

面向对象程序设计基本原则

消除代码复制

设计类时，尽量让自己的所有成员变量都是私有的

类和类之间的关系乘坐耦合，耦合越低越好，保持距离是形成良好代码的关键

一种跨平台语言

既是编译型，又是解释型

运算符表达式与语句

a++与++a的区别

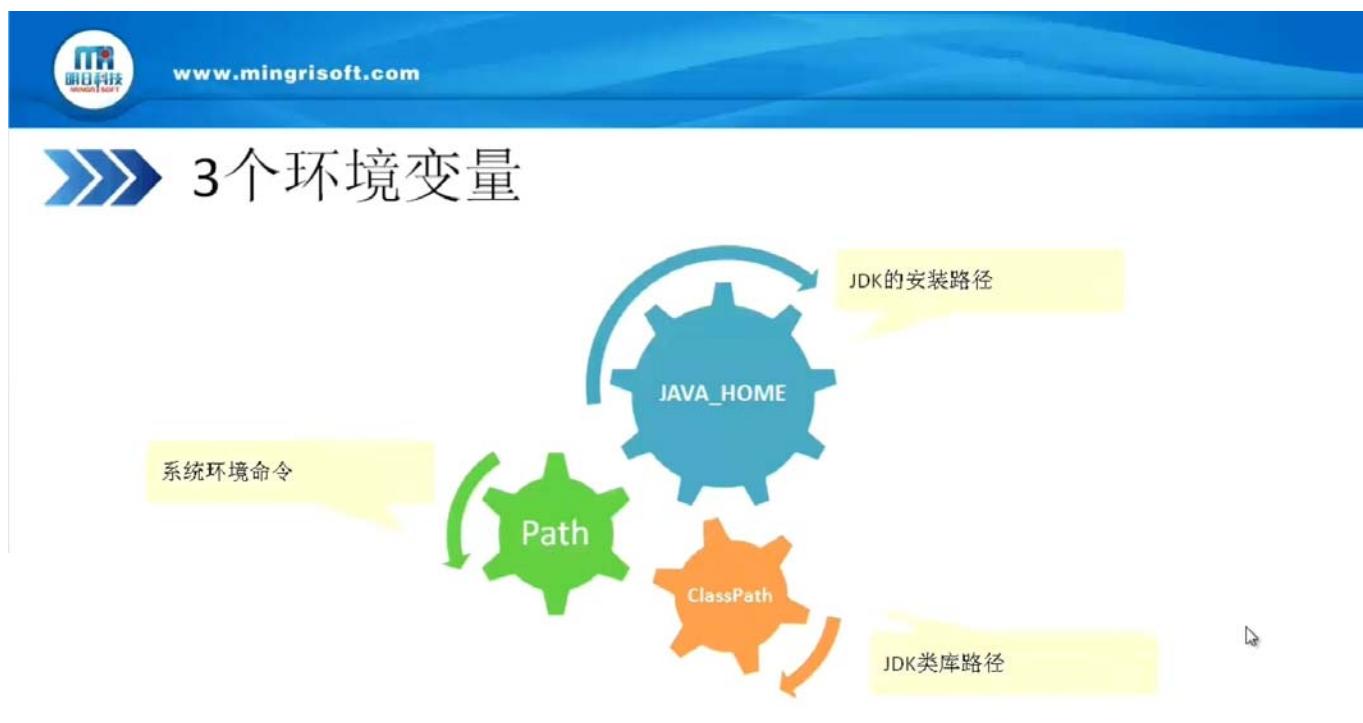
a++：先+再赋值

++a：先赋值再+

javase：桌面级开发工具

javaee：企业级开发工具，包含javase

javame：嵌入式系统开发



程序调试

单步跳过：运行完这行代码后，继续运行之后的代码

单步跳入：运行这行代码时，观察代码的运行情况

注释（文档注释）

API其实是由文档注释自动生成的

代码编写规范

每个变量的声明尽量单独占一行

关键字之间只认一个空格

关键方法要加注释

注意使用半角英文符号

标识符（严格区分大小写）

数字（不能放在最前面），字母，_，4

不推荐使用中文命名任何的标识符

标识符命名规范

www.mingrisoft.com

» 标识符的命名规范

类名	方法名	变量名	常量名
<ul style="list-style-type: none">通常使用名词，所有单词首字母大写。例如：<code>PandaFamily</code>	<ul style="list-style-type: none">通常使用动词，首单词字母小写，后续单词首字母大写。例如：<code>openDoor()</code>	<ul style="list-style-type: none">首单词字母小写，后续单词首字母大写。例如：<code>sisterName</code>	<ul style="list-style-type: none">所有字母均大写。例如：<code>GENDER(性别)</code>

变量（可以通过赋值更改变量的值）

整型，长整型，字符型，布尔型

变量声明

数据类型 变量名称=变量值

变量声明尽可能靠近变量第一次使用的地方

常量

常量声明

final(**final**是声明常量的关键字)数据类型 常量名称=常量值

习惯上，常量名全部用大写

整型

类型	内存空间
byte	1字节
short	2字节
int (默认类型)	4字节
long	8字节

给long赋值的时候要加上L或l的后缀，否则可能造成精度的丢失

使用不同进制赋值

10进制：直接int

8进制：以0开头

16进制：以0x或者0X开头

浮点型

类型	内存空间
float	4字节
double (默认类型)	8字节

给float赋值的时候，要加上F后缀，否则可能报错

float型的值一般就直接在后面加上F

java浮点值是近似值，不精确，可以使用java提供的四舍五入方法Math.round()

字符串型

概念：用单引号包含的可打印的单个字符

char可以有两种赋值方式

```
char a1='汉';  
char a2=27721;
```

//两种赋值方式相同，前一种是直接赋予字符，后一种是赋予ASCII码

转义字符

\r和\n的区别：\r是回车，使光标回到行首；\n是换行，使光标下移一行

布尔类型

boolean

只有true和false两种类型

数据类型转换

小类型向大类型转换

隐式转换（自动转换）

显示转换（强制转换）

语法（类型名）要转换的值

```
int a=100;
```

```
byte b=(byte)a;
```

大类型向小类型转换

显示转换（强制转换）

其中，char类型和int类型可以相互兼容

赋值运算符

算数运算符

+：正号；加法；拼接字符串

自增自减运算符（单目）

后置形式先赋值再自增运算

前置形式先自增运算再赋值

关系运算符（双目）

计算结果是布尔值

不能串联的使用关系运算符，使用逻辑运算符`&&`连接

位运算符

运算符	含义	举例
<code>&</code>	与	<code>a&b</code>
<code> </code>	或	<code>a b</code>
<code>~</code>	取反	<code>~a</code>
<code>^</code>	异或	<code>a^b</code>
<code><<</code>	左移位	<code>a<<2</code>
<code>>></code>	右移位	<code>b>>2</code>
<code>>>></code>	无符号右移位	<code>x>>>2</code>

位逻辑运算符也可以用作逻辑运算符

位移运算符



>>> 位移运算



左移补零

右移正数补零，负数补一

无符号右移不看正负数，一律补零

byte和short属于低精度整型，不适合做无符号右移，会溢出

复合运算符



>>> 复合赋值运算符

运算符	说明	举例	等价效果
$+=$	相加结果赋予左侧	$a += b;$	$a = a + b;$
$-=$	相减结果赋予左侧	$a -= b;$	$a = a - b;$
$*=$	相乘结果赋予左侧	$a *= b;$	$a = a * b;$
$/=$	相除结果赋予左侧	$a /= b;$	$a = a / b;$
$%=$	取余结果赋予左侧	$a %= b;$	$a = a \% b;$
$&=$	与结果赋予左侧	$a &= b;$	$a = a \& b;$
$ =$	或结果赋予左侧	$a = b;$	$a = a b;$
$^=$	异或结果赋予左侧	$a ^= b;$	$a = a ^ b;$
$<<=$	左移结果赋予左侧	$a <<= b;$	$a = a << b;$
$>>=$	右移结果赋予左侧	$a >>= b;$	$a = a >> b;$
$>>>=$	无符号右移结果赋予左侧	$a >>>= b;$	$a = a >>> b;$

复合赋值运算符可以自动完成类型转换

如

```
byte a=1;
```

a=a+2;//这是错的，原因在于2是int型，int型与byte型相加是int型，在没有强制类型转化的情况下，int型不能赋给byte型

三元运算符

```
int a=1;  
int b=a<3?2: 3;
```

注意：三元运算符是有返回值的

运算符优先级

可以简单地理解为单目运算符的优先级高于双目运算符

程序结构

顺序结构

选择结构

循环结构

if else语句

在不加括号的情况下，else默认匹配的是距离自己最近的那个if

else必须和if一起出现

布尔型判断的时候直接写就行，切记不要出现对布尔值赋值的情况

switch语句

switch语句的参数可以是字符，枚举，整数，字符串、

| 枚举类型：一个类的对象时有限且固定的，这样的类成为枚举类

if和switch的比较

|if|switch

—|—|—

判断的类型|布尔值|整型值，字符，字符串，枚举

使用的场景|判断关系表达式,逻辑表达式，浮点值|多整数值判断，多字符串值判断

while语句 (do while)

表达式不能为空

表达式不能为常数

循环体重要有改变循环条件的语句

while循环条件若是判断为真，则执行循环体；若是判断为假，则跳出循环

do while先执行语句，后判断

while先判断，后执行

do while循环在最后的while后面还有一个分号

for循环

语法

```
for (初始化表达式; 布尔表达式; 表达式3) {
```

语句

}

foreach循环

用来遍历一个集合

语法

```
for(type x:obj){
```

引用了x的java语句；

}



》》》 foreach语句的常见错误

```
int a[] = {9, 1, 6};
```

这里没有中括号

```
for(int i : a[]){  
}
```

这里没有中括号

```
for(int i[] : a ){  
}
```

是冒号，不是分号

```
for(int i ; a ){  
}
```



上一章

对数据进行操作的时候选择for循环，仅仅是遍历的时候选择foreach循环

break语句

break一般控制的是内层循环

要想使break控制外层循环，需要在外层循环前加上标签Loop，然后在break后紧跟Loop标签

```
Loop: for(i=1;i<10;i++){  
    system.out.println("i="+i);  
    for(j=1;j<9;j++){  
        system.out.println("j="+j);  
        if(j=8){  
            break Loop;  
        }  
    }  
}
```

continue语句

与break语句的使用场景很像，但是不适用于switch语句

continue也可以使用Loop标签

continue与break的区别

continue是结束这次循环，进入下一次循环；而break是直接跳出循环，终止循环

数组

将n个同类型的变量以整体的形式表示，能以简单的方式访问整体中的每个元素

整形数组，浮点型数组，字符数组，字符串数组

语法

```
数组元素类型 数组名字[];
```

```
数组元素类型[] 数组名字;
```

一维数组初始化方法

```
//第一种
```

```
int a[]={1,2,3};  
a[0]=1;  
a[1]=2;  
a[2]=3;
```

```
//第二种
```

```
int a[]={1,2,3};
```

```
//第三种
```

```
int a[]={1, 2, 3}
```

