优化软件与应用

主讲人:　　雒兴刚
东北大学系统工程研究所
Email：luoxinggang@ise.neu.edu.cn
Tel: 83682292

信息科学与工程学院
COLLEGE OF INFORMATION SCIENCE AND ENGINEERING

第十章 ILOG脚本

ILOG 脚本：基本语法结构

1、大括号表示复合语句，如
if (a > b) { var c = a; a = b; b = c }

2、注释与C语言相同；

3、代码支持的常量：
Numbers, for example: 12 14.5 1.7e-100
Strings, for example, "Ford" "Hello world\n"
Booleans, either true or false
The null value: null.

4、支持的运算符见下页；

第十章 ILOG脚本

Category	Operators
Sequence	,
Assignment	= += -= *= /= %= <<= >>= >>>= &= ^= |=
Conditional	?:
Logical-or	||
Logical-and	&&
Bitwise-or	|
Bitwise-xor	^
Bitwise-and	&
Equality	== !=
Relational	< <= > >=
Bitwise shift	<< >> >>>
Addition, subtraction	+ -
Multiply, divide	* / %
Negation, increment, typeof	! ~ - ++ -- typeof
Call	()
New	new
Property	. []

第十章 ILOG脚本

快捷运算符

Syntax	Shorthand for
++X	X = X+1
X++	Same as ++X, but returns the initial value of X instead of its new value.
--X	X = X-1
X--	Same as --X, but returns the initial value of X instead of its new value.
X += Y	X = X + Y
X -= Y	X = X - Y
X *= Y	X = X * Y
X /= Y	X = X / Y
X %= Y	X = X % Y
X <<= Y	X = X << Y
X >>= Y	X = X >> Y
X >>>= Y	X = X >>> Y
X &= Y	X = X & Y
X ^= Y	X = X ^ Y
X |= Y	X = X | Y

第十章 ILOG脚本

ILOG 脚本的基本语法结构

5、存取属性
格式1：value.name ，如

```
str.length;
getCar().name;
```

格式2：value[name] ，如

```
str["length"]; // Same as str.length
getCar()[getPropertyName()] ;
myArray[10] ;
myArray[i+1] ;
```

6、函数调用：可带参数

```
parseInt(field) writeln("Hello ", name);
 doAction() ;
str.substring(start, start+length) ;
```

第十章 ILOG脚本

ILOG 脚本的基本语法结构

7、特殊关键字

this：当前对象，同C＋＋语法；

arguments：实参数组；例如

```
function sum() {
var res = 0 ;
for (var i=0; i<arguments.length; i++)
        res = res+arguments[i] ;
return res; }
```

调用 **sum(1, 3, 5)** 返回 **9.**

第十章 ILOG脚本

ILOG 脚本的基本语法结构

8、特殊操作符

new 、delete、typeof：用法同C＋＋

expression1 , expression2 ： 顺序执行expression1 和 expression2，例

```
for (var i=0, j=0; i<10; i++, j+=2)
        { writeln(j, " is twice as big as ", i); }
```

第十章 ILOG脚本

ILOG 脚本的控制语句

1、条件语句

语法：if (expression) statement1 [else statement2]

```
if (a == b) writeln("They are equal");
  else writeln("They are not equal");

if (s.indexOf("a") < 0) {
  write("The string ", s);
  writeln(" doesn't contains the letter a");
}
```

2、循环语句

while语句：while (expression) statement*t*

```
while (a*a < b) a = a+1 ;
```

第十章 ILOG脚本

ILOG 脚本的控制语句

For语句：for ([initialize] ; [condition] ; [update]) statement

```
for (var i=0; i < a.length; i++) {
  sum = sum+a[i];
  prod = prod*a[i];
}
```

For语句： for ([var] variable in expression) statement

```
function printProperties(v) {
  for (var p in v)
    writeln(p, " -> ", v[p]);
}
```

第十章 ILOG脚本

ILOG 脚本的控制语句

break语句：

```
while (i < a.length) {
  if (a[i] == "foo") {
    foundFoo = true;
    break;
  }
  i = i+1;
}
```

continue语句：

```
for (var i=0; i < a.length; i++) {
  if (a[i] < 0) continue;
  writeln("A positive number: ", a[i]);
}
```

