This关键字

1 this调用本类属性

this可以算是比较复杂的关键字，因为this的使用形式上决定他的灵活性，在程序里面，使用this可以使用三类结构描述：

当强类中的属性：this属性
当前类中的方法（普通方法、构造方法）this()、 this.方法名称()
描述当前对象

使用this调用当前属性
利用方法或者是setter方法都可以进行类中的属性赋值，但是在进行赋值的时候，之前可能采用的是如下的定义形式：

```
class Person {
    private String name ;
    private int age ;
    public Person(String n,int a) {
        name = n ;
        age = a ;
    }
    public void tell() {
        System.out.println("姓名：" + name + "年龄：" + age) ;
    }
    //setter | getter略

}
public class JavaStudy {
    public static void main(String args[]) {
        Person per = new Person("王五",38) ;
        per.tell() ;
    }
}
```

有个问题出现在了参数名称上，可以发现此时构造方法中的两个目的蚕食是为了类中的name或age属性初始化，但是发现此时代码n和a参数名称不好

```
public Person(String name,int age) {
name = name ;
age = age ;
}
```

在Java程序中，{ } 是属于一个结构的边界符，那么在程序里面当进行变量（参数、属性都被称为变量），使用的时候都会以 {} 作为一个查找边界。
所以就按照 就近取用的原则，此时的构造方法并没有能够访问类中的属性，为了区别类中的属性与参数的区别，往往会在属性前加一个 this，表示本类属性

```
class Person {
    private String name ;
    private int age ;
    public Person(String n,int a) {
        this.name = n ;
        this.age = a ;
    }
    public void tell() {
```

```
        System.out.println("姓名：" + this.name + "年龄：" + this.age) ;
    }
    //setter | getter略

}
public class JavaStudy {
    public static void main(String args[]) {
        Person per = new Person("王五",38) ;
        per.tell() ;
    }
}
```

只要是访问本类中属性的时候，一定要加上 this ，实现访问。

2.使用this调用方法
除了使用属性之外，也可以实现方法的调用，但是对于方法的调用就必须考虑构造和普通方法：
构造方法调用（this()）：使用关键字new实例化对象的时候次啊会调用构造方法；
普通调用方法（this.方法名称()）：实例化对象产生之后就可以调用同方法；

范例：调用类中的普通方法

```
class Person {
    private String name ;
    private int age ;
    public Person(String name,int age) {
        this.setName(name) ;
        this.setAge(age) ; //加于不加，都表示本类方法
    }
    public void tell() {
        System.out.println("姓名：" + this.name + "年龄：" + this.age) ;
    }
    public void setName(String name) {
        this.name = name ;
    }
    public void setAge(int age) {
        this.age = age ;
    }
    public String getName() {
        return this.name ;
    }
}
public class JavaStudy {
    public static void main(String args[]) {
        Person per = new Person("王五",38) ;
        per.tell() ;
    }
}
```

除了普通的方法调用之外，还需要进行方法调用，对于方法的调用，是需要放在构造方法中执行。

假设：类中一共定义有三个构造方法，但是他要求不管调用哪个构造方法，都执行一行输出语句 （一个新的Person类实例化）

传统做法实现：

```
class Person {
    private String name ;
    private int age ;
    public Person(String name,int age) {
        this.setName(name) ;
        this.setAge(age) ; //加于不加，都表示本类方法
    }
    public void tell() {
        System.out.println("姓名：" + this.name + "年龄：" + this.age) ;
    }
    public void setName(String name) {
        this.name = name ;
    }
    public void setAge(int age) {
        this.age = age ;
    }
    public String getName() {
        return this.name ;
    }
}
public class JavaStudy {
    public static void main(String args[]) {
        Person per = new Person("王五",38) ;
        per.tell() ;
    }
}
```

代码结构可以重用，提供的是一个中间独立的支持

利用this()构造调整优化

```
class Person {
    private String name ;
    private int age ;
    public Person() {
        System.out.println("*****") ;
    }
    public Person(String name) {
        this () ; //调用本地无参构造
        this.name = name ;
    }
    public Person(String name,int age) {
        this(name) ; //调用单参构造
        this.age = age ;
    }
    public void tell() {
        System.out.println("姓名：" + this.name + "年龄：" + this.age) ;
    }

}
public class JavaStudy {
    public static void main(String args[]) {
        Person per = new Person("王五",38) ;
```

```
        per.tell() ;
    }
}
```