因为初始化的是数组，所以记得要加方括号

数组的length属性

用于输出数组的长度

```
int a[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19};
```

a.length//a.length的值等于数组下标最大值加一，返回的是int值

二维数组

语法

```
数组元素类型 数组名字[][];
```

```
数组元素类型[][] 数组名字;
```

```
//第一种
```

```
int a[][]=new int[3][2];
```

```
a[0]=new int[]{1,2};
```

```
a[1][0]=1;
```

```
a[1][1]=2;
```

```
a[2][0]=1;
```

```
a[2][1]=2;
```

```
//第二种
```

```
int a[][]=new int[][]{{1,2,3},{2,3,4}};
```

```
//第三种
```

```
int a[]={{1, 2, 3},{2,3,4}}
```

二维数组可能存在不规则数组的情况,即不同类的元素个数不一样

二维数组定义时一定要定义行 , 列可定义可不定义

二维数组中的length

a[][] , a.length 相当于二维数组的行数 ; a[i].length 对应的是第 i 行的长度

遍历数组

就是获取数组中的每个元素

填充和批量替换数组元素

语法

```
Arrays.fill(arr,int value)//arr:数组; value: 填充的值
```

```
Array.fill(arr,int fromIndex ,int toIndex, int value)//fromIndex: 填充的第一个索引 toIndex: 填充的最后一个索引(不包括)
```

复制数组

语法

```
Arrays.copyOf(arr,newlength);//会返回一个int型的数组
```

```
Array.copyOfRange(arr,fromIndex,toIndex);
```

多维数组

冒泡排序与选择排序

需要两个for循环嵌套（外层for循环是比较的轮数，内层for循环是每轮比较的次数）

选择排序与冒泡排序很像，都是每一轮选出一个最大（最小）元素。区别在于选择排序是每轮比较出最大（最小）的元素，再将其替换到最后一个元素的位置上；冒泡排序则是每次比较都要更改元素位置

数组排序的其他方法

Arrays.Sort()方法

```
int arr[]={1,2,3,4,5,6,32,12,11};  
Arrays.Sort(arr);//升序  
Arrays.sort(arr,Collections.reverseOrder())//降序
```

可以直接对数组排序，但是只能是升序

要想实现降序，可以使用Collections.reverseOrder()方法，如上

字符串

声明一个字符串就是创建一个字符串对象

给字符串赋值的方法：

引用字符串常量

利用构造方法直接实例化

利用字符串数组实例化

利用构造方法直接实例化：

语法

```
public String (String orginal) //文本内容  
String a=new String("一个人");  
String b=new String(a);
```

利用字符数组实例化

语法

```
public String(char[] value)//字符数组  
char[] charArray={'t','i','m','e'};  
String a=new String(charArray);
```

使用构造方法创建字符串变量时，字符串变量实质上是一个变量

利用字符数组实例化的第二种方法

语法

```
public String(char[]value,int offset,int count)//offset:起始位置, cout: 获取个数
```

拼接字符串

使用“+”运算符

还可以使用 “+=” 运算符，类似于整型的+=

concat方法

语法

```
public String concat(String str)  
String str="abc";  
str=str.concat("de");
```

字符串是常量，在内存中时不可改变的对字符串的更改其实是在内存中重新开辟出了一块空间，存储新的字符串值

但是可以将多个字符串指向同一个内存地址

字符串类型转换

显式转换：

```
static String valueOf(args)//args指代任意类型
```

static用于变量时代表类变量，是属于类自己的变量

不同于普通的成员变量（只能通过对对象来调用）

可以直接使用类名来调用

也可以像普通成员变量一样通过对象来调用

static用于函数时代表类函数，使用方法和类变量类似

隐式转换：

```
String str = "" + 数字
```

```
//System.out.println()其实也是使用到了隐式转换的思想，将括号内的内容转换为字符串输出
```

隐式转换的三种情况：

碰到字符串后，直接输出后面的内容

碰到字符串前，先做运算，后输出内容

碰到括号，先算括号中的值，后输出内容

获取字符串长度的方法：length()//类的成员方法，有括号，而且会包括空格，不能使用空字符串

获取数组长度的方法：length//数组的一个属性，没有括号

获取指定位置的字符

语法：

```
public char charAt(int index)//index是要获取的索引位置
```

获取子字符串的（索引）位置

若是子字符串不存在的话就会返回-1，存在的话就会返回所查询字符所在位置的下标索引

获取第一次出现的索引

```
public int indexOf(String str)//str:要获取的子字符串
```

```
public int indexOf(String str, int fromIndex)//复用形式，fromIndex是字符
```

串的起始位置，从起始位置向后查询

获取最后一次出现的索引

`public int lastIndexOf(String str)`//**str**: 要获取的子字符串

`public int lastIndexOf(String str,int fromIndex)`//**fromIndex**是字符串的起始位置，从起始位置向前查询

判断字符串结尾内容

语法

`public boolean endsWith(String suffix)`//**suffix**:要对比的字符串

//例如

```
String fileName="Helloworld.java";
Boolean bool1=filename.endsWith(".java");
```

判断字符串句首内容

语法

`public filename startWith(String prefix)`//可以类比**endsWith**学习

获取字符串数组

语法

```
public char[] toCharArray()
String str="这是一个字符串";
char ch[]=str.toCharArray();
```

对象数组

对象数组中每个元素都是对象的管理者而非对象本身

判断子字符串是否存在

语法

```
public boolean contains(String string)//要查找的子字符串
```

或者使用Indexof方法，若是不存在则会返回-1，存在就返回它的下标

判断的内存地址，对于字符型是行不通的，字符型相等判断要使用equals()

忽略大小写比较

语法

```
public boolean equalsIgnoreCase(String anotherString)//anotherString是被比较的字符串
```

直接引用字符串常量时，可以使用==，因为使用的是同一块内存区

java虚拟机当连接两个字符串时，先执行拼接的操作，然后赋值

截取字符串

语法

```
public String substring(int beginIndex)//beginIndex: 截取开始的位置
```

```
public String substring(int beginIndex, int endIndex)//endIndex: 截取结束的位置
```

替换字符串

语法

```
public String replace()//旧字符串序列替换成新的序列c
```

```
public String replaceAll()//旧字符串序列或者正则表达式替换成新的序列
```

//正则表达式是含有一些遇有特殊意义字符的字符串，这些特殊字符称为正则表达式的元字符。例如“\d”表示数字0~9的任何一个，“\d”就是元字符

```
public String replaceFirst()//替换第一次出现的字符串
```

字符串分割

```
public String[] split(String regex)//regex用于分割符号，返回的是一个数组
```

```
public String[] split(String regex, int limit)//limit是限制分割次数
```

字符串大小写转换

```
public String toUpperCase()//转换成大写
```

```
public String toLowerCase()//转换成小写
```

去除空白内容

```
public String trim()//不需要参数，只能去除收尾空格
```

可以使用replaceAll方法，使用\\s正则表达式，可以去除中间空格

格式化字符串

format是格式化关键字

```
String.format (String format, Object...args)//format格式化的公式，args被格式化的值
```

时间格式化

常規格式化

%d:数字格式化 (%nd , n为正时空格在左边，为负时空格在右边)

%c : 字符格式化

%f:浮点数格式化

%b : 布尔值格式化

%e : 科学计数法格式化

若要输出%，则应该使用两个%符号

.....

数字格式化

```
%nd//数字左对齐  
%-nd//数字右对齐  
%#o//8进制表示  
%#x//16进制表示  
%(d//区别正负数  
% d//同上  
%0nd//0是用来补位的  
,d//金额中的千分号格式
```

可变字符串

StringBuffer (可以类比于String学习)

```
StringBuffer sbf=new StringBuffer()  
StringBuffer sbf=new StringBuffer("ABC")  
StringBuffer sbf=new StringBuffer(32)
```

StringBuffer(32)|new String("32")

—|—

缓冲区|

初始字符序列容量为32个字符|初始字符序列值为 "32"

内存所占容量为32个字符|内存所占容量为默认的16个字符

改变值时是在原来的空间上改变|改变内容时开辟新的内存空间

修改指定索引处的字符

```
//实例  
StringBuffer sbf=new StringBuffer("0123456");  
sbf.setCharAt(3, 'A');
```

字符串反序

```
//实例  
StringBuffer sbf=new StringBuffer("0123456");
```

```
sbf.reverse();
```

删除子字符串

```
//实例
```

```
StringBuffer sbf=new StringBuffer("0123456");  
sbf.delete(3,5); //3代表开始位置的下标，5代表结束位置的下标（不包括）
```

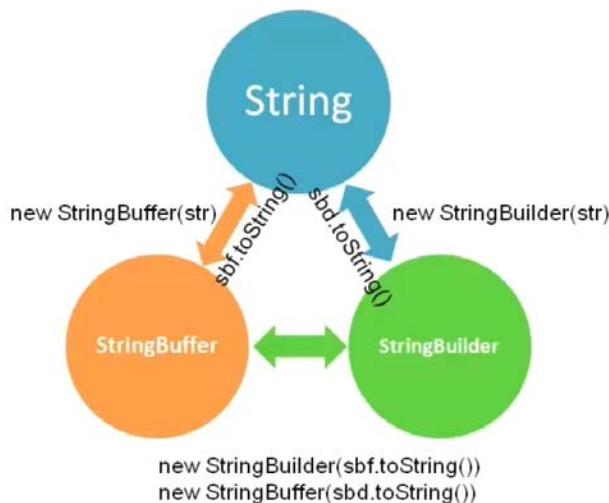
StringBuilder

StringBuilder方法与StringBuffer方法的使用方法完全相同，但是由于前者不保证线程的同步，所以在效率上会比StringBuffer会快一点



www.mingrisoft.com

➤ String、StringBuffer、StringBuilder之间的关系



类名|String|StringBuilder|StringBuffer

—|—|—

对象类型|字符串常量|字符串变量|字符串变量

线程安全性|不安全|不安全|安全

执行效率|低|高|中

面向对象

创建对象

让对象做事

特征：封装，继承，多态

封装：把数据和对数据的操作放在一起

成员变量：就是java中的属性（例如鸟类所具有的特点（属性），爪子，翅膀，羽毛等）

在函数中可以直接写成员变量的名字来访问成员变量

多态变量：

java的对象变量时多态的，它们能保存不止一种类型的对象

它们可以保存的是声明类型的对象，或声明类型的子类的对象

当把子类的对象赋给父类的变量的时候，就发生了向上造型

方法（函数）

返回值：使用return关键字

参数：

值参数

引用参数（比如数组）

不定长参数

函数调用的绑定

当通过对象变量调用函数的时候，调用哪个函数这件事情就叫做绑定

静态绑定：根据变量的声明类型来绑定

动态绑定：根据变量的动态类型来绑定

在成员函数中调用其他成员函数也是通过this这个对象变量来调用的

参数可以有多个，不限类型

每个参数都必须声明参数类型

调用方法时，注意参数顺序

成员方法：就是类的行为

构造方法

是一个与类重名的方法，对象的创建就是通过构造方法完成的，可以用来进行类的初始化操作

构造方法名与类名相同

构造方法没有返回值

私有构造方法

使用private修饰

无法用new实例化

局部变量

在成员方法内定义的变量，区别于类中定义的全局变量（使用时必须初始化）

不同区域内可以存在同名的局部变量

public static void main(String args[])什么意思？

public static void main(String[] args)