第十章 ILOG脚本

ILOG 脚本的控制语句

3、变量定义语句

语法：**var decl1, ..., decln**

```
var x ;
var name = "Joe" ;
var average = (a+b)/2, sum, message="Hello" ;
```

在函数体内定义的变量是局部变量，在外部定义的是全局变量；

```
var count = 0;

function NextNumber() {
  var i,j=0;
  count = count+1;
  return count;
}
```

第十章 ILOG脚本

ILOG 脚本的控制语句

4、函数定义语句

语法：**[static] function name(v1, ..., vn) { statements }**

return [expression] ;

比较特别的是，这里的形式参数和实际参数数量（例如表示为**m**和**n**）可不等。如果前者多，则前**n**个按照实际参数初始化，后**m**－**n**个为未初始化状态；如果后者多，则只有前**m**个实际参数有效。函数里可以没有**return**，这种情况下返回一个初始化的不确定值。

例如

```
function add(a, b) {
  return a+b;
}
```

第十章 ILOG脚本

ILOG 脚本的控制语句

5、块语句

语法：with (expression) statement

例如

```
with ("abc") {
  writeln("The length is ", length);
}
```

相当于输出“abc”.length

第十章 ILOG脚本

ILOG 脚本中的数字

1、n进制数字

10进制数字：
15
3.14
4e100
.25
5.25e-10

16进制数字：0x前缀
0x3ff
0x0

8进制数字：0前缀
0123
0777

第十章 ILOG脚本

ILOG 脚本中的数字

2、特殊数字

NaN (Not-A-Number)，例如

Math.sqrt(-1) NaN
Math.sqrt(NaN) NaN
NaN + 3 NaN
NaN == NaN false
NaN <= 3 false
NaN >= 3 false

Infinity (positive infinity)和 -Infinity (negative infinity)，例如

1/0 Infinity
-1/0 -Infinity
1/Infinity 0
Infinity == Infinity true

第十章 ILOG脚本

ILOG 脚本中的数字

3、数字转换

脚本会在表达式中自动 转换数字，基本规则是

✓字符串转换成数字。如果不是一个数字则为**NaN.**

✓**true** 转换成 **1**， **false** 转换成 **0.**

✓**null** 转换成 **0.**

✓日期转换成毫秒（从 **00:00:00 UTC, January 1, 1970**开始）

例如，可以这么写

math.sqrt(“25”) 结果是**5**

"3" * "4"结果是**12**

有些操作符（如＋）可以对字符串和数字操作，那么只要有一个元素是字符就转换成字符；如果没有字符，就转换成数字。如

“3” + true结果是**"3true"**

3 + true结果是**4**

第十章 ILOG脚本

ILOG 脚本中的数字

对于比较操作符（如== 、 >= ），只要有一个元素是数字就转换成数字；
如果没有数字，就按照字符比较。如

“10” > “2” 结果是false
"10" > 2结果是true

反之，如果将数字转换成字符，需要一个函数，格式为
number.toString()

例如
(14.3e2).toString() 结果是"1430"

第十章 ILOG脚本

ILOG 脚本中的数字

4、数字函数：见表

Syntax	Effect
Math.abs(x)	Returns the absolute value of x.
Math.max(x, y) Math.min(x, y)	Math.max(x, y) returns the larger of x and y, and Math.min(x, y) returns the smaller of the two.
Math.random( )	Returns a pseudo-random number between 0, inclusive, and 1, exclusive.
Math.ceil(x) Math.floor(x) Math.round(x)	Math.ceil(x) returns the smallest integer value greater than or equal to x. Math.floor(x) returns the greatest integer value less than or equal to x. Math.round(x) returns the nearest integer value to x.
Math.sqrt(x)	Returns the square root of x.
Math.sin(x) Math.cos(x) Math.tan(x) Math.asin(x) Math.acos(x) Math.atan(x) Math.atan2(y, x)	Math.sin(x), Math.cos(x) and Math.tan(x) return the trigonometric functions sine, cosine and tangent respectively of a radian argument. Math.asin(x) returns the arcsine of x in the range -pi/2 to pi/2. Math.acos(x) returns the arc cosine of x in the range 0 to pi. Math.atan(x) returns the arc tangent of x in the range -pi/2 to pi/2. Math.atan2(y, x) converts rectangular coordinates (x, y) to polar coordinates (r, a) by computing a as an arc tangent of y/x in the range -pi to pi.
Math.exp(x) Math.log(x) Math.pow(x, y)	Math.exp(x) computes the exponential function. Math.log(x) computes the natural logarithm of x. Math.pow(x, y) computes x raised to the power y.

