
        FROM ItemA A LEFT JOIN ItemB B ON A.type = B.stype
SELECT A.name,B.sname
        FROM ItemA A RIGHT JOIN ItemB B ON A.type = B.stype
SELECT A.name,B.sname
        FROM ItemA A INNER JOIN ItemB B ON A.type = B.stype
```

ItemA

id	type	name
1	X	AAA
2	Y	BBB
3	Z	CCC
4	X	DAA

ItemB

id	stype	sname
1	X	111
2	U	222
3	X	333
4	Z	444

LEFT JOIN

name	sname
AAA	111
DAA	111
AAA	333
DAA	333
CCC	444
BBB	(Null)

RIGHT JOIN

name	sname
AAA	111
AAA	333
CCC	444
DAA	111
DAA	333
(Null)	222

INNER JOIN

name	sname
AAA	111
DAA	111
AAA	333
DAA	333
CCC	444

➢插入语句

```
INSERT INTO tableName(fieldName1, fieldName2,…)
                      VALUES(value1,value2,…);

INSERT INTO tableName(fieldName1, fieldName2,…)
                      (SELECT …);
```

例：
```
INSERT INTO ItemA(`type`,`name`)
     VALUES("G","EEE");
INSERT INTO ItemA(`type`,`name`)
     (SELECT stype,sname FROM ItemB);
```

➢修改语句

```
UPDATE tableName
    SET fieldName1=value1, fieldName2=value2,…
    WHERE condition;
```

例如：
```
UPDATE shape
    SET type ='circle'
    WHERE id=5;
```

➢删除语句

```
DELETE FROM tableName
        WHERE condition;
```

例如：

```
DELETE FROM shape
        WHERE id=14;
```

- Java的数据查询

```
// ShowItems.java
import java.sql.*;
public class ShowItems {
    static private Connection conn;
    static int cnt = 0;
    public static void main(String args[]){
        if(connect()){
            ResultSet rs = executeQuery("select * from ItemA;");
            showItems(rs);
        }
        else{
            System.out.println("Connect Error!");
        }
    }
    //建立连接
    private static boolean connect() {  ...}
    //执行SQL查询语句，返回结果集
    static private ResultSet executeQuery(String sqlSentence){...}
    //显示查询结果
    private static void showItems(ResultSet rs){...}
}
```

```
//建立连接
private static boolean connect() {
    String connectString= "jdbc:mysql://localhost:3306/test"+
          "?autoReconnect=true&useUnicode=true&characterEncoding=UTF-8";
   try {                                  类库名
        Class.forName("com.mysql.jdbc.Driver");
        conn = DriverManager.getConnection(connectString, "root", "123456");
        return true;
   }                                                    用户名  密码
   catch (Exception e) {
        System.out.println(e.getMessage());
   }
   return false;
}
```

- connectString中的test为数据库名。
- Class.forName用于查找连接类， com.mysql.jdbc.Driver为类库名。
- root和123456分别为登录数据库的用户名和密码。

```java
//执行SQL查询语句，返回结果集
static private ResultSet executeQuery(String sqlSentence) {
     Statement stat;
     ResultSet rs = null;

     try {
        stat = conn.createStatement();         //获取执行sql语句的对象
        rs = stat.executeQuery(sqlSentence); //执行sql查询，返回结果集
     } catch (Exception e) {
        System.out.println(e.getMessage());
     }
     return rs;
}
```

```
createStatement(int type, int concurrency);

  参数 int type:
      ResultSet.TYPE_FORWORD_ONLY          结果集的游标只能向下滚动。 (默认)
      ResultSet.TYPE_SCROLL_INSENSITIVE    结果集的游标可以上下移动，当数据库变化时，当前结果集不变。
      ResultSet.TYPE_SCROLL_SENSITIVE      返回可滚动的结果集，当数据库变化时，当前结果集同步改变。
  参数 int concurrency
      ResultSet.CONCUR_READ_ONLY           不能用结果集更新数据库中的表。 (默认)
      ResultSet.CONCUR_UPDATETABLE         能用结果集更新数据库中的表。
```

```
//显示查询结果
private static void showItems(ResultSet rs){
    try {
       while(rs.next()){
          System.out.println(rs.getString("name"));

       }
    }
    catch (Exception e) {
       System.out.println(e.getMessage());
    }
}
```

- 游标(cursor)是指向当前行(row)的指针 。
- rs.next()将游标移动到下一行。第一次执行该方法时游标会移动到第一行。如果游标已经处于最后一行记录时执行该方法会返回false。
- rs.getString(“type”)返回域type的值。

ResultSet其它方法:

public void rs.previous();	将游标移动到前一行(row)
public void rs.first();	将游标移动到第一行
public void rs.next();	将游标移动到下一行，如果到了末尾返回FALSE
public void rs.last();	将游标移动到最后一行
public void rs.afterLast();	将游标移动到最后一行之后
public void rs.beforeFirst();	将游标移动到第一行之前
public boolean isAfterLast()	判断游标是否在最后一行之后。
public boolean isBeforeFirst()	判断游标是否在第一行之前。
public boolean ifFirst()	判断游标是否指向结果集的第一行。
public boolean isLast()	判断游标是否指向结果集的最后一行。
rs.getRow();	得到当前行的行号(从1开始)，如果结果集没有行，返回0。
rs.getInt(fieldName);	取出当前行指定字段的整数值。
rs.getDate(fieldName);	取出当前行指定字段的日期值。
rs.getBoolean(fieldName);	取出当前行指定字段的布尔值。
rs.getFloat(fieldName);	取出当前行指定字段的浮点值。
public boolean absolute(int row)	将游标移到参数row指定的行号。如果row取负值，就是倒数的行数，absolute(-1)表示移到最后一行，absolute(-2)表示移到倒数第2行。当移动到第一行的前面或最后一行的后面时，该方法返回false

- Java的数据修改

```
// UpdateItems.java
import java.sql.*;
public class UpdateItems {
    static private Connection conn;
    static int cnt = 0;
    public static void main(String args[]){
        if(connect()){
                updateItems("update ItemA set type ='X' where id=3;");
        }
        else{
                System.out.println("Connect Error!");
        }
    }
   //建立连接
   private static boolean connect() {...} // 与数据查询的connect相同

   //执行SQL修改语句，返回结果集
   private static boolean executeUpdate(String sqlSentence) {...}

   //进行修改
   private static void updateItems(String sqlSentence){...}
}
```

```java
//执行SQL修改语句, 返回结果集
private static boolean executeUpdate(String sqlSentence) {
    Statement stat;

    try {
      stat = conn.createStatement();          // 根据连接获取一个执行sql语句的对象
      cnt = stat.executeUpdate(sqlSentence); //执行sql语句,返回所影响行记录的个数
    }
    catch (Exception e) {
         System.out.println(e.getMessage());
    }
    return (cnt>=0);
}