这绝对不是凭空想出来的，也不是没有道理的死规定，而是java程序执行的需要。

jvm在试图运行一个类之前，先检查该类是否包含一个特殊方法。这个方法必须是公有的，以便在任何位置都能访问得到。这个方法必须是static的，因为这个方法不能依赖任何该类的实例即可运行，而非static的方法，在运行之前要先创建该类的实例对象。

这个方法没有返回值。和C/C++程序不一样，java的程序默认都以常态结束，所以main不返回int。如果要以非常态结束程序，可以用System.exit(1)

这个方法必须可以接受数目不定的String类型的参数，因为运行者可能要附加运行参数。如java HelloWorld jack 100，这里的jack和100就是运行参数

为什么是String呢？因为String具有普遍性。任何字面形式都可以解释成String，而其他类型则不然（如jack就不能解释成整数或浮点数），所以用String来存储参数最合适。而因为参数数目不限一个，所以用了数组，即String[]。在Java 1.5以后，还可以写成String...，表示数目不定。

至于参数的变量名可以任意，只要保证在方法内部按该变量名来获得参数就行了，从变量的作用角度来说，叫arguments或args当然最合适。

至于这个方法的名字为什么一定是main，有历史原因，因为最早的C以main函数作为程序入口，java沿用了这个历史传统。

对象的创建

使用new关键字调用构造方法创建，特例是string对象可以之间创建

对象的类成员

属性

方法

通过对象.类成员的方式调用

对象的销毁

垃圾回收机制

两种对象消亡机制（只能回收new创建的）

```
{  
    Example e = new Example();  
}  
//局部程序结束，自动销毁  
  
{  
    Example e = new Example();  
    e=null; //对象被置为null，就被销毁  
}
```

finalize()方法

System.gc()强制启动垃圾回收器

匿名对象：没有任何栈内存使用，所以使用一次之后就等待被牢记收集机制回收

```
//主方法之中  
new 类(参数1, 参数2, .....).方法()
```

this关键字

代表类的本身，相当于类本身的一个对象

表示类中的属性（为类的属性赋值）

调用构造方法

```
this() //放在构造方法的第一行
```

表示当前对象

类的主方法

类的入口点，定义了程序从何处开始

```
public static void main(String[] args){  
}
```

静态

无返回值

参数是数组

主方法相当于是大门，通过它才能调用其他方法

类的封装

以类为载体进行封装

封装的思想

将对象的属性和行为封装起来，而将对象的属性和行为封装起来的载体就是类，类通常对客户隐藏其实现细节，只能通过具体的对象访问类的属性和行为

类的继承

extends关键字

```
Child extends Father;  
//例如  
public class Pad extends Computer{  
    ....  
}
```

方法的重写

super关键字

代表父类的对象

```
super.pwer()
```

```
super.ass
```

调用父类的构造方法

调用父类的属性

调用父类的方法

*java中，一个类只允许有一个父类

要想实现继承多个父类，需要使用多重继承的方法*

```
public class Parent1 extends Parent2;  
public class son extends Parent1;
```

子类不仅会覆盖父类的方法，还会覆盖父类的属性

object类

是类层次结构的根类

object类的三个方法

getClass():返回对象执行时的Class实例

toString():将对象返回为字符串形式

equal();比较两个对象是否相等（地址是否相等）

方法的重载

一个类可以有多个构造函数，只要它们的参数表不同

创建对象的时候给出不同的参数值，就会调用不同的构造函数

通过this()还可以调用其他构造函数

一个类里的同名但参数表不同的函数构成了重载关系

怎样构成方法的重载

方法名相同参数不同

方法名相同，参数顺序不同

方法名相同，参数类型不同

在String的API文档中，string的定义利用了方法重载，将不同类型的方法转换为字符串

类的上下转型（造型cast）

类的向上转型（父类声明一个对象，由子类进行实例化）

```
Father a =new Child;
```

类的向下转型

```
Parents P=new Parents();
Child c=(Child)p;//强制将父类对象转化为子类对象
```

造型不同于类型转换：对象本身并没有发生任何变化

向上造型：拿一个子类的对象当做父类的对象来用；向上造型是默认的，不需要运算符；向上造型总是安全的

instanceof关键字

```
boolean result =child instanceof parents//判断子类是否继承自父类，两个没有继承关系的类会直接报错
```

抽象类

```
abstract class 类名 //在不清楚类的实际情况的前提下创建的类，不能实例化，必须用它的子类进行实例化
```

子类继承了抽象类之后，需要实现这个抽象类和它的父类的所有的抽象方法

抽象方法

定义在抽象类中的方法，抽象方法没有具体实现（没有大括号），必须由抽象类的子类去实现

接口

接口同样支持类的向上向下转型

两个类不存在继承关系，可以通过接口实现同一个抽象方法（不是通过抽象类进行控制）

接口的多重继承

接口继承接口（接口可以继承另外的接口）

```
interface intf1

{

}

interface intf2 extends intf1

{



}
```

类实现接口（类可以实现多个接口）

```
class 类名 implements 接口1, 接口2, ...., 接口n
```

一个类只能继承一个父类，但是一个类可以继承多个接口

接口与抽象类的区别

接口	抽象类
接口是对动作的抽象	抽象类是对根源的抽象（人类，鸟类等）
子类可以实现任意多个接口	子类只能继承一个抽象类
接口中的方法都是抽象方法	抽象类可以有非抽象方法
接口中的成员变量只能是静态常量（final static）	抽象类中的成员变量可以是各种类型
接口不能有静态方法和静态代码块	抽象类可以
接口没有构造方法	抽象类有

Java类包

不能调用两个包下的同名类

访问控制

访问控制符

	public 公有	protected 受保护	default 缺省	private
本类	可见	可见	可见	可见
本类所在包	可见	可见	可见	不可见
其他包中的子类	可见	可见	不可见	不可见
其他包中的非子类	可见	不可见	不可见	不可见

final关键字

final类

```
final class 类名{}
```

final类不能被继

final方法

不能被重写

final常量

不能被修改

内部类

在类体里面定义了一个类，这个类叫做内部类

成员内部类（定义在类内的内部类）

局部内部类（定义在成员方法内的内部类）

匿名内部类（在使用过程中才进行编写的内部类，没有名字）

静态内部类（提供一个调试的功能，可以将main方法写在静态内部类中）

内部类的继承

```
class ClassA{  
    class ClassB{  
    }  
}  
class OutputInnerClass extends ClassA.ClassB{  
}
```

内部类能直接访问外部的全部资源

包括任何私有的成员

外部是函数时，只能访问那个函数里面final的变量

异常处理

异常是会影响程序正常运行，但是可以被解决的问题

异常的分类

Exception

RuntimeException:运行时异常

Error：不应试图捕获的严重问题

java使用try catch语句处理异常，不能用来处理Error

语法

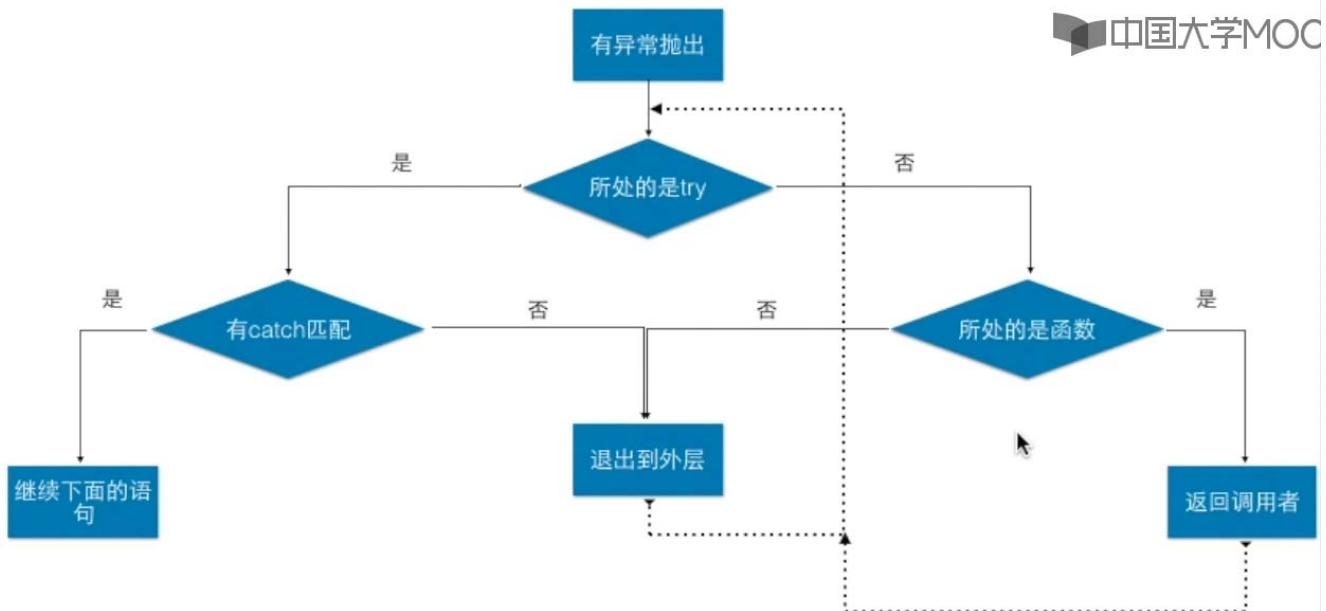
```
try{  
    //程序体  
}catch(Exception e){//catch内是异常类型  
    e.//打印等  
}
```

捕捉异常

自动捕捉异常

try catch捕捉异常

```
try{
    //被捕获的代码
}catch(异常类型 e){
    //对异常的处理
}catch(some Exception1 e){
}catch(soem Exception2 e){
}//.....多重try catch代码
```



异常的中断机制

发生异常后，异常之后的代码就不会再执行了

恢复机制

finally代码块（一般情况下都会执行）

```
try{
    //被捕获的代码
}catch(异常类型 e){
    //异常处理
}finally{
    //最后一定会执行的代码
}
```

`finally`不会执行的四种情况

`finally`块中发生异常

程序所在线程死亡

在前面的代码中使用了`System.exit()`语句

关闭cpu

在方法中抛出异常

`throw`关键字：手动制造一个异常;更改一个异常

```
throw new 异常类型();
```

`throws`关键字：将代码中可能产生的异常交给别人处理

```
public void method() throws 异常类型1, 异常类型2.....异常类型n{//也可以在main方法处抛出这些异常}
```

```
}
```