第十章 ILOG脚本

ILOG 脚本中的数字

5、数字常量：见表

Syntax	Value
NaN	Contains the NaN value.
Infinity	Contains the Infinity value.
Number.NaN	Same as NaN.
Number.MAX_VALUE	The maximum representable number, approximately 1.79E+308.
Number.MIN_VALUE	The smallest representable positive number, approximately 2.22E-308.
Math.E	Euler's constant, e, and the base of natural logarithms, approximately 2.718.
Math.LN10	The natural logarithm of 10, approximately 2.302.
Math.LN2	The natural logarithm of 2, approximately 0.693.
Math.LOG2E	The base 2 logarithm of e, approximately 1.442.
Math.LOG10E	The base 10 logarithm of e, approximately 0.434.
Math.PI	The ratio of the circumference of a circle to its diameter, approximately 3.142.
Math.SQRT1_2	The square root of one-half, approximately 0.707.
Math.SQRT2	The square root of 2, approximately 1.414.

第十章 ILOG脚本

ILOG 脚本中的数字

6、数学操作符

x + y 、x - y 、x * y 、x / y 、- x： 加减乘除、负；

x % y ； 取余数；

x == y、x != y： 是否相等的比较，返回**true**或者**false**；

x < y 、x <= y、x > y、x >= y： 大小比较，返回**true**或者**false**；

x & y、x | y、x ^ y 、~ x ： AND, OR, XOR，NOT 位运算

例如

14 & 9 8 (1110 & 1001 1000)

14 | 9 15 (1110 | 1001 1111)

14 ^ 9 7 (1110 ^ 1001 111)

~ 14 1 (~ 1110 0001)

第十章 ILOG脚本

ILOG 脚本的数字

6、数学操作符

x << y 、x >> y 、x >>> y： 位位移。<< 左移， >> 右移 (保留符号位), >>>右移并左面补**0**，例如

9 << 2 36 (1001 << 2 100100)
9 >> 2 2 (1001 >> 2 10)
-9 >> 2 -2 (1..11001 >> 2 1..11110)
-9 >>> 2 1073741821 (1..11001 >>> 2 01..11110)

第十章 ILOG脚本

ILOG 脚本的字符串

1、字符串自动转换

函数实参或表达式需要转换成字符串格式时，系统自动转换，如

"The " + 10 + " commandments" -> "The 10 commandments"

2、字符串属性：只有一个取长度的属性
***格式：string*.length**
“abc”.length 结果是3
"".length结果是0

第十章 ILOG脚本

ILOG 脚本的字符串

3、字符串方法

***string*.substring (*start* [, *end*])：取子串**

0123456".substring(0, 3) "012"
"0123456".substring(2, 4) "23"
"0123456".substring(2) "23456"

***string*.charAt (*index*) ：取字符串指定位置的字符**

""abcdef".charAt(0) "a"
"abcdef".charAt(3) "d"
"abcdef".charAt(100) ""

第十章 ILOG脚本

ILOG 脚本的字符串

***string*.charCodeAt(*index*) ： 取字符串指定位置的字符的ASCII码**

"abcdef".charCodeAt(0) 97
"abcdef".charCodeAt(3) 100
"abcdef".charCodeAt(100) NaN

***string*.indexOf (*substring* [, *index*]) ： 子串在什么位置？找到返回索引，没有返回－1**

"abcdabcd".indexOf("bc") 1
"abcdabcd".indexOf("bc", 1) 1
"abcdabcd".indexOf("bc", 2) 5
"abcdabcd".indexOf("bc", 10) -1
"abcdabcd".indexOf("foo") -1
"abcdabcd".indexOf("BC") -1

第十章 ILOG脚本

ILOG 脚本的字符串

***string*.lastIndexOf (*substring* [, *index*])：** 子串最后一次出现的位置

"abcdabcd".lastIndexOf("bc") 5
"abcdabcd".lastIndexOf("bc", 5) 5
"abcdabcd".lastIndexOf("bc", 4) 1
"abcdabcd".lastIndexOf("bc", 0) -1
"abcdabcd".lastIndexOf("foo") -1
"abcdabcd".lastIndexOf("BC") -1