构造方法必须在实例化，新对象的时候调用 ，所以this()的语句只允许放在构造方法的首行

构造方法互调用案例：
（该类中提供有编号、部门、姓名、工资等）
无参构造：编号定义为1000，定义形式为无名氏；
但参构造：传递编号、新明定义为“新员工”，部门定义为“未定”，工资为0
三参构造：传递编号，姓名，部门，工资为2500
四参构造：所有的属性全部进行传递。

范例：代码的初期实现：

```
class Emp {
    private long empno ; //员工编号
    private String ename ; //员工姓名
    private String dept ; //部门名称
    private double salary ; //基本工资
    public Emp() {
        this.empno = 1000 ;
        this.ename = "无名氏" ;
    }
    public Emp(long empno) {
        this.empno = empno ;
        this.ename = "新员工" ;
        this.dept = "未定" ;
    }
    public Emp(long empno,String ename,String dept) {
        this.empno = empno ;
        this.ename = ename ;
        this.dept = dept ;
    }
    public Emp(long empno,String ename,String dept,double salary) {
        this.empno = empno ;
        this.ename = ename ;
        this.dept = dept ;
        this.salary = salary ;
    }
    public String getInfo() {
        return "雇员编号 :" + this.empno +
               "、雇员姓名 :" + this.ename +
               "、部门名称 :" + this.dept +
               "、基本工资 :" + this.salary ;
    }
}
public class JavaStudy {
    public static void main(String args[]) {
        Emp emp = new Emp(7369L,"SSSSA","AAAAA",6500.00) ;
        System.out.println(emp.getInfo()) ;
    }
}
```

可以对Emp进行简化定义：

```
class Emp {
    private long empno ; //员工编号
    private String ename ; //员工姓名
    private String dept ; //部门名称
    private double salary ; //基本工资
    public Emp() {
        this(1000,"无名氏",null,0.0) ;
    }
    public Emp(long empno) {
        this(empno,"新员工","未定",0.0) ;
    }
    public Emp(long empno,String ename,String dept) {
        this(empno,ename,dept,2500.00) ;
    }
    public Emp(long empno,String ename,String dept,double salary) {
        this.empno = empno ;
        this.ename = ename ;
        this.dept = dept ;
        this.salary = salary ;
    }
    public String getInfo() {
        return "雇员编号 :" + this.empno +
                "、雇员姓名 :" + this.ename +
                "、部门名称 :" + this.dept +
                "、基本工资 :" + this.salary ;
    }
}
public class JavaStudy {
    public static void main(String args[]) {
        Emp emp = new Emp(7369L,"SSSSA","AAAAA") ;
        System.out.println(emp.getInfo()) ;
    }
}
```

3.简单Java类在

在以后的项目开发的过程中，Java类都将作为一个重要的组成部分存在，慢慢接触到了正规项目的设计之后简单Java类无处不在，并且会产生一系列的变化

所谓的简单Java类，就是可以描述一类信息的程序类，例如描述一个人，一本书

并且这个类之中并没有特别复杂的逻辑操作，只作为一种信息储存的媒介存在。

对于简单的Java类而言，其核心的开发结构如下：

类名称一定有意义，可以明确描述某一类事务

类之中的所有属性都必须使用private进行封装，同时封装后的数据必须要提供setter、getter方法；

类之中可以提供无数多个构造方法，但是必须要保留有无参构造方法。

类之中不允许出现任何输出语句，所有 内容的获取必须返回

（非必须）可以提供一个获取对象详细信息的方法，暂时将此类方法名称定义为 getInfo()

范例：定义一个简单Java类

```
class Dept { //类名称可以明确描述出某类事物
    private long deptno ;
    private String dname ;
```

```
    private String loc ;
    public Dept() {} //必须提供有无参
    public Dept(long deptno,String dname,String loc) {
        this.deptno = deptno ;
        this.dname = dname ;
        this.loc = loc ;
    }
    public String getInfo() {
        return "部门单位 :" + this.deptno + "部门名称" + this.dname + "部门位置" + this.loc ;
    }
    public void setDeptno(long deptno) {
        this.deptno = deptno ;
    }
    public void setDname(String dname) {
        this.dname = dname  ;
    }
    public void setLoc(String loc) {
        this.loc = loc ;
    }
    public long getDepton() {
        return this.deptno ;
    }
    public String getDname() {
        return this.dname ;
    }
    public String getLoc() {
        return this.loc ;
    }

}
public class JavaStudy {
    public static void main(String args[]) {
        Dept dept = new Dept(10,"技术部","北极熊") ;
        System.out.println(dept.getInfo()) ;
    }
}
```

这种的简单Java类，基本上融合到了现在所接触到的概念，例如：类的定义，数据类型分析，数据类型划分，private 封装，构造方法，方法定义，对象实例化