调用这个方法时，必须使用`try catch`语句进行异常捕捉（三种方式）

main方法后抛出异常

多次抛出异常

一次抛出多个异常

抛出异常要慎重，自己能处理的异常不要抛出，自己处理不了的一定要抛出

自定义异常

自己创建一个API中没有的异常

```
class 自定义异常类 extends 已有的异常类{
```

```
}
```

异常使用原则

不要忽略捕捉到的异常

不要过度使用异常

不要使用过于庞大的try-catch块

子类抛出的异常不能比父类更高级，包含于父类的异常（`RuntimeException`例外）

异常声明

如果函数可能抛出异常，就必须在函数的头部加以声明

可以声明并不会真的抛出的异常

异常声明遇到继承关系

当覆盖一个函数的时候，子类不能声明抛出比父类的版本更多的异常

在子类的构造函数中，必须声明父类可能抛出的全部异常

运行时刻异常

像`ArrayIndexOutOfBoundsException`这样的异常是不需要声明的

但是如果没有适当的机制来捕捉，就会最终导致程序终止

包装类

包装类的变量可以转换为其对应基础类型的变量

也可以使用`valueOf`创建对象

数据类型通过包装类转换为对象供java进行处理

Integer

用于封装int类型数据

```
Integer(int number)  
Integer(String str)  
Integer.valueOf(String str)
```

double

```
Double(int number)
```

```
Double(String str)
```

```
Double.valueOf(数字或字符串)
```

++如果基本的整数和浮点数不能满足精度需求，可以使用java.math包中的BigInteger类（实现任意精度的整数运算）和BigDecimal类（任意精度的浮点数运算）

但是要使用add和multiply方法

同时，使用valueOf方法可以将普通的数值转换为大数值++

Boolean

```
Boolean(Boolean value)
```

```
Boolean(String str)
```

Boolean值默认为false

Character

```
Character(char value)
```

Number类

doubleValue

intValue

byteValue

floatValue

自动装箱和自动拆箱

装箱和拆箱就是包装类和基本数据类型的相互转换

一般方法

```
Integer i=new Integer(100); //装箱
```

```
int i=num.intValue(100); //拆箱
```

自动方法

```
Integer num=100;//装箱  
  
int i=new Integer(100);//拆箱
```

特殊情况

JVM会将相等的byte值和Boolean值保存在同一个对象

Math类

java.lang.Math

java.lang包不需要引用，系统会自动调用

Math类提供的方法都是静态方法

```
Math.round(x)=(int)Math.floor(x+0.5F);  
  
//Math.floor向下取整  
  
//Math.round四舍五入
```

Math.random

返回一个大于0小于1的随机double值

随机数 Random类

```
Random r=new Random();//实例化对象  
  
//方法  
  
nextInt(int n);//返回随机生成的int值，在0到n之间，小于n  
  
nextLong();  
  
nextDouble();  
  
nextFloat();  
  
nextBoolean();
```

Date类

java.util.Date

```
Date date=new Date()  
Date date=new Date(long time)
```

Dateformat类

输出指定格式的时间

```
Dateformat format=new SimpleDateFormat("yyyy-MM-dd");//使用子类进行实例化
```

getTimeZone方法：获取时区

setTimeZone方法：设置时区

Calendar类

垃圾！没有12月，是从0月开始！垃圾垃圾垃圾！

```
Calendar rightNow=Calendar.getInstance();
```

方法：

set():设置日历字段

get()：获取日历字段

add()/roll():添加时间量，前者会自动向前进位，后者不会进位

集合的概述

集合类就像一个容器，相当于一个动态数组

集合类总共有两大接口：

Collection接口：元素集合

List接口：有序元素集合

ArrayList

LinkedList

Set接口：无序元素集合

HashSet

Map : 键值对集合

HashMap

HashMap性能更好

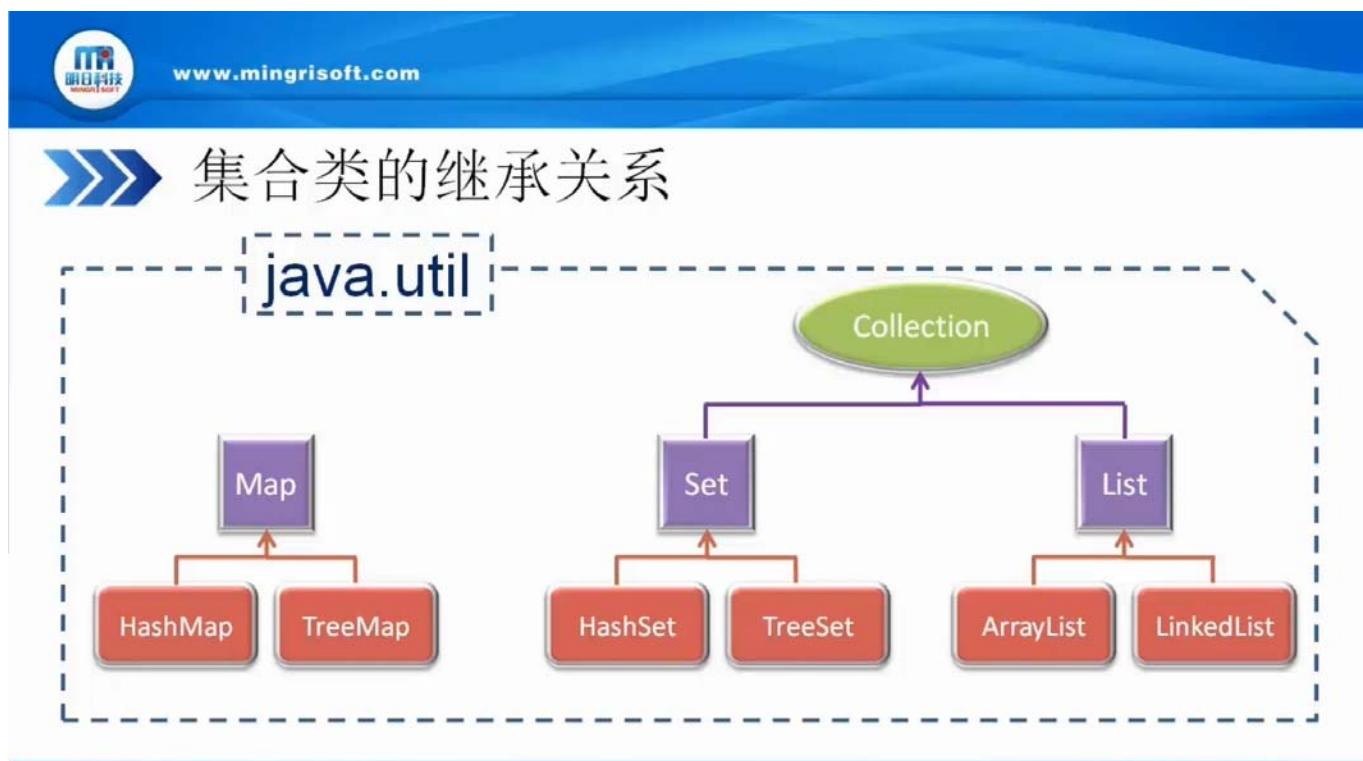
HashTable

HashTable是线程安全的

Treemap

树状键值对存放有序

集合类的继承关系



Collection接口

方法

`isEmpty()`

`size()`

`add()`

`remove()`

`iterator()`: 遍历集合中的元素并依次取出

以上这些方法collection的所有子类都可以直接使用

List集合 (动态更改)

创建List对象

```
List<E> list=new ArrayList<E>()  
List<E> list=new LinkedList<E>()//<E>是泛型（可以省略）
```

方法

add()

remove()

get()

set()

创建对象时，一般使用类本身或者子类进行实例化

一个特点就是存在索引，可以通过索引查询元素

Iterator迭代器

对集合 (collection) 进行迭代

```
Iterator<E>i=list.iterator();//list是指collection的任意子类
```

方法

hasnext():如果有可迭代的元素，则返回true

next():返回迭代的下一个元素，object值

特点在于可以遍历集合中的所有元素，并且执行效率很高

set集合 (set接口)

重复的元素不能被添加到set集合中

HashSet

HashSet可以添加null值

TreeSet

TreeSet不能添加null值

.java

```
Set set=new TreeSet();
```

~

方法

add()

remove()

contains(Object o)

iterator()

HashSet

```
Set<E>set=new HashSet<E>();
```

HashSet存储的对象应该重写hashCode()和equals()两个方法

排列顺序根据Hash值进行分配

当一个元素添加到集合之后，不要修改可能会让其Hash值改变的属性

HashSet是根据hash值来存放元素的

同一个hash地址可以存放多个不同的对象

Map集合

键值映射

使用Key映射Value：



Map接口

键(key) 值(value)
Map<K, V>

put(K key, V value)

- 将键值数据存入Map中

containsKey(Object key)

- 查找Map中是否存在某个键

get(Object key)

- 通过键，获取值

containsValue(Object value)

- 查找Map中是否存在某个值

```
Map<K, V>m=new HashMap<K, V>();
```

```
Map<K, V>m=new TreeMap<K, V>();
```

hash表中的所有value都是对象

对hash表来说，键值对一定是唯一的，如果多次存入，那么保存下来的一定只有最后一个

三种集合的应用

List关心索引

Set关心唯一性，不允许重复

Map关心的是唯一的标识符，键值对

常量

创建枚举

```
public enum 枚举类名 { //enum 代表一个枚举类型，类似 class, interface }
```

枚举1, //默认为final public static类型

枚举2,

枚举3,

.....