//进行修改
private static void updateItems(String sqlSentence){
    if(executeUpdate(sqlSentence)){
       System.out.println(""+cnt + " records are updated.");
    }
}
```

执行executeQuery()可以进行插入、修改和删除行的操作。

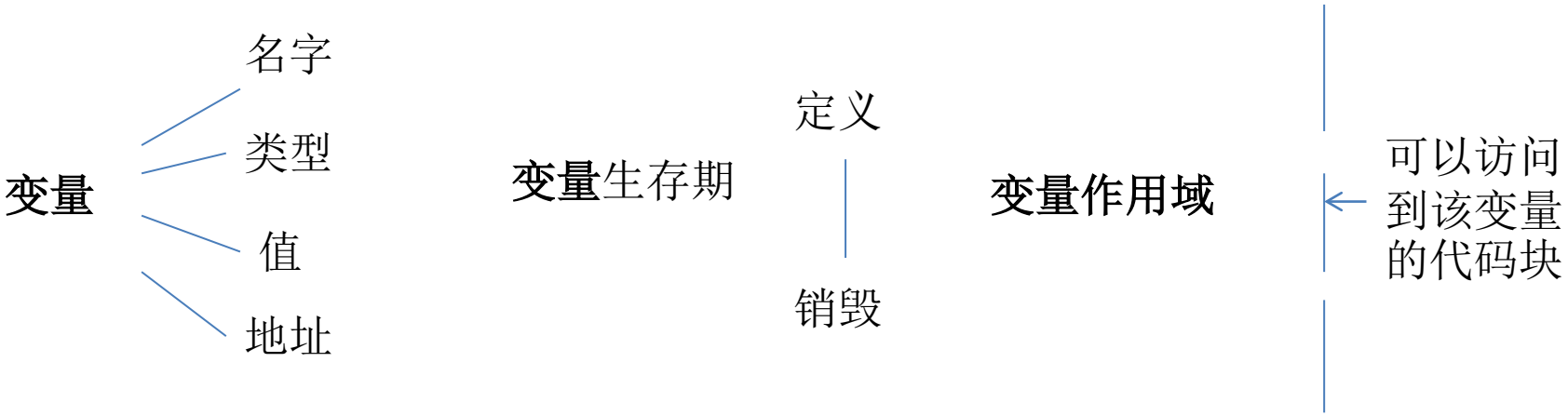
- 调用存储过程

存储过程sumtwo(int, int,int)把前两个参数相加，第三个参数为返回值(传址参数)。调用该存储过程的方法如下：

```
 //
CallableStatement cs
     =conn.prepareCall("{call sumtwo(?,?,?)}"); //指出存储过程名和参数

cs.setInt(1,50);                          // 将第一个参数的值设置成50
cs.setInt(1,150);                         // 将第二个参数的值设置成150
cs.execute();                             // 执行存储过程
System.out.println (cs.getInt(3));        // 显示返回结果
```

Java变量的作用域和生存期



- **变量**具有名字、类型、地址和值四个部分。变量的类型指出值的类型，变量的地址指出值存放的地址。**对象变量的值**为一个指针，指向对象的存放位置。
- 何时可以对变量赋值和引用它取决于变量的生存期和作用域。**变量的作用域**是指可以访问到该变量的代码范围。**变量的生存期**是指变量占用内存的时间，在一个变量的生存期外即使在作用域内也不能访问该变量。

```
class  MyMath {
  String msg;
  int sum(int n){
    int res=0;
    for(int i=1;i<n;i++){
       n+=i;
    }
    return res;
  }
}
```

局部变量i的作用域：定义它的块

局部变量i的生存期：只在方法被调用的时候存在

- 一个方法的**非静态局部变量**只在该方法被调用时存在，一旦退出方法，非静态局部变量所占空间将被释放。**静态局部变量的生存期**为从第一次调用其所在方法直到程序结束。
- **所有局部变量的作用域**都是定义它的块(由{}括住)。同名的局部变量不能在某个块及其子块同时定义（C语言可以，只使用最近定义的）。
- **非静态成员变量(数据域)的生存期**与对象相同。**静态成员变量的生存期**从第一次使用该变量直到程序结束。
- **非静态和静态成员变量的作用域**由其访问权限确定。静态成员变量和方法可以作为C++中的全局变量和函数使用。

静态设置

- 类的方法和成员变量可以定义为静态的，方法内部的局部变量也可以定义为静态的。它们在第一次被访问时分配内存空间，然后一直保留到程序结束才释放。
- 静态变量在堆中分配空间。静态方法和静态成员变量可以直接通过类名直接访问而不需要建立对象。
- 静态成员变量和静态局部变量在堆中所分配的空间，可以被所有处于其作用域的方法所使用。因此，静态方法和静态成员变量可以作为C++语言的全局函数和变量使用。
- main()就是静态方法，它也不用预先建立对象就可以让系统直接调用。
- 同一个类的非静态方法不能被直接使用。在不建立对象时，静态方法只能使用静态方法和静态数据域。只有建立对象才能使用非静态数据域和非静态方法。
- import语句后加入static可以令该包的类的所有方法变为static。

  import static com.group.show.*;

ClassA.java

```
class ClassA {
    static  int  x1 =0;
    static String s1;
    {
        ClassA.s1 = “abcd”;  //不能用于静态数据域的初始化
     }
     static void f1(){
        System.out.println("Hello!");
     }
}
class TestStatic {
     public static void main(String[] args){
          ClassA.f1();
          System.out.println(ClassA.x1);
          System.out.println(ClassA.s1);
     }
}
```

显示：
Hello
0
null

final设置

- 在Java中，如果成员变量（数据域）、局部变量或者参数加上final的修饰词，表示初始化后不能被修改。
- 定义为final的成员变量的初始化可以在定义之后进行，但是只能初始化一次。
- 定义为final的方法不能被导出类所覆盖(override)。
- 定义为final的类不能被继承。