***string*.toLowerCase()**
***string*.toUpperCase() ：** 转换大小写

***string*.split(*separator*) ：** 按照separator分割串成为数组
“first name,last name,age”.split(“,”) 结果是一个数组，长度位3，**a[0]** ＝ **“first name”, a[1]＝ “last name”, a[2] ＝ "age".**

第十章 ILOG脚本

ILOG 脚本的字符串

***string*.lastIndexOf (*substring* [, *index*])：** 子串最后一次出现的位置

"abcdabcd".lastIndexOf("bc") 5
"abcdabcd".lastIndexOf("bc", 5) 5
"abcdabcd".lastIndexOf("bc", 4) 1
"abcdabcd".lastIndexOf("bc", 0) -1
"abcdabcd".lastIndexOf("foo") -1
"abcdabcd".lastIndexOf("BC") -1

***string*.toLowerCase()**
***string*.toUpperCase() ：** 转换大小写

***string*.split(*separator*) ：** 按照separator分割串成为数组
“first name,last name,age”.split(“,”) 结果是一个数组，长度位3，**a[0]** ＝ **“first name”, a[1]＝ “last name”, a[2] ＝ "age".**

第十章 ILOG脚本

ILOG 脚本的字符串

4、字符串函数

String.fromCharCode(*code*)：由ASCII变成字符

String.fromCharCode(65) -> "A"
String.fromCharCode(0xA9) -> "©"

parseInt(*string* [, *base*])：字符串转整型

parseInt("123") -123
parseInt("-123") -123
parseInt("123.45") 123
parseInt("1001010010110", 2) 4758
parseInt("a9", 16) 169
parseInt("0xa9") 169
parseInt("010") 8
parseInt("123 poodles") 123
parseInt("a lot of poodles") NaN

第十章 ILOG脚本

ILOG 脚本的字符串

parseFloat(*string*) ： 字符串转浮点型

parseFloat("-3.14e-15") -3.14e-15
parseFloat("-3.14e-15 poodles") -3.14e-15
parseFloat("a fraction of a poodle") NaN

5、字符串操作符

***string1* + *string2* ： 字符串连接**
"Hello," + " world" "Hello, world"
"Your age is " + 23 -> "Your age is 23“
23 + " is your age" -> "23 is your age"

第十章 ILOG脚本

ILOG 脚本的字符串

string1* == *string2 、***string1* != *string2*** ： 字符串比较是否相等

"a string" == "a string"　　-> true
"a string" == "another string" -> false
"a string" == "A STRING"　　-> false
"a string" != "a string"　　-> false
"a string" != "another string" -> true

string1* < *string2 、***string1* <= *string2*** 、***string1* > *string2*** 、***string1* >= *string2*** ： 按字典顺序，字符串比较大小

"abc" < "xyz"　-> true
"a" < "abc"　-> true
"xyz" < "abc"　-> false
"abc" < "abc"　-> false
"abc" > "xyz"　-> false
"a" > "abc"　-> false
"xyz" > "abc"　-> true

第十章 ILOG脚本

ILOG 脚本的布尔值

如果需要布尔值时，系统会自动进行转换，规则是：

- ✓**0 转换成 false.**
- ✓空字符串 ""转换成 **false.**
- ✓**null转换成 false.**
- ✓未定义类型转换成 **false.**
- ✓其它任何非布尔型转换成 **true.**

```
if ("") writeln("True"); else writeln("False");
if (123) writeln("True"); else writeln("False");
```

布尔值的方法只有一个：***boolean*.toString()** ，结果是“**true**”或者“**false**”两个字符串之间的一个，例如

```
true.toString -> "true"
false.toString -> "false"
```

逻辑操作有：！*boolean* 、*exp1* && *exp2* 、*exp1* || *exp2* 、*condition* ? *exp1* : *exp2*

第十章 ILOG脚本

ILOG 脚本的数组

创建数组有2种方法，一种是：**new Array(*length*)**

new Array(12) -> 创建一个数组，共12个成员，**a[0] to a[11]**，初始值**null.**

还有一种是：**new Array(*element1, ..., elementn*)**

new Array(327, “hello world”) ->创建一个数组， 共2个成员，**a[0] = 327**，**a[1] = "hello world".**