}

不会存在像普通常量一样多个常量引用同一个值的问题

枚举类型的常用方法

values():可将枚举类型的成员以数组的形式返回

~java

```
Constants enumArray[] = Constants.values();
```

~

valueof():可将普通字符串转换为枚举实例

~java

```
public enum Constants{  
    Constants_A;  
}
```

Constants c=Constants.valueOf("constants_A")//将字符串转换为枚举类型。创建的值一定要真实存在

~

compareTo() : 比较两个枚举对象在定义时的顺序

ordinal() : 用于得到枚举成员的位置索引

创建枚举的类成员

枚举类型的常量成员必须在其他成员之前定义，否则这个枚举类型不会产生对象

构造方法必须是private修饰或者无修饰符

private:

只有这个类的内部可以访问

类内部指的是类的成员函数和定义初始化

这个限制是对类的而不是对对象的

public :

都可以访问

不带public和private

同一个包内可以访问

一个.java文件是一个编译单元

注意：在枚举中添加了属性或者方法的话，在最后一个枚举成员之后要加上；

枚举实现接口

```
public enum 枚举名称 implements 接口名称{}
```

枚举类型可以实现一个或多个接口，但是不能继承类。枚举默认继承java.lang.Enum类

如果枚举成员没有实现接口的抽象方法，则需要在枚举下来调用实现父类的抽象方法，这样的话，之前实现抽象方法的枚举成员就相当于是重写了这个方法

泛型

可以定义安全的数据类型

```
类名<类型参数1, 类型参数2, ...., 类型参数n>{}
```

泛型继承类或接口

```
A<T extends anyClass>a;
```

泛型通配符

是在声明对象时使用，而不是在创建类的时候使用

限制泛型类型，并可以限制泛型对象的使用

```
A<?>a;  
A<? extends anyClass> a;//使用extends关键字指定了上界  
A<? super anyClass> a;//使用super关键字指定了下届
```

继承泛型类和泛型接口

```
class ExtendClass<T>{  
}//定义泛型  
  
class SubClass<T> extends ExtendClass<T>{  
}//继承泛型
```

```
interface TestInterface<T>{
} // 定义泛型接口

class SubClass2<T> implements TestInterface<T>{
} // 实现泛型接口
```

继承泛型的四种情况

```
// 第一种情况

abstract class Father<T1, T2>{}

// 全部继承

class Child<T1, T2, T3> extends Father<T1, T2>{} // 子类可以在继承父类泛型的同时，拥有自己的泛型

// 部分继承

class Child<T1, A, B> extends Father<T1, String>{} // 父类实现了一个泛型（对父类的一个泛型实例化了），子类可以拥有多个泛型

// 实现父类泛型

class Child<A, B> extends Father<Integer, String>{} // 子类在继承父类泛型的同时，将父类的所有泛型全部直接实现了，子类的泛型就全部是自己独有的

// 不实现父类的泛型

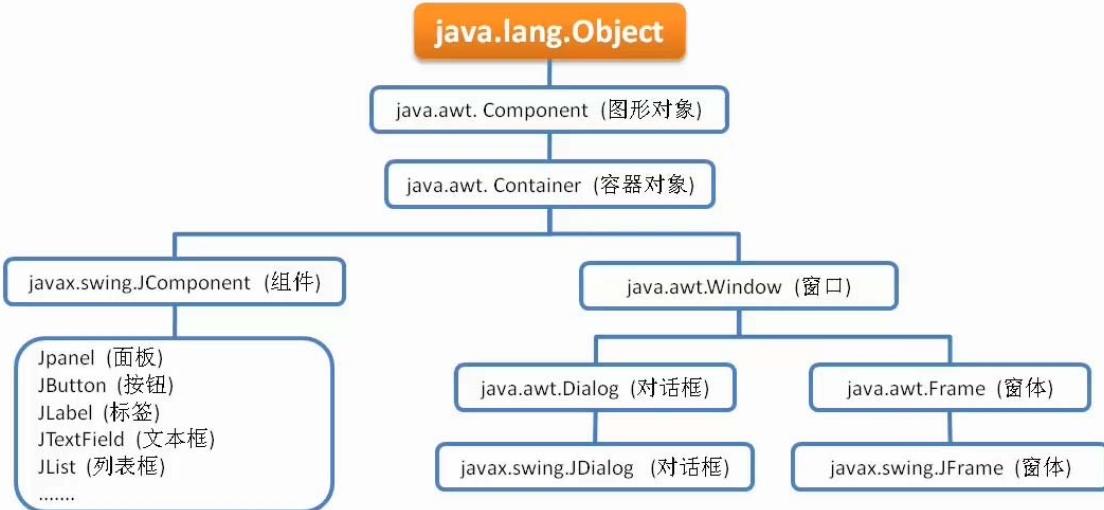
class Child extends Father()
```

Swing

用于开发图形用户界面的包



窗体组件类结构



窗体 (JFrame)

对话框窗体 (JDialog)

标签 (JLabel)

主要作用就是展示一段文字

如何在窗体里添加图片

使用label标签

绝对布局 (null布局)

硬性指定了组件在容器中的位置和大小

使用绝对布局的窗口通常都是固定大小，组件位置和形状不会随着窗体的改变而发生变化

流布局 (FlowLayout)

从左到右排列，默认居中对齐

像流水一样，向某个方向流动，遇到障碍就折回

边界布局 (borderLayout)

窗体容器的默认布局

添加组建时，需要指定区域，否则默认添加到CENTER区

同一区域的组件会相互覆盖

网格布局 (GridLayout)

面板 (JPanel)

窗体中可以有多个容器，容器之间互不干扰

滚动面板 (JScrollPane)

按钮 (JButton)

单选按钮 (JRadioButton)

复选框 (JCheckBox)

下拉框 (JComboBox)

列表框 (Jlist)

文本框 (JTextField)

密码框 (JPasswordField)

文本域 (JTextArea)

动作事件监听器(ActionListener)

```
addActionListener(ActionListener) {  
    @Override
```

```
public void actionPerformed(ActionEvent arg0) {//这里的ActionEvent会捕  
捉动作触发者，返回arg0参数  
  
// TODO Auto-generated method stub  
  
}  
  
}
```

焦点事件监听器 (FocusListener)

```
void focusGained(FocusEvent e)//组件获取焦点时调用的方法  
void focusLost(FocusEvent e)//组件失去焦点时调用的方法
```

可以使用自定义实现焦点监听器类，简化代码

键盘事件

KeyListener键盘监听（接口）

方法

```
public void keyTyped(KeyEvent e); //发生击键时触发  
public void keyPressed(KeyEvent e); //按键被按下时触发  
public void keyReleased(KeyEvent e); //按键被释放时触发  
  
//KeyEvent表示键盘事件类
```

鼠标事件

MouseListener鼠标事件监听（接口）

```
public void MouseEntered(MouseEvent e); //鼠标移入组件时被触发  
public void MousePressed(MouseEvent e); //鼠标按键被按下时触发  
public void MouseReleased(MouseEvent e); //鼠标按键被释放时触发  
public void mouseClicked(MouseEvent e); //发生单击事件时被触发  
public void mouseExited(MouseEvent e); //鼠标移除组件时被触发  
  
//MouseEvent表示鼠标事件类
```

窗体焦点事件

WindowFocusListener窗体焦点监听

```
public void windowGainedFocus(WindowEvent e)//窗体获得焦点时被触发
```

```
public void windowLostFocus(WindowEvent e)//窗体失去焦点时被触发
```

窗体状态事件

WindowStateListener窗体状态监听 (接口)

```
public void windowStateChanged(WindowEvent e)//窗体状态发生变化时被触发
```

```
getNewState()//获得窗体现在的状态
```

```
getOldState()//获得窗体以前的状态
```

窗体事件

WindowListener (接口)

The screenshot shows the Mingrisoft website with a blue header. Below the header, there is a large title 'WindowListener 窗体事件监听 接口' with a blue arrow icon. To the left of the title is a list of seven methods, each with a corresponding description to its right.

public void windowActivated(WindowEvent e);	窗体被激活时触发
public void windowOpened(WindowEvent e);	窗体被打开时触发
public void windowIconified(WindowEvent e);	窗体被最小化时触发
public void windowDeiconified(WindowEvent e);	窗体恢复为正常大小时触发
public void windowClosing(WindowEvent e);	窗体将要被关闭时触发
public void windowDeactivated(WindowEvent e);	窗体不再处于激活状态时触发
public void windowClosed(WindowEvent e);	窗体已经被关闭时触发

点击关闭时不会触发失去激活状态

选项事件

ItemListener选项事件监听 (接口)

```
public void itemStateChanged(ItemEvent e); //在用户选定或取消某项时触发的事件
```

ItemEvent选项事件

```
public object getItem() //选中返回事件  
public int getStateChange() //返回选中的状态  
public object getSource() //返回触发选项事件的组件
```

分割面板 (JSplitPane)

JSplitPane构造方法

```
JSplitPane()  
JSplitPane(int newOrientation) //指定分割方向 (水平 JSplitPane.HORIZONTAL_SPLIT 或垂直 JSplitPane.VERTICAL_SPLIT)  
JSplitPane(int newOrientation, boolean newContinuousLayout) //指定是否重绘
```

选项卡面板 (JTabbedPane)



》 JTabbedPane构造方法



`setTabPlacement`方法 // 设置选项卡标签位置

`setTabLayoutPolicy`方法 // 设置选项卡面板布局方式

桌面面板和内部窗体

JDesktopPane类：表示虚拟桌面

JInternalFrame类：表示内部窗体

菜单栏 (JMenuBar)

创建菜单栏步骤

 创建菜单栏对象 (JMenuBar类)

 创建菜单对象(JMenu类)

 创建菜单项对象(JMenuItem类)

弹出式菜单 (JPopupMenu)

添加弹出式菜单

```
JPopupMenu popupMenu=new JPopupMenu();//新建弹出式菜单对象  
//添加鼠标事件监听  
//调用对应的鼠标事件方法  
//判断是否为该组件的弹出菜单触发事件  
//判断在哪个组件中显示以及显示的位置
```

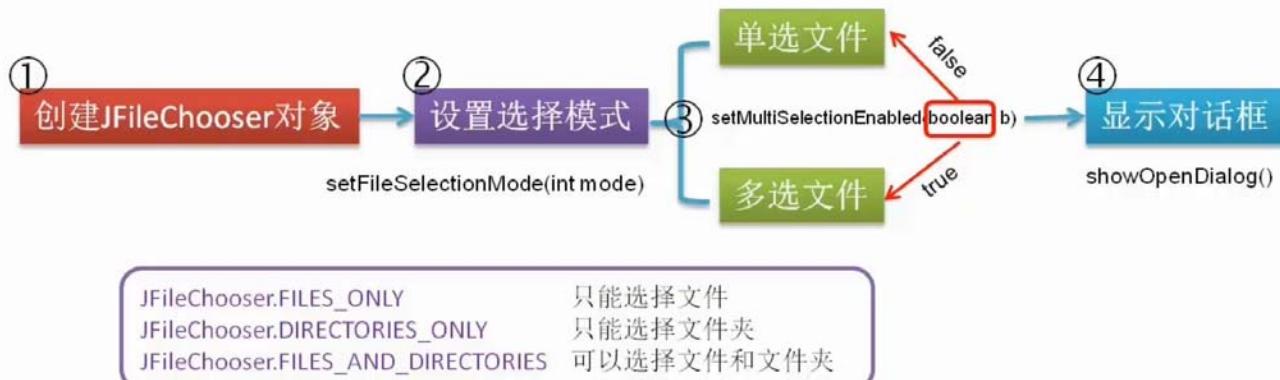
工具栏(JToolBar)

文件选择器(JFileChooser)



文件选择器

javax.swing.JFileChooser类



过滤文件

抽象类 **abstract class FileFilter类**

→ FileNameExtensionFilter类

语法

FileNameExtensionFilter(String description, String...extensions)

文件类型描述信息

文件类型

示例

```
FileFilter filter = new FileNameExtensionFilter("图像文件", "JPG", "PNG", "GIF");
fileChooser.setFileFilter(filter); // 文件选择器添加过滤器
```

读文件

打开文件

判断文件大小

分配足够的内存空间

把文件读入内存

关闭文件

进度条 (JProgressBar)

```
setIndeterminate(true)//不确定进度条  
setIndeterminate(false)//确定进度条（默认）  
setStringPainted(true)//设置显示提示信息  
    setValue(66)//设置进度条进度值  
    setString("66%")//设置进度条显示的文本
```