```
private static Random rand = new Random(47);
static final int INT_5 = rand.nextInt(20);    //0-19
```

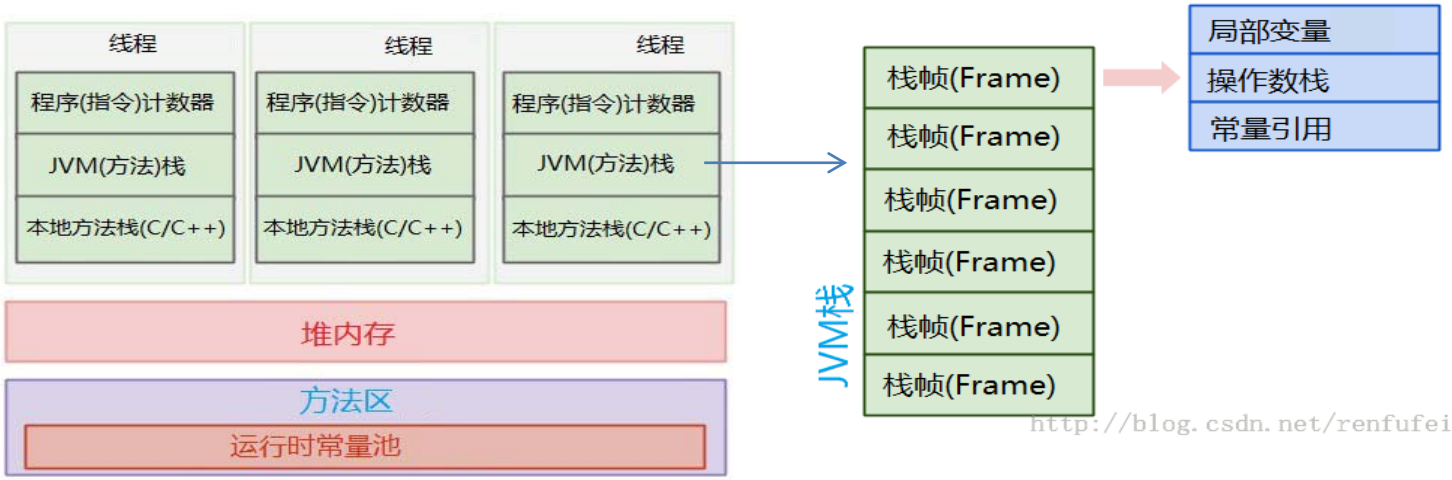
变量和函数的内存分配

栈(Stack)区：每个线程都PC(程序计数器),JVM(方法)栈,以及本地方法栈(调用C语言等的动态链接库用)。每个栈帧存放用于一个方法的非静态局部变量、方法参数、方法返回值、操作数栈（计算表达式）、常量引用。对于基本数据类型变量，直接存放值；对于对象变量，只存放对象引用 。由于编译器可以直接确定局部变量等的相对地址(相对栈帧基址)，所以访问速度很快。

堆(heap)区用于存放(new)数组和对象。堆中对象当没有任何引用时，其空间通过垃圾回收进行释放。每一个Java应用都唯一对应一个JVM实例，每一个实例唯一对应一个堆。

方法区包含静态数据区、常量池和代码区。常量池存储了常量、类有关的信息。代码区保存方法和构造器的代码。常量以及静态变量的引用直接放在程序的指令或栈帧中。

* 定义对象变量只是定义了对象的引用（在栈中）只有new之后才在堆中为对象分配内存。



常量池的常量分为字面量和引用量。文本字符串、final变量等都是字面量，类信息、接口信息、数据段信息、方法信息都属于引用量。引用量最常见的一种用法是在调用方法的时候，根据方法名找到方法的引用，并以此定位到函数体进进行函数代码的执行。

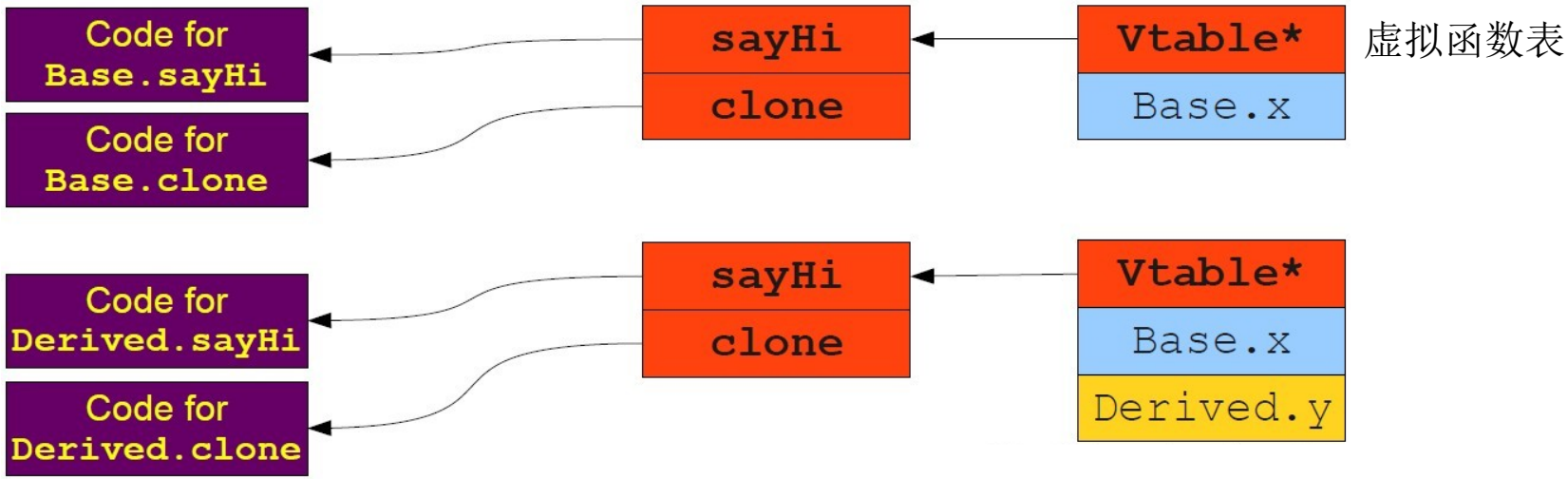
JVM把常量池按数据类型存放常量，并通过索引访问它们的入口。常量池在编译期间就被确定，并被保存在已编译的.class文件中。Java编译器会自动对常量进行优化，会查找并使用常量池已有的常量。见下例：

```
String str1 = new String("abc");
String str2 = new String("abc");
String str3 = "abc";
String str4 = "abc";
String str5 = "ab"+"c";
System.out.println(str1 == str2);      //false
System.out.println(str1 == str3);      //false
System.out.println(str3 == str4);      //true
System.out.println(str4 == str5);      //true
```

新建对象时JVM要在常量池中找到对象信息，然后在堆中建立对象，再把地址填入对象变量中。**Java对象在内存中是怎样分配的呢?** 一旦对象在堆中分配了空间,那本质上就是一系列的字节. 那么如何找到对象中某个特定的属性域呢? 编译器通过一个内部表来保存每个域的偏移量。

```
class Base {
    int x;
    void sayHi() {
        Print("Base");
    }
    Base clone() {
        return new Base;
    }
}
```

```
class Derived extends Base {
    int y;
    void sayHi() {
        Print("Derived");
    }
    Derived clone() {
        return new Derived;
    }
}
```



方法参数与重载

- Java的方法可以采用数组和对象作为参数，也用可变参数类型。数组和对象也可以作为返回类型。
- 因为Java取消了指针类型，所以Java只有传值参数没有传址参数。采用基本数据类型和字符串的参数不能作为传址参数（也称为引用参数），即不能带回返回值。如果需要带回参数值，只能用自定义类作为参数类型，因为这种参数传入的是对象的引用。
- 一个类中定义多个同名方法的做法叫重载(overload)。这些方法主要通过参数进行区分，包括参数个数、类型和顺序。不能使用返回值的类型进行重载。
- 子类也可以重载父类的方法。如果参数完全相同，则会覆盖(override)基类的方法，使父类的同名方法不能使用(可以通过super.method()调用)。
- 在覆盖方法前加上@Override，编译器会根据这个指示进行检查是否父类有这个相同的方法。

可变参数

如果希望方法带入不同个数的参数，就可以使用可变参数。

```
class Example{
  static void print(String...  args){
    System.out.println("***"+args.length);//参数个数
    for(String temp:args)
       System.out.println(temp);
   }
   public static void main(String[] args){
       print();
       print("1","2");
       print("1","2","3");
       String a[]={"a","b","c","d"};
       print(a);
   }
}
```

结果：

```
***0
***2
1
2
***3
1
2
3
***4
a
b
c
d
```

可变参数只能作为最后一个参数使用。args[i]可以取出第i个可变参数，用args.length可以得到可变参数的个数。

http://blog.csdn.net/testcs_dn/article/details/38920323

抽象类和接口

- 只要一个类有一个抽象方法(可以是继承来的)，则该类要定义为抽象类。抽象方法只有声明没有方法体。不能采用抽象类定义对象。

//ShapeAbs.java

```
public abstract class ShapeAbs {                    // 抽象类
    String color;                                   // 变量或属性(field)
    public ShapeAbs() {                             // 构造函数(constructor)
        System.out.println("Shape Initialized!");
        color = "black";
    }

    public abstract void draw();                    // 抽象函数或方法(method)

    public void setColor(String color) {            //方法: 设置颜色
        this.color = color;                         //this.color表示本类的属性
    }

    public String getColor() {                      //方法: 取出颜色
        return this.color;
    }

}
```

//CircleA.java

```
public class CircleA extends ShapeAbs {
    public CircleA() {                  // 构造函数(constructor)
        System.out.println("CircleA Initialized!");
    }

    public void draw() {                // 定义方法draw()
        System.out.println("CircleA draw() is called!");
    }
}
```

//RectangleA.java

```
public class RectangleA extends ShapeAbs {
    public RectangleA() {
        System.out.println("RectangleA Initialized!");
    }

    public void draw() {
        System.out.println("RectangleA draw() is called!");
    }
}
```

//ShapeInheritA.java

```
public class ShapeInheritA {
    public static void main(String args[]){
        CircleA circle = new CircleA();
        circle.draw();

        RectangleA rectangle = new RectangleA();
        rectangle.draw();
    }
}
```

执行结果：

```
Shape Initialized!
CircleA Initialized!
CircleA draw() is called!
Shape Initialized!
RectangleA Initialized!
RectangleA draw() is called!
```

接口是更加抽象的一种类，接口的方法都只有声明而没有方法体。
接口不能定义成员变量，但是可以定义常量。

//Door.java

```
public interface Door {
    void open();   // 开门
    void close();  // 关门
}
```

//Ordinary.java