存取数组成员的语法是 ***array*[*index*]**，例如

a = new Array(“foo”, 12, true) 结果是
a[0] -> “foo”； a[1] -> 12； a[2] -> true a[3] -> null ； a[1000] -> null

如果越界使用，系统会自动**expand**数组长度，而不是出错。
a[1000] = “bar” ->可以！

第十章 ILOG脚本

ILOG 脚本的数组

取数组成员个数可以用：***array*.length** 例如，

a = new Array(“a”, “b”, “c”) 结果是:
a.length -> 3

a[100] = “bar”; 结果是a.length -> 101

也可以通过**length**来改变数组大小，如

a = new Array();
a[4] = "foo";
a[9] = “bar”; 结果是a.length -> 10
a.length = 5 结果是a.length -> 5 a[4] -> "foo" a[9] -> null

第十章 ILOG脚本

ILOG 脚本的数组

数组有一些很有用的函数，例如

***array*.join([*separator*])** ： 将数组成员连接起来；

a = new Array(“foo”, 12, true) 那么:
a.join(“//”) 结果是 “foo//12//true” a.join() 结果是 "foo,12,true"

array*.sort([*function*])** ： 按照字母顺序，将数组元素排序； 也可以自己写 ***function 自定义排序规则；

array.reverse() ： 数组成员颠倒顺序
a = new Array(“foo”, 12, “hello”, true, false)
结果为
a[0] false
a[1] true
a[2] "hello"
a[3] 12

第十章 ILOG脚本

ILOG 脚本的对象

ILOG脚本支持对象的创建，例如：

```
myCar = new Object() // 开始没有属性
myCar.name = “Ford” //对象属性赋值
myCar.year = 1985
```

定义对象的方法是先定义一个函数，然后赋给对象，如

```
function start_engine() {
  writeln("vroom vroom\n")
}
myCar.start = start_engine
```

一般的， start_engine还是命名为start比较好，可以避免引起混淆。

```
function get_name() {
  return this.name    //使用this指针
}
myCar.getName = get_name
```

第十章 ILOG脚本

ILOG 脚本的对象

用户自定义构造函数的例子：

```
function Car(name, year) {
  this.name = name;
  this.year = year;
  this.start = start_engine;
}

new Car("Ford", "1985")
```

第十章 ILOG脚本

ILOG 脚本的时间类型

时间类型的对象有以下几种构造方法：

new Date()：返回当前时间；

new Date(*milliseconds*)：返回**00:00:00 UTC, January 1, 1970** 加***milliseconds***的时间，例如：

new Date(0) -> a date representing 00:00:00 UTC, January 1, 1970.
new Date(1000*60*60*24*20) -> a date representing twenty days after 00:00:00 UTC, January 1, 1970.
new Date(-1000*60*60*24*20) -> a date representing twenty days before 00:00:00 UTC, January 1, 1970.

第十章 ILOG脚本

ILOG 脚本的时间类型

new Date(*string*) ： string格式是month/day/year hour:minute:second msecond

例如：**new Date("12/25/1932 14:35:12 820")**
A date representing December 25th, 1932, at 2:35 PM plus 12 seconds and 820 milliseconds, local time.

new Date(*year*, *month*,
[, *day*
[, *hours*
[, *minutes*
[, *seconds*
[, *mseconds*]]]]]) ： 按照year, month, day等给出的Date，没给出的按0

new Date(1932, 11, 25, 14, 35, 12, 820)
A date representing December 25th, 1932, at 2:35 PM plus 12 seconds and 820 milliseconds, local time.

