类变量也叫静态变量，也就是在变量前加了static 的变量； 实例变量也叫对象变量，即没加static 的变量；区别在于：类变量和实例变量的区别在于：类变量是所有对象共有，其中一个对象将它值改变，其他对象得到的就是改变后的结果；而实例变量则属对象私有，某一个对象将其值改变，不影响其他对象。

在类内部，变量定义的先后顺序决定了初始化的顺序。即使变量定义散布于方法定义之间，它们仍旧会在任何方法（包括构造器）被调用之前得到初始化。

/\*

\* 由此可见，先初始化变量，其次构造方法，另外，w3这个引用被初始化两次。

\*/

**class** Window{

Window(**int** marker){

System.***out***.println("Windows("+marker+")");

}

}

**class** House{

Window w1=**new** Window(1);

House(){

System.***out***.println("House()");

w3=**new** Window(33);

}

Window w2=**new** Window(2);

**void** f(){

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

}

Window w3=**new** Window(3);

}

**public** **class** ConstructorInit{

/\*\*

\* **@param** args

\*/

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

// **TODO** Auto-generated method stub

House house=**new** House();

house.f();

}

}

静态数据的初始化

**只有在第一个对象被创建,或第一次访问静态数据**的时候，它们**才会被初始化**。**此后**，静态对象**不会再被初始化**。多个类加载的时候，不是依次初始化好每个类的静态数据，再初始化每个类的实例成员，而是初始化好第一个类的静态数据，实例变量，再初始化下一个类的静态数据，实例变量。

class Bowl{

Bowl(int marker){

System.out.println("Bowl("+marker+")");

}

void f1(int marker){

System.out.println("f1("+marker+")");

}

}

class Table{

static Bowl bowl1=new Bowl(1);

Table(){

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

bowl2.f1(1);

}

void f2(int marker){

System.out.println("f2("+marker+")");

}

static Bowl bowl2=new Bowl(2);

}

class Cupboard{

Bowl bowl3=new Bowl(3);

static Bowl bowl4=new Bowl(4);

Cupboard(){

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

bowl4.f1(2);

}

void f3(int marker){

System.out.println("f3("+marker+")");

}

static Bowl bowl5=new Bowl(5);

}

public class StaticInit {

public static void main(String args[])

{

System.out.println("Creating new Cupboard() in main");

new Cupboard();

System.out.println("\nCreating new Table() in main");

new Table();

}

}

static关键字不能应用于局部变量，因此它只能作用于域:

你在方法里面定义的变量是局部变量，就是说他有一定的作用范围和生命周期，就只能在方法里面使用而不能将其扩展到别的地方，这个变量在方法结束后就会被回收器回收，就不再存在了，而你要强制将其加上static就是要把它的作用范围扩展到整个类，这就与你开始定义这个变量时违背了，这是语法上的错误。  
建议深刻理解这句话  
static成员变量属于类，而不是对象

无论创建多少个对象，**静态数据永远只占用一份存储空间**。static关键字不能应用于局部变量，因此它只能作用于域。如果一个域是静态的基本类型域，且没有对它进行初始化，那么它就会获得基本类型的标准初值，而如果它是一个对象的引用，则默认初始值为null。例子如下  
**package** InitClass;

**class** Bowl{

Bowl(**int** marker){

System.***out***.println("Bowl("+marker+")");

}

**void** f1(**int** marker){

System.***out***.println("f1("+marker+")");

}

}

**class** Table{

**static** Bowl *bowl1*=**new** Bowl(1);

Table(){

System.***out***.println("Table()");

*bowl2*.f1(1);

}

**void** f2(**int** marker){

System.***out***.println("f2("+marker+")");

}

**static** Bowl *bowl2*=**new** Bowl(2);

}

**class** Cupboard{

Bowl bowl3=**new** Bowl(3);

**static** Bowl *bowl4*=**new** Bowl(4);

Cupboard(){

System.***out***.println("Cupboard()");

*bowl4*.f1(2);

}

**void** f3(**int** marker){

System.***out***.println("f3("+marker+")");

}

**static** Bowl *bowl5*=**new** Bowl(5);

}

**public** **class** StaticInit {

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

{

System.***out***.println("Creating new Cupboard() in main");

**new** Cupboard();

System.***out***.println("Creating new Cupboard() in main");

**new** Cupboard();

*table*.f2(1);

*cupboard*.f3(1);

}

**static** Table *table*=**new** Table();

**static** Cupboard *cupboard*=**new** Cupboard();

}

**class** Mug{

Mug(**int** marker)

{

System.***out***.println("Mug("+marker+")");

}

}

**public** **class** Mugs

{

**int** i = printNumber(8);

Mug mug1;

Mug mug2;

{

mug1 = **new** Mug(1);

mug2 = **new** Mug(2);

}

Mugs(){

System.***out***.println("Mugs()");

}

Mugs(**int** n)

{

System.***out***.println("Mugs("+n+")");

}

**int** printNumber(**int** n)

{

System.***out***.println(n);

**return** n;

}

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

System.***out***.println("new Mugs(): ");

**new** Mugs();

System.***out***.println("\nnew Mugs(n): ");

**new** Mugs(6);

}

}

**继承与初始化**

加载器开始启动，加载导出类，然后加载基类...加载根基类。根基类的static初始化，然后是下一个导出类。所有类加载完毕，对象就可以创建。首先，对象中所有的基本类型都会被设为默认值，对象引用被设为null。然后基类的构造器会被调用。在基类构造器完成之后，实例变量按其次序被初始化。最后，构造器的其余部分被执行。

自己的总结：加载类时首先初始化static变量。

基类静态变量初始化--->子类静态变量初始化--->创建对象--->调用基类实例变量初始化方法--->调用基类构造方法--->调用子类实例变量初始化方法--->调用子类构造方法

//: Beetle.java

//The full process of initialization.

**class** Insect {

**int** i = 9;

**int** j=*prt*("实例变量 j initialized");

**static** **int** *x1* = *prt*("static Insect.x1 initialized");//注意这里是static字段

Insect() {

*prt*("i = " + i + ", j = " + j);

j = 39;

}

**static** **int** prt(String s) {

System.***out***.println(s);

**return** 47;

}

}

**public** **class** Beetle **extends** Insect {

**int** k = *prt*("Beetle.k initialized");

**static** **int** *x2* = *prt*("static Beetle.x2 initialized");//注意这里是static字段

Beetle() {

*prt*("k = " + k);

*prt*("j = " + j);

}

**static** **int** prt(String s) {

System.***out***.println(s);

**return** 63;

}

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

*prt*("Beetle constructor");

Beetle b = **new** Beetle();

}

}

class Code{

{

System.out.println("Code的构造块");

}

static{

System.out.println("Code的静态代码块");

}

public Code(){

System.out.println("Code的构造方法");

}

}

public class CodeBlock03{

{

System.out.println("CodeBlock03的构造块");

}

static{

System.out.println("CodeBlock03的静态代码块");

}

public CodeBlock03(){

System.out.println("CodeBlock03的构造方法");

}

public static void main(String[] args){

System.out.println("CodeBlock03的主方法");

new Code();

new Code();

new CodeBlock03();

new CodeBlock03();

}

}/\*

CodeBlock03的静态代码块

CodeBlock03的主方法

Code的静态代码块

Code的构造块

Code的构造方法

Code的构造块

Code的构造方法

CodeBlock03的构造块

CodeBlock03的构造方法

CodeBlock03的构造块

CodeBlock03的构造方法\*/

<http://www.cnblogs.com/sophine/p/3531282.html>

类加载的时候先加载main方法所在的类，初始化该类的静态变量和静态方法。