```
public class OrdinaryDoor implements Door {
    public void open(){
        System.out.println("open door!");
    }

    public void close(){
        System.out.println("close door!");
    }
}
```

// Alarm.java

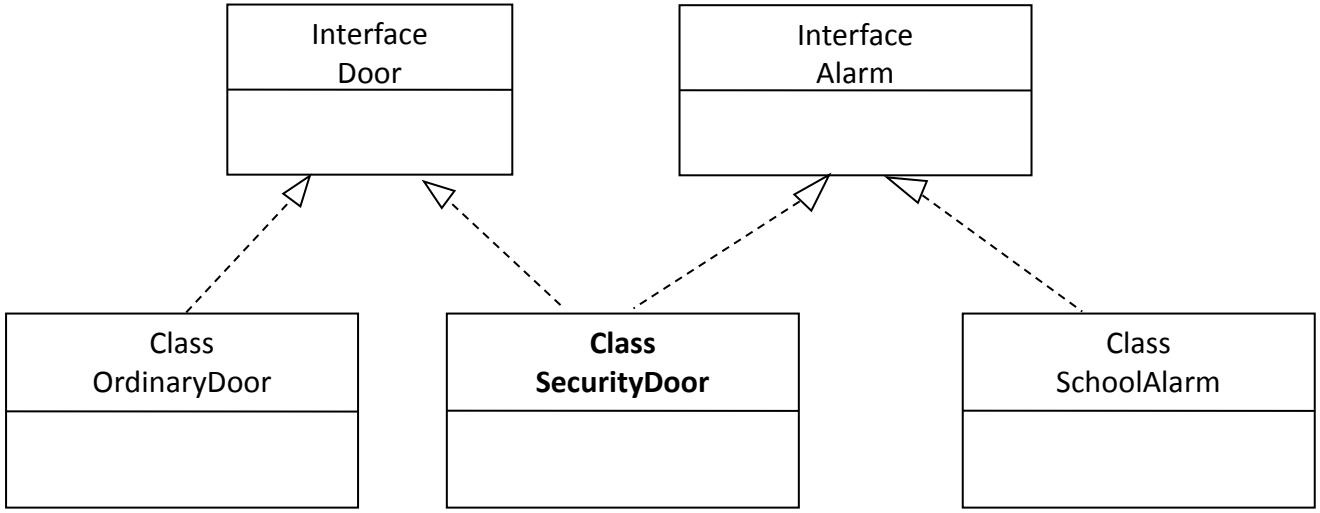
```
public interface Alarm {
    void alarm();    // 拉响警报
}
```

// SchoolAlarm.java

```
public class SchoolAlarm implements Alarm {
    public void alarm(){
        System.out.println("alarm!");
    }
}
```

还可以定义其它各种门的类，其它程序只通过接口提供的方法使用接口的子类提供的功能，这样即使子类则更加或发生变化，只要接口不变，使用该接口的程序不用改变，这是使用接口的本来目的。

如果要实现一个安全门，就要用到多重继承。和C++不同，Java没有多重继承，不过多重继承通过实现多个接口来实现。



// Security.java

```
public class SecurityDoor implements Door,Alarm {
    public void open(){
        System.out.println("open door!");
    }

    public void close(){
        System.out.println("close door!");
    }

    public void alarm(){
        System.out.println("alarm!");
    }
}
```

// 一个类除了可以实现多个接口，还可以同时继承一个基类。Java没有多重继承，例如：下面ClassA继承ClassB，实现InterfaceA和InterfaceB。

```
Class ClassA extends ClassB implments InterfaceA, InterfaceB  {


}
```

InterfaceTest.java

```
public class InterfaceTest {
    public static void main(String args[]){
        Door doors[] = new Door[10];
        Alarm alarms[] = new Alarm[10];

        SecurityDoor door1 = new SecurityDoor();
        doors[0] = door1;
        alarms[0] = door1;
        OrdinaryDoor door2 = new OrdinaryDoor();
        doors[1] = door2;
        SecurityDoor door3 = new SecurityDoor();
        doors[2] = door3;
        alarms[1] = door3;
        SchoolAlarm alarm1 = new SchoolAlarm();
        alarms[2] = alarm1;

        for(Door door:doors){      //统一处理所有的door
            if(door!=null){
              door.open();
            }
        }
        for(Alarm alarm:alarms){   //统一处理所有的alarm
            if(alarm!=null){
              alarm.alarm();
            }
        }

    }
}
```

执行结果：

```
open door!
open door!
open door!
alarm!
alarm!
alarm!
```

Door door = new SecuriityDoor();
door.open(); //新的对象不会影响后面的程序。
door.close();

采用继承的方法可以实现更大的接口

```
public interface SecurityDoorInterface extends Door,Alarm {
    public void radio();          // 遥控：开门，关门，响铃
}
```

//除了继承Door和Alarm的方法，还增加了新方法radio()。

内部类

在类的内部定义的类为内部类。内部类可以被内部方法所使用，并对外部实现了隐藏。在内部类中可以直接使用其它外部类。

```
//InClass.java
class InClass {
      int x=0;
      class B {
         void f1(){
           x = 5;
         }
      }
      B getB(){
         return new B();
      }
      public static void main(String[] args){
          InClass a1= new InClass();
          System.out.println(""+a1.x); //输出0
          B b1 = a1.getB();
          b1.f1();
          System.out.println(""+a1.x);  //输出5
      }
}
```

try和throws

(CatchError.java)

```
public class CatchError {
    public static void main(String args[]){      // main为主程序入口
        int x;
        try {                                    // 打开例外处理语句
            for (int i = 5; i >= -2; i--) {
                x = 12 / i;                      // 出现例外后将不执行后面的语句
                System.out.println("x=" + x);    // 直接跳到catch内执行
            }
        }
        catch (Exception e) {                    // 捕捉例外信息。可以并列用多个catch
            System.out.println(e.getMessage()); // 显示当前错误信息
            // e.printStackTrace();              // 显示系统错误信息
        }
        finally{                                 // 出现例外必须执行这里的语句
            x=0;
        }
        System.out.println(x);

    }
}
```

执行结果：

```
x=2
x=3
x=4
x=6
x=12
Error:/ by zero
0
```

如果一个方法不愿意处理一些例外，可以采用throws定义方法的例外，把这些例外交给调用者去处理。

(DivideClass.java)

```
public class DivideClass {
    void divide() throws Exception {     // 出错后交给调用程序处理
        for (int i = 5; i >= -2; i--) {
            int x = 12 / i;              // 出现例外(x==0)后将不执行后面的语句
            System.out.println("x=" + x);
        }
    }
}
```

执行结果：

```
x=2
x=3
x=4
x=6
x=12
Error:/ by zero
```

(ThrowError.java)

```
public class ThrowError  {
    public static void main(String args[]){     // main为主程序入口
        try {                                    // 打开例外处理语句
           DivideClass div = new DivideClass();
           div.divide();
        }
        catch (Exception e) {                    // 捕捉例外信息。可以并列用多个catch
            System.out.println("Error:"+e.getMessage()); // 显示当前错误信息
        }
    }
}
```

构造器

- 构造器(constructor)主要用于初始化对象中的成员变量。每个类可以定义一个或多个构造器。构造器可以带或不带参数。如果一个类没有定义构造器，Java系统会为它生成一个默认的构造器。
- 如果定义了构造器，则只能使用这些构造器，而不能使用默认构造器。
- 一个类的构造器的名字要与其类名完全相同。 构造器默认的访问类型为public，如果定义为private，则不能在外部直接用new建立对象，而要使用静态方法建立对象。
- 构造器也可以重载，此时要根据新建对象带入的参数选择构造器。
- 当建立导出类对象时，要先调用基类构造器。如果基类没有构造器，则会自动调用默认构造器。如果基类构造器没有参数，也会自动调用该构造函数。如果基类构造器有参数，则不会调用基类的构造器。此时需要在导出类的构造器中用super(参数)调用基类的构造器。

```
class BaseA {
        int x1;
        BaseA(){
       x1=5;
        }
        BaseA(int y){
           x1=y;
        }
}
class DerivedA extends BaseA{
        String s1;
        DerivedA(String s2,int y){
                super(y);
                this.s1="'"+s2+"'";
        }
}
class TestStatic {
     public static void main(String[] args){
        DerivedA o1= new DerivedA("",12);
        System.out.println(o1.s1+o1.x1);