表格

创建表格

www.mingrisoft.com

创建表格

语法 `JTable(Object[][] rowData, Object[] columnNames)`

表格中的数据

列名集合

设置选中行字体颜色
`setSelectionForeground(Color selectionForeground)`

设置选中行背景色
`setSelectionBackground(Color selectionBackground)`

设置选择模式
`setSelectionMode(int selectionMode)`

设置选择模式
ListSelectionModel.MULTIPLE_INTERVAL_SELECTION 随便选
ListSelectionModel.SINGLE_INTERVAL_SELECTION 连选
ListSelectionModel.SINGLE_SELECTION 单选

表格模型(TableModel)

TableModel接口

AbstractTableModel抽象类

DefaultTableModel类

添加行 : addRow()/insertRow()

修改行 : setValueAt()

删除行 : removeRow()

JTable使用表格模型

构造方法中直接传入表格模型

调用设置表格模型的方法

表格模型事件

TableModelListener表格模型事件监听

```
tableModel.addTableModelListener(TableModelListener listener);
```

```
public void tableChanged(TableModelEvent e);
```

TableModelEvent表格模型事件

getColumn()返回触发事件的列//TableModelEvent.INSERT插入事件

getFirstRow()返回第一个被更改的行//TableModelEvent.UPDATE修改事件

getType()返回事件类型//TableModelEvent.DELETE删除事件

数据库基础

使用JDBC操作数据库的基本步骤

加载JDBC驱动程序 (DriverManager类)

连接数据库 (connection类)

发送SQL语句

Statement接口

PreparedStatement接口继承自Statement接口

但是不同于Statement，这个的初始化使用的是prepareStatement关键字

CallableStatement接口继承自PreparedStatement接口

不同于PreparedStatement，这个的初始化使用的是prepareCall关键字

处理结果集 (ResultSet接口)

关闭数据库

连接数据库

加载驱动程序

连接数据库

数据库查询

创建接口对象

```
Statement stmt=con.createStatement(); //con: Connection对象  
ResultSet res=stmt.executeQuery("select * from tb_stu"); //res: 返回结果集对象, executeQuery: 执行SQL
```

使用jdbc添加，修改和删除数据

executeUpdate

插入 insert

删除 delete

修改 update

批处理

语法

```
void addBatch(String sql) //将sql语句放入执行列表中; sql参数通常为INSERT或者UPDATE语句  
int[] executeBatch() //执行列表中的sql语句; executeBatch方法返回每一条sql所影响的行数的一个int型的数组
```

动态查询

避免了sql注入的问题

调用存储过程

CallableStatement接口

I/O流概述

输入流：数据从起始地出入程序

输出流：数据从程序传入目的地



File类

代表磁盘文件或者文件夹

```
File(String Pathname)//通过将路径名字符串转换为抽象路径名来创建一个新file实例
```

```
File(String parent,String child)//根据parent路径名字符串和child路径名字符串创建一个新file实例
```

```
File(File parent,String child)//根据parent抽象路径名和child路径名字符串创建一个新file实例
```

项目下的路径（默认路径）：word.txt

包中的文件路径：src//mr/word.txt

注意：/表示文件夹

\表示文件夹（转义字符）

绝对路径：C/test/word.txt

File类的使用

操作：创建文件，删除文件

状态：文件是否存在，是否隐藏

属性：文件名，绝对路径，文件大小，文件修改时间

文件夹的操作

创建，删除，判断是否存在，判断是否为文件夹，获取所有子文件和子文件夹

`mkdir()`：创建文件夹

`mkdirs()`：创建多层文件夹

删除文件夹时指的是删除文件路径最后一个文件夹

文件字节流

`FileInputStream`文件字节输入流

`FileOutputStream`文件字节输出流

文件字符流

避免了读取数据不全造成数据不全问题

`FileReader` 文件字符输入流

`FileWriter` 文件字符输出流

`InputStreamReader`类的子类。所有的方法都从父类中继承而来

```
FileReader(File file)//在给定从中读取数据的File的情况下创建一个新的FileReader
```

```
FileReader(String filename)//在给定从中读取数据的文件名的情况下创建一个新FileReader
```

汉字编码

```
InputStreamReader(InputStream in)//创建一个使用默认字符集的InputStreamReader
```

```
InputStreamReader(InputStream in,Charset cs)//创建使用给定字符集的InputStreamReader
```

```
InputStreamReader(InputStream in,CharsetDecoder dec)//创建使用给定字符解码器的InputStreamReader
```

```
InputStreamReader(InputStream in,String charsetName)//创建使用指定字符集的InputStreamReader
```

缓冲字节流

BufferedInputStream 缓冲输入流

BufferedOutputStream 缓冲输出流

当存在其他字节流时，可以用缓冲字节流将其包裹起来，可以提高程序的执行效率

使用缓冲字节输出流时，要多使用刷新操作，否则缓冲区会等待缓冲区填满后才写入文件中

缓冲字符流

可以以行为单位进行输入/输出

BufferedReader : 缓冲字符输入流 public String readLine()

BufferedWriter : 缓冲字符输出流 public void newLine()

流的关闭顺序，先创建的后关闭

PrintWriter (在流上建立文本处理)

```
PrintWriter pw=new(PrintWriter  
    new BufferedWriter(  
        new OutputStreamWriter(  
            new FileOutputStream("a.txt"))))
```

数据流

可以从流中读取或写入java基本数据类型

DataInputStream 数据输入流

DataOutputStream 数据输出流

写入数值类型时尽量分开写

字节流转换为字符流

```
InputStreamReader(InputStream in,String charsetName)  
OutputStreamWriter(OutputStream out,String charsetName)  
//charsetName表示字符集名称 GBK UTF-8等
```

最大的特点是可以指定字符集

流的两种关闭方式

使用close()关闭

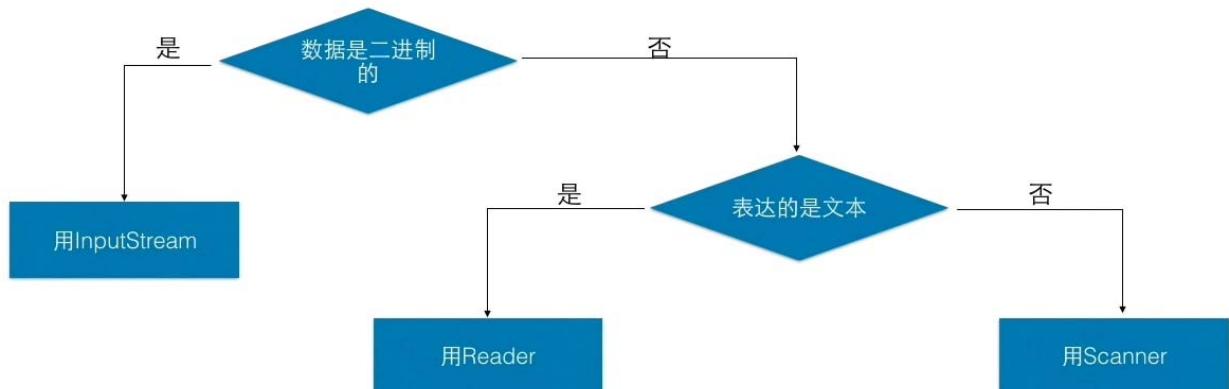
使用try语句自动关闭

流过滤器

以一个介质流对象为基础层层构建过滤器流，最终形成的流对象能在数据输入输出的过程中，逐层使用过滤器流的方法来读写数据

Stream/Reader/Scanner

中国大学MOC



Class类

Object obj=new Object();

一定要使用new关键字

类必须存在，否则编译不能通过

可能需要引入包

阻塞/非阻塞

read()函数是阻塞的，在读到所需的内容之前会停下来等

使用read()的更“高级”的函数，如nextInt(),readLine()等也是一样的

所以常用单独的线程来做socket读的等待，或使用nio的channel的选择机制

对于socket，可以设置SO时间

```
setSoTimeout(int timeOut)
```

反射的本质

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法；这种动态获取的以及动态调用对象的方法的功能称为java语言的反射机制

Java反射机制主要提供了以下功能

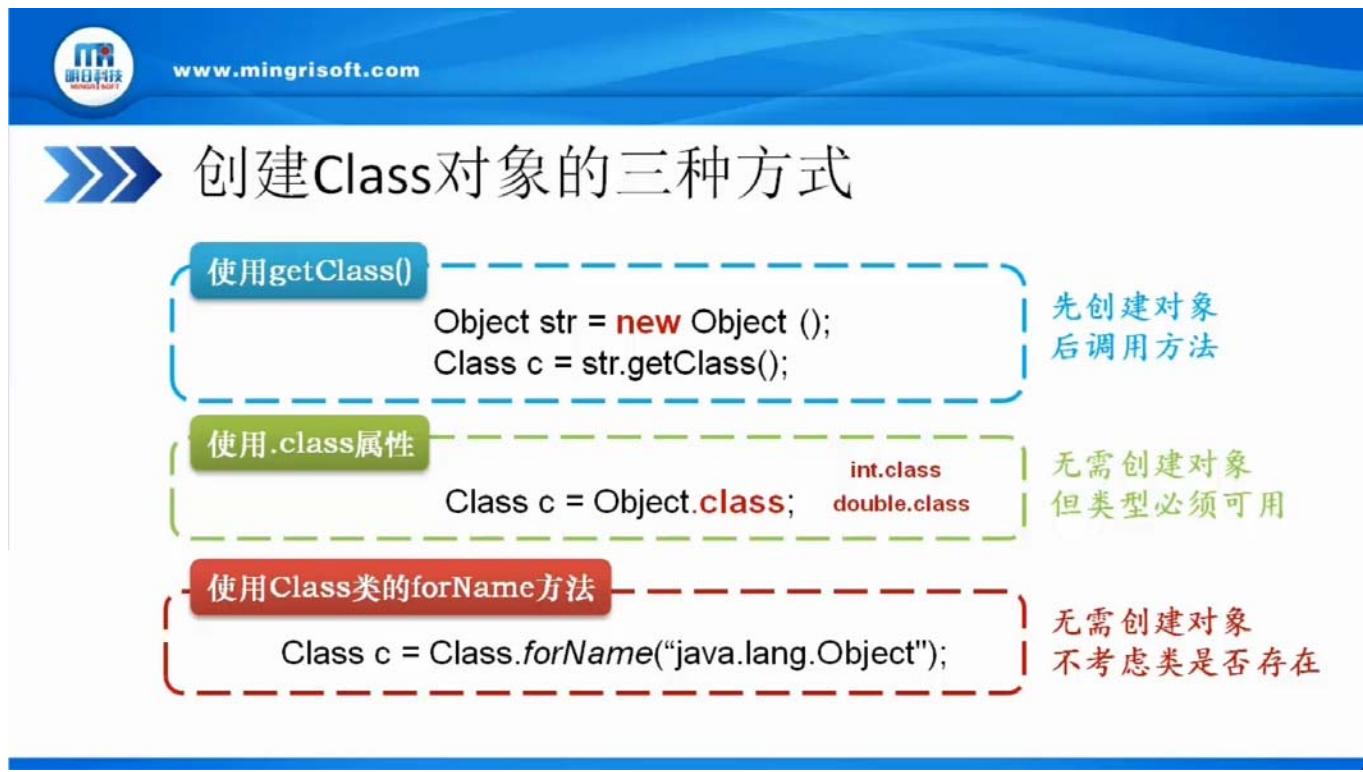
在运行时判定任意一个对象所属的类

在运行时构造任意一个类的对象

在运行时判定任意一个类所具有的成员变量和方法

在运行时调用任意一个对象的方法

生成动态代理



hashcode和equals相等的关系

`equals`返回为true，则两者的`hashcode`一定相等，意即相等的对象必须具有相等的哈希码。每当`equals`方法被覆写，通常需要重写`hashCode`方法从而保持对象行为的一致性