第十章 ILOG脚本

ILOG 脚本的时间类型

Date有以下的方法：

***date*.getTime() 、*date*.setTime(*milliseconds*)**

d = new Date(3427) 则:
d.getTime() -> 3427

***date*.toLocaleString()、*date*.toUTCString()**

d = new Date(“3/12/1997 12:45:00 0”) 则:
d.toLocaleString() -> "03/12/1997 12:45:00 000"
d.toUTCString() -> "03/12/1997 10:45:00 000",

第十章 ILOG脚本

ILOG 脚本的时间类型

Date有以下的方法：

***date*.getYear() 、*date*.setYear(*year*) ;**

***date*.getMonth() 、*date*.setMonth(*month*) ;**

***date*.getDate() 、*date*.setDate(day)**

***date*.getHours()、*date*.setHours(*day*)**

***date*.getMinutes() 、date.setMinutes(*minutes*)**

***date*.getSeconds() 、date.setSeconds(*seconds*)**

***date*.getMilliseconds() 、*date*.setMilliseconds(*millisecs*)**

date.toString()

第十章 ILOG脚本

ILOG 脚本的时间类型

Date的函数有2个，一个是Date.UTC(*year, month, [, day [, hours [, minutes [, seconds [, mseconds]]]]]*) ，可以将日期转为UTC数；

另一个是Date.parse(*string*) ，string格式参见new Date(string)，函数也返回UTC数。

日期操作的例子：

date1 - date2 先转UTC数然后减
date1 < date2 先转UTC数然后比较
Date(date+10000) ->先转UTC数然后相加

```
before = new Date();
<do something>;
after = new Date();
writeln("Time for doing something: ", after-before, " milliseconds.");
```

第十章 ILOG脚本

ILOG 脚本的杂函数

1、*null*.toString() 返回字符串 “null”;
2、*undefined*.toString() 返回字符串 "undefined".
3、*function*.toString() 返回和这个函数相关的字符串 ， 例如
“foo”.substring.toString() “[primitive method substring]” eval.toString() “[primitive function eval]”

4、stop() ： 停止程序运行，转**debug**模式;
5、write(*arg1*, ..., *argn*) 、writeln(*arg1*, ..., *argn*) ： 输出函数，注意**write**每次输出后不回车，而**writeln**每次输出一行后回车;
6、loadFile(*string*) ： 调入一个脚本文件;
7、eval(*string*) ： 将字符串当作程序执行，返回最后求值的表达式，但不允许有函数。例如:

eval("2*3") -> 6
eval("var i=0; for (var j=0; j<100; j++) i=i+j; i") -> 4950 n=25;
eval("Math.sqrt(n)") -> 5 eval("function foo(x) { return x+1 }") -> error

第十章 ILOG脚本

ILOG 脚本的OPL扩展

ILOG脚本中提供了一些OPL扩展接口，这些接口(Interface)以Ilo 开头，每个接口又提供若干的属性（Property）和方法（方法），例如，我们前面例子中用到的IloOplModelSource，有一个属性和两个方法：

Named Properties

Property Name	Scripting Type	Available	Description
name	string	main	Name of this model source. See the *Interfaces User's Manual* and the API Reference Manuals for concepts.

Methods

Method Name	Return Type	Arguments	Available	Description
(constructor)	IloOplModelSource	string	main	Creates a new model source for the .mod file on the given path. See the *Interfaces User's Manual* and the API Reference Manuals for concepts.
end()	void	void	main	Releases the memory used by this object. The object becomes then unavailable for further usage.

第十章 ILOG脚本

ILOG 脚本的OPL扩展接口

这些接口(Interface)分为三类：

- ✓前处理类
- ✓后处理类
- ✓流程控制类

下面我们只列出每类的接口名称，具体每个接口的属性和方法的用法，因为量比较大，请大家参照ILOG帮助文档。

- ✓前处理类
  - ✓**IloCplex**
  - ✓**IloDiscreteDataCollection**
  - ✓**IloIntRange**
  - ✓**IloMap**
  - ✓**IloNumVar**
  - ✓**IloOplModel**
  - ✓**IloTuple**
  - ✓**IloTupleSet**

第十章 ILOG脚本

ILOG 脚本的OPL扩展

- 后处理类
  - IloConstraint
  - IloCplex
  - IloDiscreteDataCollection
  - IloIntRange
  - IloMap
  - IloNumVar
  - IloOplModel
  - IloTuple
  - IloTupleSet
- 流程控制类
  - IloConstraint
  - IloCplex
  - IloOplDataElements
  - IloOplDataSource
  - IloDiscreteDataCollection
  - IloIntRange
  - IloMap
  - IloNumVar
  - IloOplCplexBasis
  - IloOplCplexVectors
  - IloOplModel
  - IloOplModelDefinition
  - IloOplModelSource
  - IloTuple
  - IloTupleSet