     }
}
```

垃圾回收

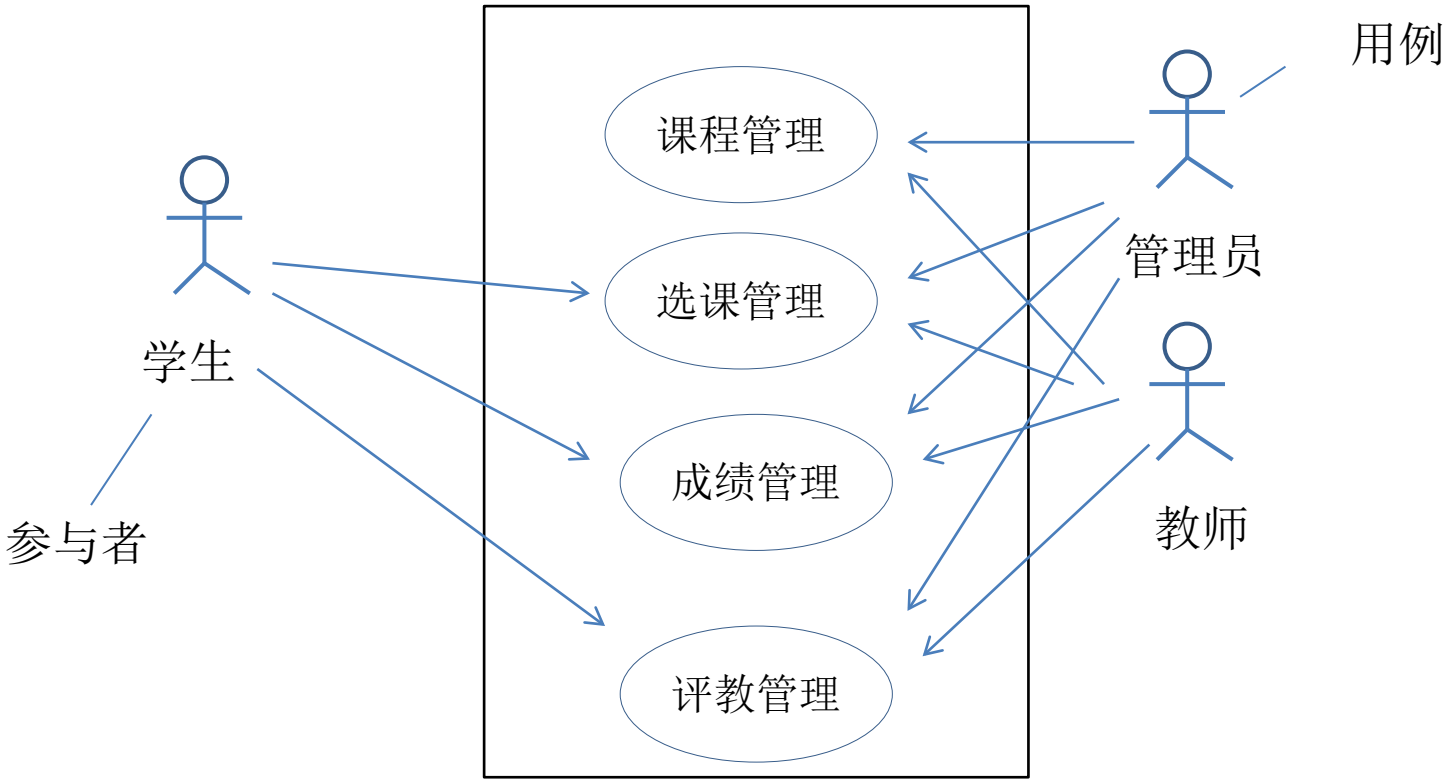
- Java语言没有析构器(destructor)，也不能主动销毁对象，所有对象都由垃圾回收器自动进行回收。
- 垃圾回收器只在内存缺乏时才会去销毁那些没有任何引用的对象，并触发对象的finalize事件。
- 回收工作也许在程序结束都不会发生。如果需要在对象使用结束前做一下清理工作，例如，关闭文件，可以在对象中专门定义一个dispose方法来处理。

UML

统一建模语言(Unified Modeling Language，UML) 是一个支持整个软件系统开发过程的建模语言。其基本模型包括：

用例图：表达系统外部的执行者与系统用例之间的关系。
类图：展示系统中类的静态结构
对象图：一种实例化类图
包图： 表示包与包之间的关系
状态图：描述一类对象具有的所有可能的状态及其转移关系
时序图/顺序图：展示对象之间随时间推移交换消息的过程
合作图：展示对象间动态协作关系，突出消息收发关系
活动图：展示系统中各种活动的执行流程和执行顺序
构件图：展示程序代码的组织结构和各种构件之间的依赖关系
配置图：展示软件在硬件环境中(特别是在分布式及网络环境中)的配置关系

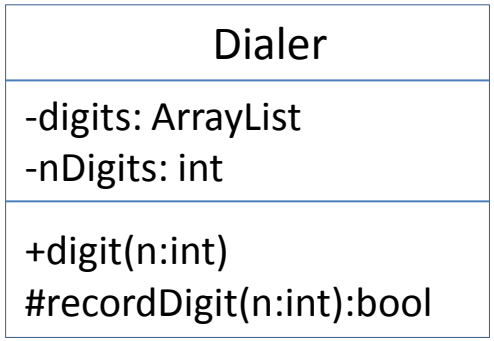
由参与者（Actor）、用例（Use Case）以及它们之间的关系构成的用于描述系统功能的静态视图称为**用例**图。



类的表示

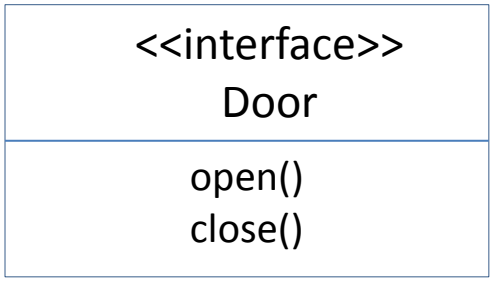

```
public class Dialer{

}
```

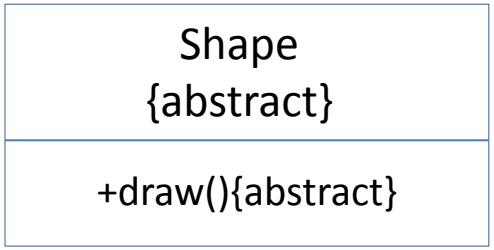



```
public class Dialer{
    private ArrayList digits;
    private int nDigits;
    public void digit(int n){...};
    protected bool recordDigit(int n){}...;
}
```

- private
+public
#protected

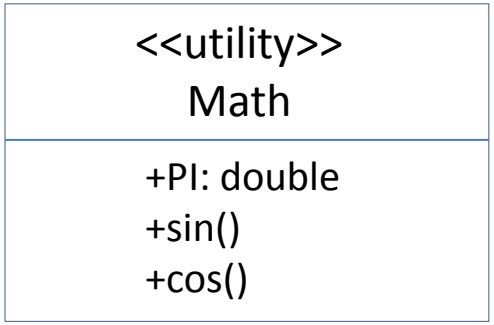



```
public interface Door {
   void open();
   void close();
}
```




```
public abstract class Shape {
   public void draw();
}
```

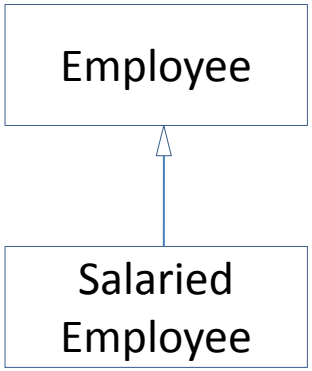
其它表示法：把名字写成斜体或在框上加(A)