而两个对象的`hashcode`相等，`equals`方法不一定返回true。只有同一类下的不同对象这种情况成立

获取构造方法

Constructor构造方法类：

- * `getModifiers()//`构造方法的修饰符
- * `getName()//`构造方法的名字
- * `getParameterTypes()//`构造方法的参数类型

获取构造方法

```
class.getConstructors()//获取公有的构造方法  
class.getDeclaredConstructors()//获取所有的构造方法  
class.getConstructor(Class<?>...parameterTypes)//获取指定的公有构造方法  
class.getDeclaredConstructors(Class<?>...parameterTypes)//获取指定的构造方法
```

获取成员变量

Filed成员变量类

```
getModifiers()//成员变量的修饰符  
getName()//成员变量的名字  
getType()//成员变量的声明类型
```

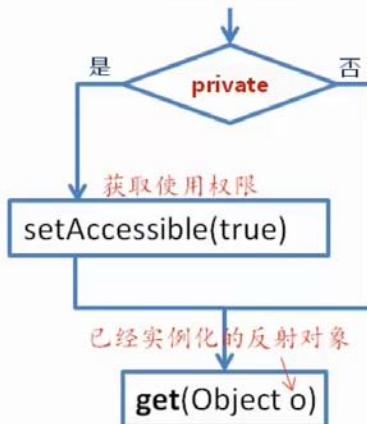
获取成员变量

```
class.getFields()//获取公有的成员变量  
class.getDeclaredFields()//获取所有的成员变量  
class.getField(String name)//获取指定的公有成员变量  
class.getDeclaredField(String name)//获取指定的成员变量
```



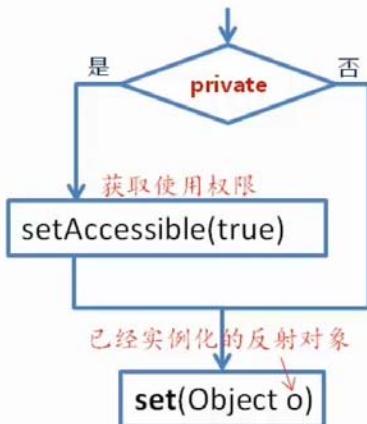
》》》 获取和修改成员变量的值

获取变量值



getBoolean(Object obj)
getByte(Object obj)
getChar(Object obj)
getDouble(Object obj)
getInt(Object obj)
getLong(Object obj)
getShort(Object obj)

修改变量值



setBoolean(Object obj)
setByte(Object obj)
setChar(Object obj)
setDouble(Object obj)
setInt(Object obj)
setLong(Object obj)
setShort(Object obj)

java可以自动生成无参数的构造方法

获取成员方法

Method成员方法类

```
getModifiers()//成员方法的修饰符  
getName()//成员方法的名字  
getType()//成员方法的声明类型  
getParameterTypes()//获取方法的参数数组
```

获取成员方法

```
class.getMethods()//获取公有的成员方法  
class.getDeclaredMethods()//获取所有的成员方法  
class.getMethod(Class<?>...parameterTypes)//获取指定的公有成员方法  
class.getDeclaredMethod(Class<?>...parameterTypes)//获取指定的成员方法
```

forName方法

forName是一个静态方法，其作用：通过调用类的静态方法来获取类名对应的Class对象，同时将Class对象加载进来。如果将类名保存在字符串（如xml）中，就可以在程序运行时，动态调用加载。

newInstance方法

将对象实例化。返回类型为Object。与new的区别在于，new可以带参，而newInstance()不可以，一般初始化无参类。通常与forName()配合使用

getMethod方法

getMethod方法则根据方法名称和相关参数，来定位需要查找的Method对象并返回

>

invoke方法

调用包装在当前Method对象中的方法

内置注解

名称	作用	作用范围
@Override	限定重写父类方法	成员方法
@suppressWarnings	抑制编译器警告	类，成员属性，成员方法
@Deprecated	标识已过时	类，成员属性，成员方法

自定义注解

反射注解



元注解

为其他注解做注解

@Documented 指示某一类型的注释通过 javadoc 和类似的默认工具进行文档化。

@Inherited 指示注释类型被自动继承。

@Retention 指示注释类型的注释要保留多久。

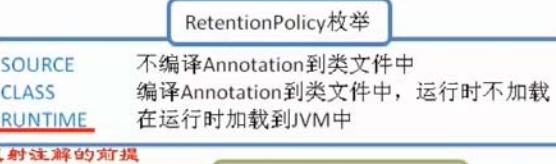
@Target 指示注释类型所适用的程序元素的种类。



元注解

@Retention

使用的枚举



@Target

使用的枚举



反射注解

Constructor (构造方法类)

isAnnotationPresent(Class annotationClass)

查看是否添加了指定的注解

Field (成员属性类)

getAnnotation(Class annotationClass)

获取指定的注解

Method (成员方法类)

getAnnotations()

获取所有注解的数组

前提

@Retention(RetentionPolicy.RUNTIME)

线程

Thread类

```
Thread t=new Thread();
```

```
Thread t=new Thread(String name);
```

run() : 线程运行时执行的方法

start():使线程开始执行的方法

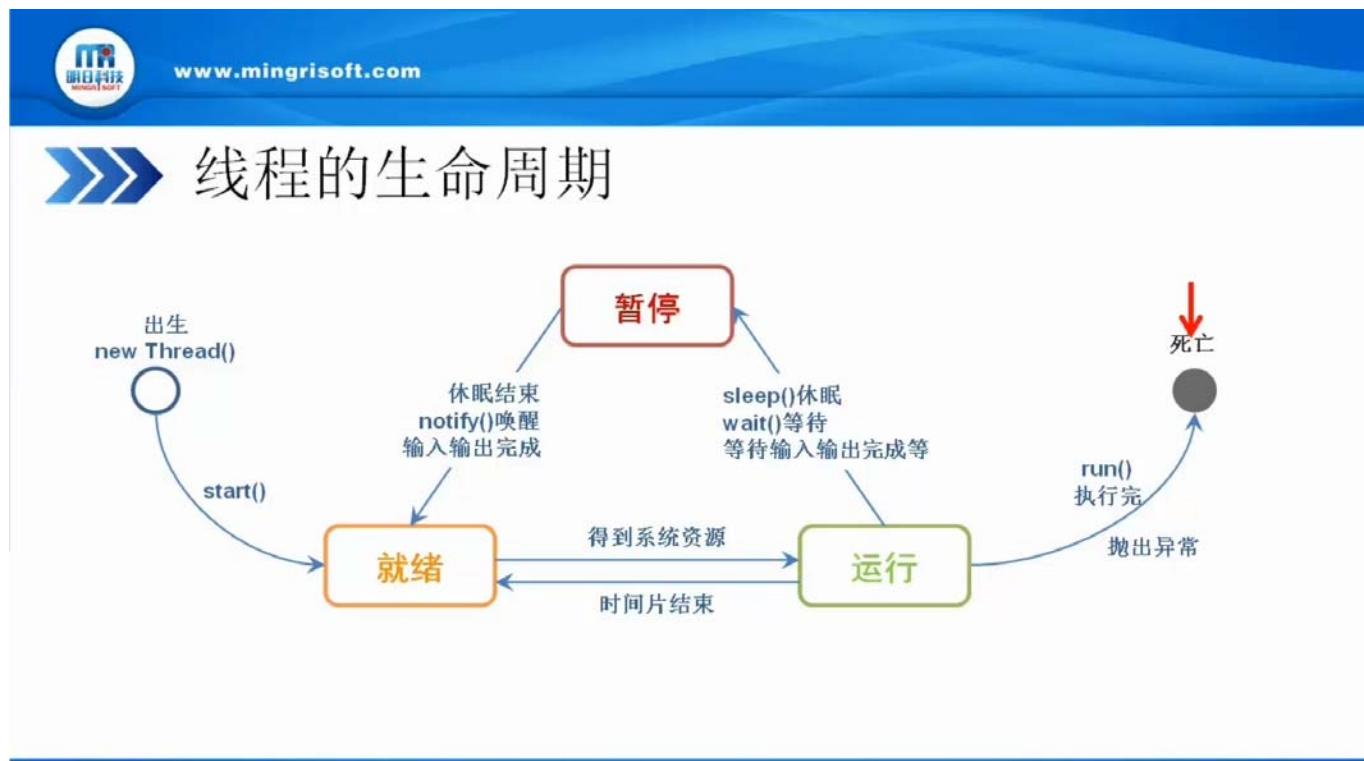
线程的执行顺序和执行时间是由CPU控制的，而不是代码控制

Runnable接口

若是使用Thread类创建线程的话，这个类只能继承Thread类，不能再继承其他类，实现其他功能。所以，可以使用Runnable接口，使这个类实现Runnable方法，在此基础上实现其他类

```
public class ThreadTest extends Object implements Runnable{  
    public void run(){}
}
```

线程的生命周期



线程的休眠

Thread.sleep(), 停顿但不释放资源

线程的加入



》》》 线程的加入

语法

线程对象.join() 等待该线程终止, 原线程才继续运行

线程对象.join(**long millis**) 最久等多少毫秒

```
class A extends Thread{  
    public void run(){  
        Thread B = new Thread;  
        B.start();  
        B.join();  
    }  
}
```



线程的中断

线程对象.interrupt()

线程的优先级

setPriority(int newPriority)