```
public class Math{
   public static double PI=3.1415926
   public static double sin(double theta){...};
   public static double cos(double theta){...};
}
```

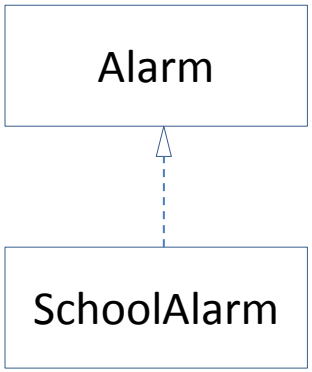
工具类：全部采用静态变量和静态方法

继承

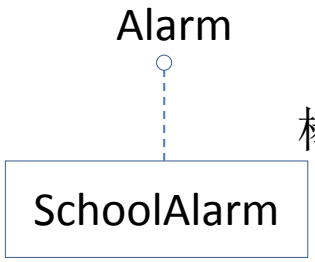



```
public class Employee{
   ...
}

public class SalariedEmployee  extends Employee {
   ...
}
```




```
public interface Alarm{
   ...
}
public class SchoolAlarm  implements  Alarm {
   ...
}
```

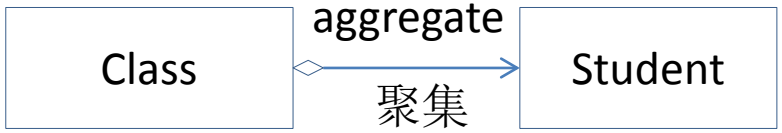


棒棒糖形状表示接口

关联

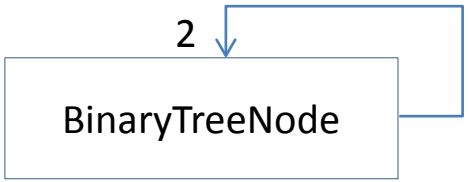

```
public class Car{
   private Wheel[4] wheels;
}
```


```
public class Phonebook{
   private ArrayList itsPnos;
}
```



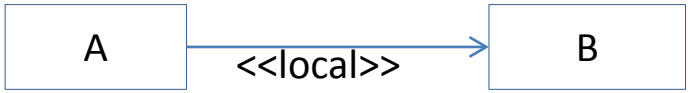

```
public class Class{
   private Student[4] students;
}
```


```
public class House{
   private Door itsDoor;
}
```




```
public class BinaryTreeNode{
   private BinaryTreeNode leftNode;
   private BinaryTreeNode rightNode;
}
```


```
public class A{
   private B makeB(){
     return new B();
   }
}
```

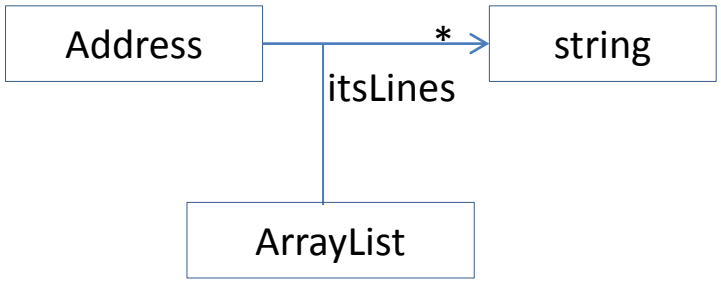



```
public class A{
   private void F(){
      B b=new B();
   }
}
```

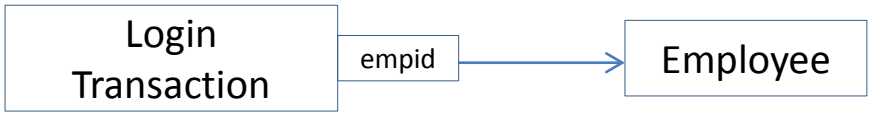

```
public class A{
   public void F(B b){
   }
}
```


```
public class A{
   private B itsB;
   public void F(){
      itsB.F();
   }
}
```


```
 public class A{
    private class B {
       ...
    }
 }
```

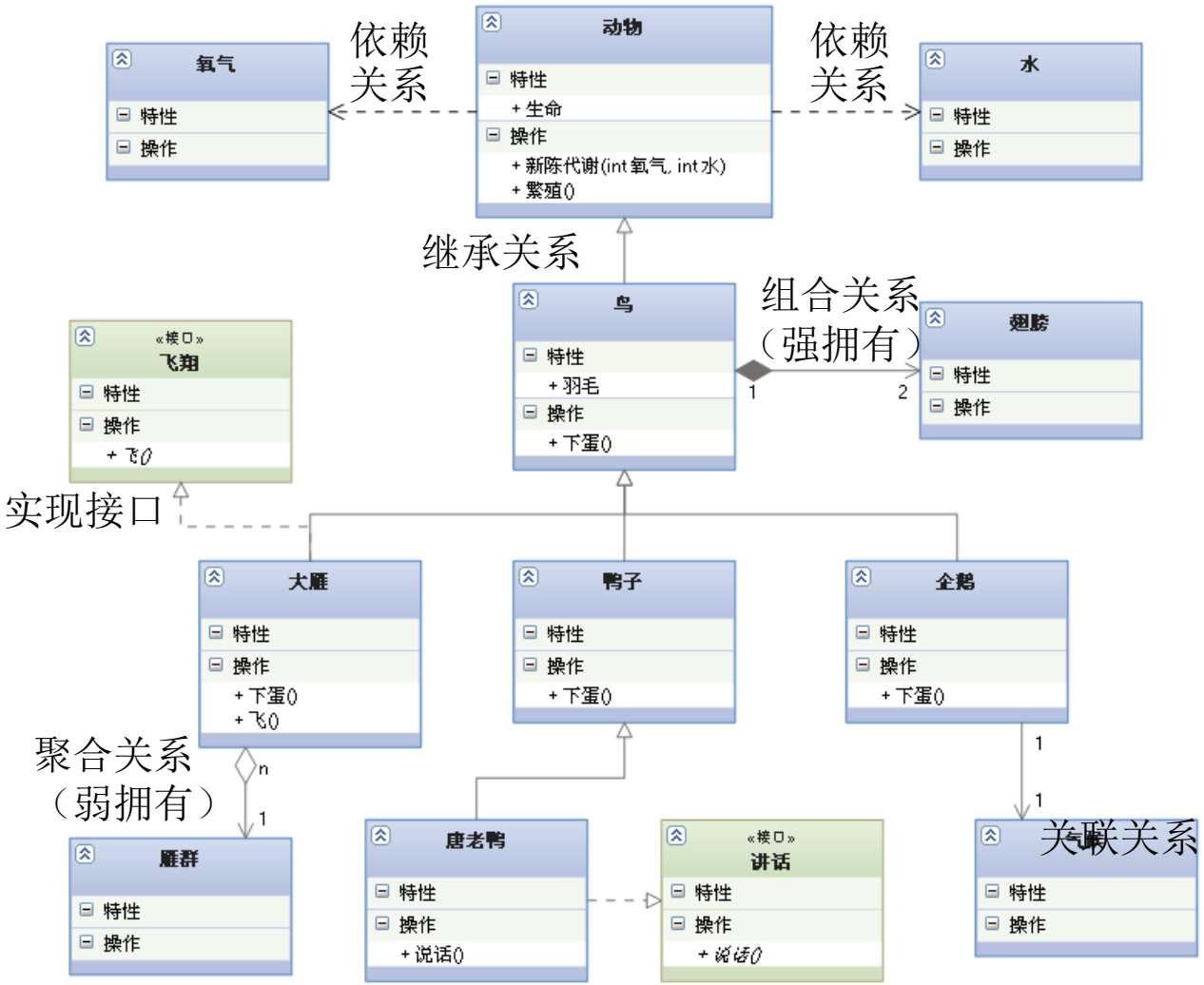



```
 public class Adress{
    private ArrayList itsLines;
 }
```



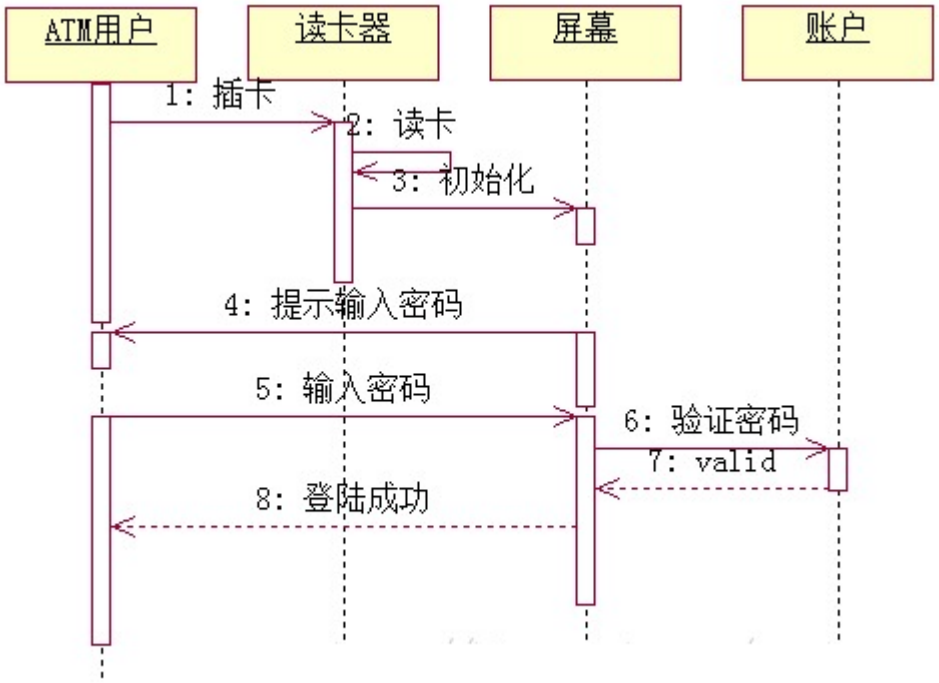

```
   public class LoginTransaction {
      private String empid
      private ArrayList itsLines;
   }
```

类图


氧气
特性
操作
依赖关系
动物
特性
+ 生命
操作
+ 新陈代谢(int 氧气, int 水)
+ 繁殖()
依赖关系
水
特性
操作
继承关系
«接口»
飞翔
特性
操作
+ 飞()
鸟
特性
+ 羽毛
操作
+ 下蛋()
组合关系
（强拥有）
1
2
翅膀
特性
操作
实现接口
大雁
特性
操作
+ 下蛋()
+ 飞()
鸭子
特性
操作
+ 下蛋()
企鹅
特性
操作
+ 下蛋()
聚合关系
（弱拥有）
n
1
雁群
特性
操作
唐老鸭
特性
操作
+ 说话()
«接口»
讲话
特性
操作
+ 说话()
1
1
关联关系
特性
操作

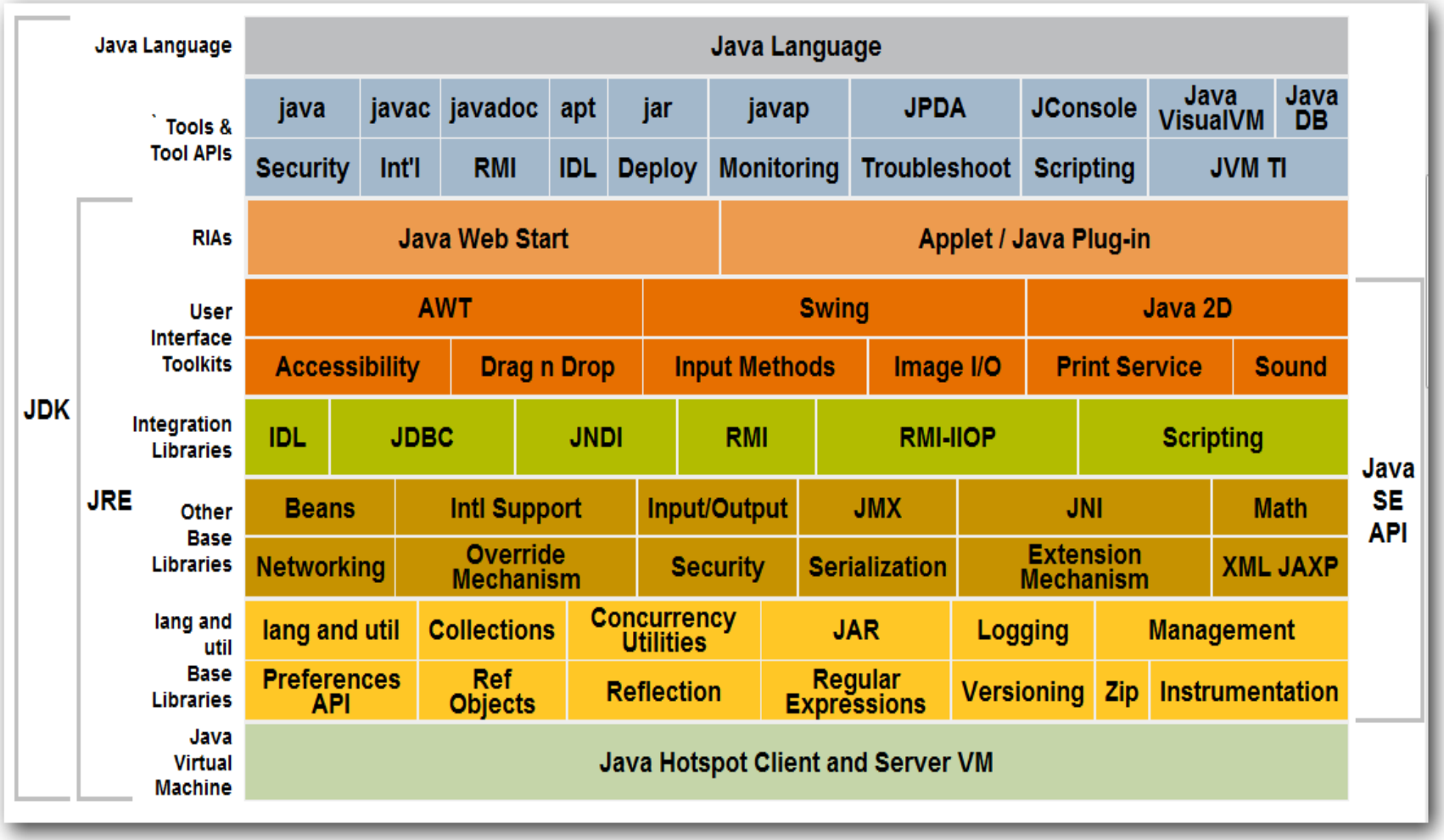
顺序图

顺序图描述了对象之间传送消息的时间顺序，用来表示用例中的行为顺序。当执行用例时，顺序图中的每条消息对应了一个类操作或者引起转换的触发事件。



附录1、Java平台的组件

- 下面的概念图展示了Java平台的所有组件以及其它们如何组合起来的。



- JDK的主要程序

javac	把Java源代码文件(.java)编译为Java字节码文件(.class) 。 $javac ShowPhoto.java
java	用于在Java虚拟机上执行Java字节码文件。$java ShowPhoto
appletviewer	用来执行HTML文件中的Java小程序代码。
javadoc	根据Java源程序中的说明语句生成HTML文档。
javap	用来显示字节码文件中字节码含义以及可访问方法和数据。
jar	Java用于发布的打包和解压软件(ZIP格式)。

直接执行命令java和javac可以看出是否了Java虚拟机和JDK。