Thread.MAX_PRIORITY = 10

Thread.MIN_PRIORITY = 1

Thread.NORM_PRIORITY = 5

设置优先级并不保证cpu对于线程的执行顺序

线程的同步

给线程加锁

synchronized关键字

同步方法

```
synchronized void method(){  
    ...  
}
```

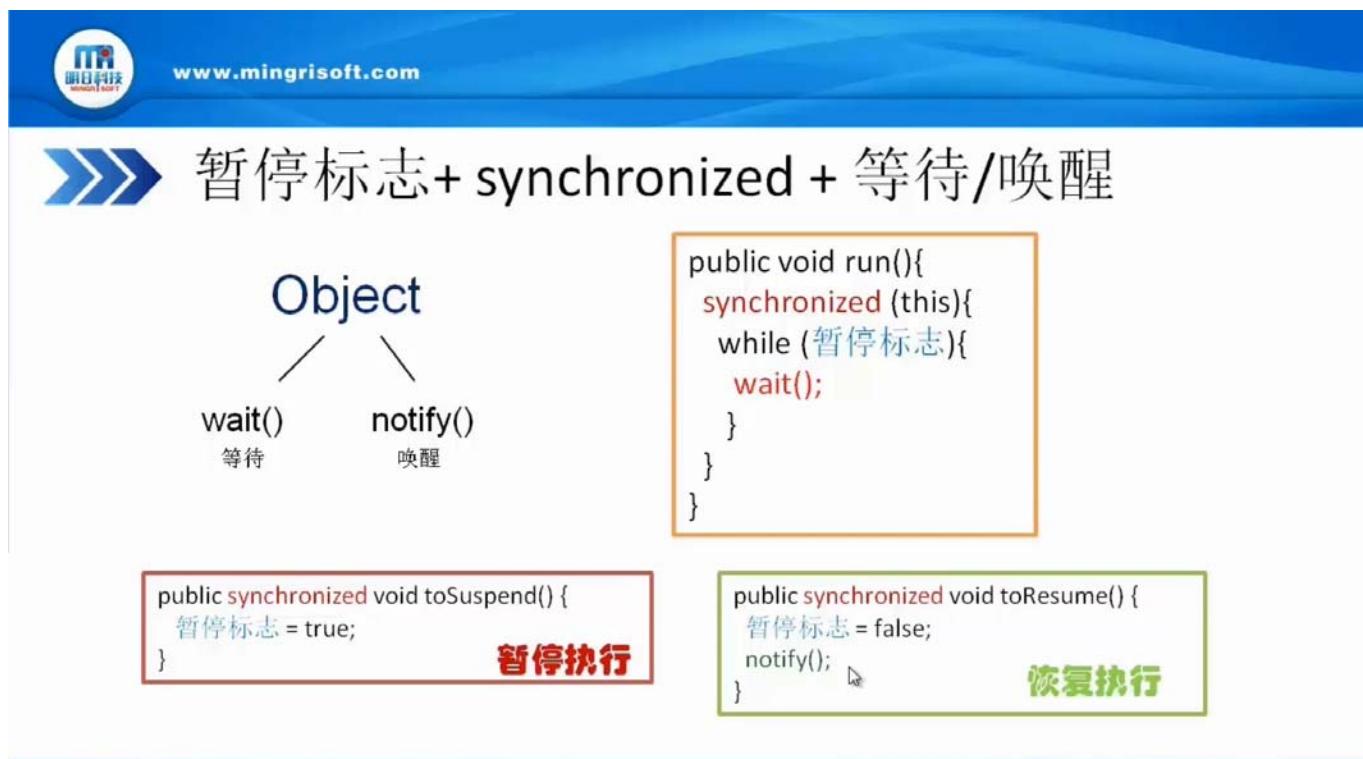
同步代码块

```
synchronized(Object){ //Object表示任意对象，仅作为加锁的标志
```

```
...  
}
```

多线程并发程序时，建议写为线程同步方法

线程的暂停与恢复



网络程序设计基础

端口(0~60035使用1024以上的)

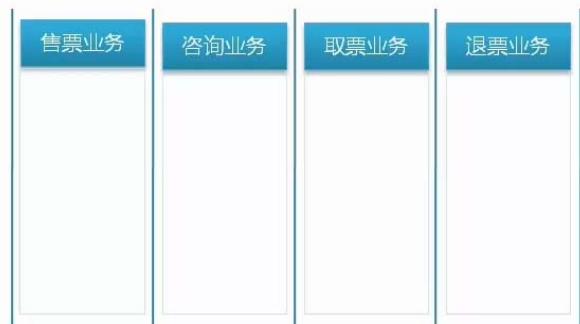


》 端口

计算机中的端口



售票大厅的窗口



套接字

Socket提供给程序可以对外进行连接的接口

IP地址封装



》 IP地址的格式

192.168.0.1
(0~255) IPv4

0.0.0.0 255.255.255.255

1080:0:0:0:8:800:200C:417A IPv6

特殊的IP：本地回送地址127.0.0.1

IP地址封装类

```
InetAddress host=InetAddress.getLocalHost(); //获取本地地址  
InetAddress host=InetAddress.getByName("192.168.0.1") //获取指定地址，IP地址  
InetAddress host=InetAddress.getByName("www.baidu.com") //域名地址  
InetAddress host=InetAddress.getByName("WIN-NQEBCJ88FO") //计算机名  
InetAddress[] host=InetAddress.getAllByName("www.baidu.com") //获取指定主机的所有地址
```

TCP程序设计

客户端套接字

```
Socket client=new Socket("192.168.1.1", 1100)
```

```
getInputStream() //获取套接字输入流，返回InputStream对象  
getOutputStream() //获取套接字输出流，返回OutputStream对象  
isConnected() //查看套接字是否连接  
close() //关闭套接字
```

服务器套接字

```
ServerSocket server=new ServerSocket(1100); //服务器开启的端口  
accept() //等待客户端的连接，返回成功连接的Socket对象  
getInetAddress() //返回服务器套接字的本地地址  
isClosed() //查看服务器套接字是否关闭  
close() //关闭套接字
```

TCP程序设计流程

创建服务器套接字

创建客户端套接字

服务器开启监听

处理传输数据

启动服务器

启动客户端

UDP程序设计

数据包

```
DatagramPacket(byte[] buf, int length, InetAddress address, int port)
```

数据包套接字

构造方法

```
DatagramSocket(int port, InetAddress addr) //绑定本地的端口和IP地址
```

```
//两个方法
```

```
send(DatagramPacket) //发送数据包
```

```
receive(DatagramPacket) //接收数据包
```

子类

```
MulticastSocket extends DatagramSocket
```

```
//两个方法
```

```
joinGroup(InetAddress) //加入广播组
```

```
leaveGroup(InetAddress) //离开广播组
```

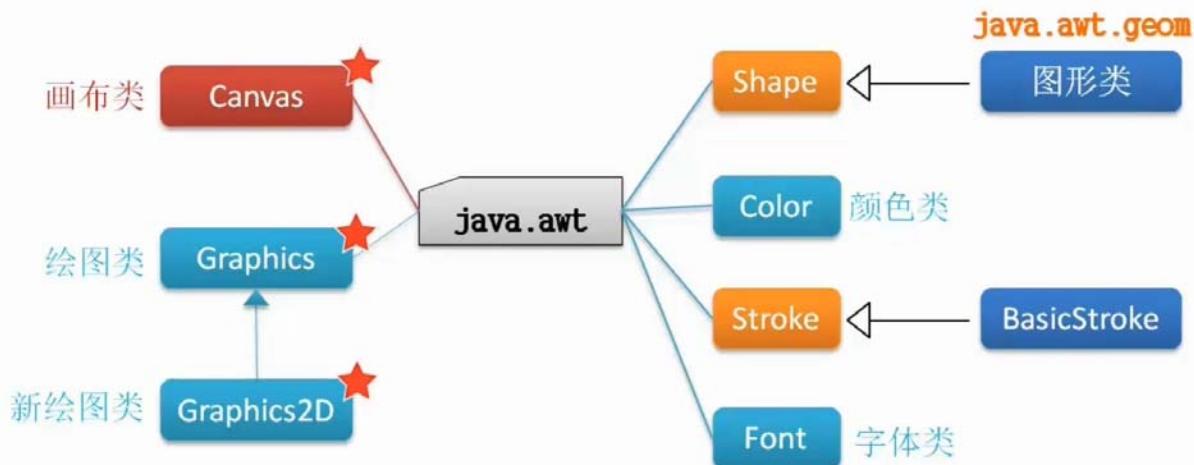
UDP多播套接字支持同时多个人接收

多线程聊天室

java绘图基础



》》》 本章知识点关系图



Graphics可以画一些基本的几何图形或者展示文字

Graphics2D可以把所有的图形对象化，控制画笔粗细，实现颜色渐变，可以实现坐标系的移动翻转

绘制几何图形

步骤：

创建窗体对象

创建画布

调用Graphics对象或创建Graphics2D绘图对象

调用绘图方法

绘图的第一要素

Canvas()画布类

paint()绘图方法

repaint()重绘

Graphics对象提供的绘图方法

`drawArc()`//画弧形

`drawLine()`//画直线

`drawRect()`//画矩形

```
drawOval()//画椭圆  
drawPolygon//画多边形  
//图形前面加上fill关键字则表示实心的
```

Graphics2D特有的绘图方法

```
draw(Shape form)//画某个图形  
fill(Shape form)//填充某个图形 (shape表示图形接口)
```

设置颜色

步骤

创建Color对象 (Color类)

```
Color col=new Color(int r,int g,int b)//rgb表示红绿蓝
```

设置颜色

```
g2.setColor(Color color)
```

设置画笔

步骤：

创建stroke接口对象

```
BasicStroke()  
BasicStroke(float width)//画笔宽度  
BasicStroke(float width,int cap,int join)  
//cap表示线端点的装饰(CAP_ROUND;CAP_BUTT;CAP_SQUARE), join路径线段交汇处的装饰 (JOIN_BEVEL;JOIN_MITER;JOIN_ROUND)
```

设置画笔

```
setStroke(Stroke stroke)//Stroke对象表示画笔
```

绘制文本

创建字体

创建Font对象

```
Font(String name,int style,int size)//name: 字体名称, style: 字体样式
```

设置对象

```
g2.setFont(Font font)//字体对象
```

绘制图像

```
drawImage(Image img,int x,int y,ImageObserver observer)//observer: 图像重新绘制时要通知的对象
```

图片放大和缩小

```
drawImage(Image img,int x,int y,int width,int height,ImageObserver observer)
```

图像翻转

语法

```
drawImage(Image img, 图片对象  
          int dx1, int dy1, int dx2, int dy2, 目标矩形坐标  
          int sx1, int sy1, int sx2, int sy2, 源矩形坐标  
          ImageObserver observer) 图像重新绘制时要通知的对象
```

sx1/sy1	源矩形左上角的x、y坐标
sx2/sy2	源矩形右下角的x、y坐标
dx1/dy1	目标矩形对应角的x、y坐标
dx2/dy2	目标矩形对应角的x、y坐标

图片旋转

```
rotate(double theta)//Graphics2D类提供 theta表示旋转角度
```

图像倾斜

```
shear(double shx,double shy)//水平方向的倾斜量和垂直方向的倾斜量
```

MVC

数据、表现和控制三者分离

M=Model(模型)：保存和维护数据，提供接口让外部修改数据，通知表现需要刷新

V=View(表现)：从模型获得数据，根据数据画出表现

C=Control(控制)：从用户得到输入，根据输入调整数据