```
$jar  cvf  ShowPhoto  ShowPhoto.class  pictureDir  videoDir  //压缩
$jar  tvf  ShowPhoto.jar                                    //显示
$jar  xf  ShowPhoto.jar                                     //解压
```

附录2、安装MySQL

- 安装数据库系统

文件名：mysql-installer-community-5.6.26.0.msi

下载地址： http://dev.mysql.com/downloads/mysql/

安装图示：http://blog.csdn.net/longyuhome/article/details/7913375

* 安装和配置一般选默认值，超级用户root的密码(假设为123456)一定要记住。
* 在Enable root access from remote machines处打勾

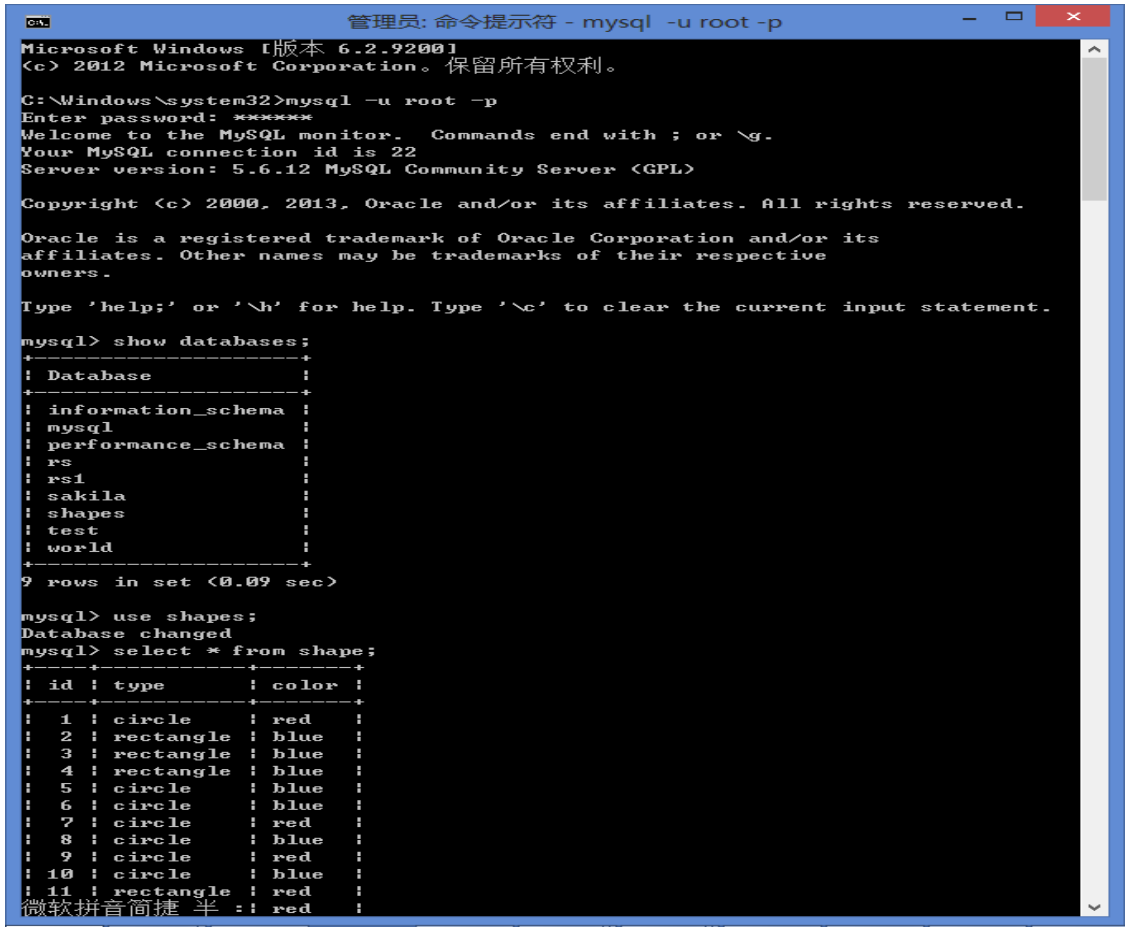
安装完毕后，可以先试一下用命令行登录（要设置路径）：

```
C:>mysql –u root –p
password:123456
```

进入后就可以操作数据库了。默认就有test数据库，可以执行命令：

```
mysql>show databases;
mysql>use test;
mysql>show tables;
```

>quit 退出



如果需要远程访问该数据库系统，则需要执行：

mysql>GRANT ALL PRIVILEGES ON *.* TO 'root'@'%'

IDENTIFIED BY 'password' WITH GRANT OPTION;

其中，password替换为上面所设置的密码123456

- 安装数据库管理软件

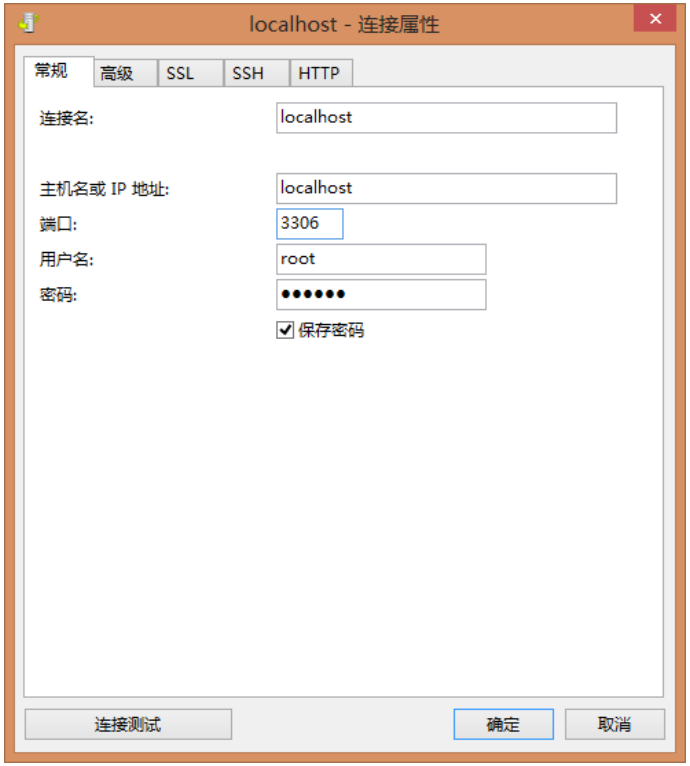
文件名： Navicat_Premium_11.0.10

其它图形化管理工具： http://www.chinaz.com/free/2009/0306/68691.shtml

安装完毕后便可以管理数据库了。可以建数据库，建表，输入数据等。还可以利用查询功能测试SQL语句。可以同时远程管理多个MySql数据库系统。

远程管理需要MySql的设置，并把localhost也改为远程域名或IP地址。注意：端口号要与MySql一致（默认为3306）。

新建数据库的字符集选择UTF-8 Unicode、排序规则选择utf8_general_ci。



*安装MySql时会安装管理系统mysql-workbench-community-6.3.4-winx64.msi，如果不用Navicat，也可以直接用这个。

- 下载Java连接类库

  文件名：mysql-connector-java-gpl-5.1.36.msi
  下载地址： http://dev.mysql.com/downloads/connector/j/

  * 安装MySql时会同时下载

- 在Java中连接MySql

  - 如果通过**Eclipse**运行Java，则需要引入该jar文件:
    右键点击项目名，进入菜单Build Path/Add Liraries.../User Library/next /User Libraries.../New 取名后再Add JARs... 找到该jar文件加入。
    如果已经存在该jar文件的命名路径(所有Java和JSP项目共用)，可以直接在User Library时选择。
  - 如果在命令行运行Java，需要在classpath中加入mysql-connector-java-5.1.36-bin.jar文件的路径。

参考资料

- Bruce Eckel，Java编程思想（第4版），机械工业出版社，2012
- Cay S.Horstmann, G.Cornell, Java核心技术，机械工业出版社，2012
- Y. Daniel Liang，Java语言程序设计（第8版），机械工业出版社，2011
- http://docs.oracle.com/javase/8/docs/
- http://api.apkbus.com/reference/java/io/package-summary.html
- http://docs.oracle.com/javase/tutorial/