目录

[Java关键字: 1](#_Toc419651960)

[位运算符： 1](#_Toc419651961)

[文档注释： 1](#_Toc419651962)

[foreach语句： 2](#_Toc419651963)

[数组： 2](#_Toc419651964)

[字符串 3](#_Toc419651965)

[类和对象 7](#_Toc419651966)

[接口、继承、多态 9](#_Toc419651967)

[类的高级特性 10](#_Toc419651968)

[Java集合类 12](#_Toc419651969)

[异常处理 15](#_Toc419651970)

[Java图形程序设计 16](#_Toc419651971)

[1、 框架定位 16](#_Toc419651972)

[1.1框架属性 16](#_Toc419651973)

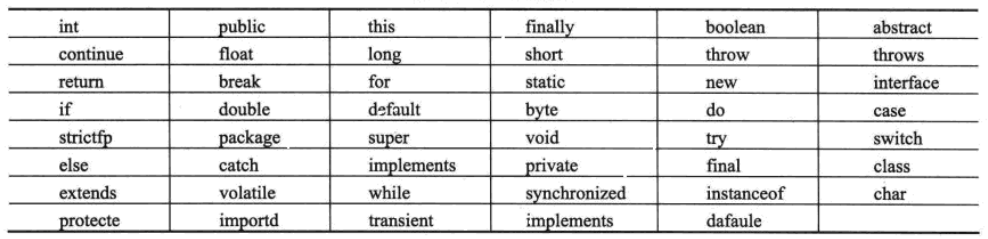
[1.2确定合适的框架大小 16](#_Toc419651974)

[2、在组件中显示信息 18](#_Toc419651975)

[3、处理2D图形 18](#_Toc419651976)

[4使用颜色 19](#_Toc419651977)

****Java关键字:****



boolean：布尔型，ture、false

final：声明常量，格式：final 数据类型 常量名[=值]；final double PI = 3.14F;

static：类的静态变量可以通过“类名.静态变量”的方式，在其他类中引用

局部变量可与成员变量同名，此时成员变量被隐藏。

public class Val

{

static int times = 3; //类的静态成员变量

public static void main(String[] args)

{

int times = 4; //局部变量，成员变量被隐藏

System.out.println(“成员变量：”+Val.times);//调用成员变量

}

}

****位运算符：****



按位异或：（^）相同为0，不同为1

左移：（<<）移空补0

右移：（>>）高位是0，移空补0；高位是1，移空补1

无符号右移：（>>>）移空补0

****文档注释：****

使用“/\*\* \*/”标记文档注释

****foreach语句：****

格式： for(元素变量x:遍历对象obj)

{

引用了x的语句；//x为数组元素；

}

foreach在遍历数组时很方便。

int arr[] = {1,2,3,4,5}

for(int x:arr)

{

System.out.println(x+”\t”);

}

数组：

1.数组声明方式：

数组元素类型 数组名[];

数组元素类型[] 数组名；

int arr[];

String []str;

2.分配内存空间：

数组名 = new 数组元素类型[元素个数];

arr = new int[5];

使用new关键字为数组分配内存时，数组中各个元素初始值都是0。

int mounth[] = new int[12]{1,2,3,4,5,6,7,8,9,10,11,12};//声明时初始化

3.二维数组

数组元素类型 数组名[][];

数组元素类型[][] 数组名;

int arr[][] = {{1,2},{3,4}};

4.填充替换数组元素

fill(int[] a, int value); 用value值填充数组a所有元素，返回填充后的数组。

fill(int[] a, int fromIndex, int toIdex, int value); 用value替换数组arr从下标fromIndex（包括）到下标toIndex（不包括）的元素，返回替换元素后的数组。如果指定的索引位置大于或等于数组的长度，则报错ArrayIndexOutOfBoundsException（数组越界）。

code:

import java.util.Arrays;

int arr[] = new int[5];

Arrays.fill(arr, 3); //填充整个数组

Arrays.fill(arr,1,3,0); //替换元素1~3（不包括3）的值为0

5.数组排序

Arrays.sort(object); 按从小到达的顺序排序，返回排序后的数组。

6.复制数组

copyOf(arr, int newlength); 复制数组至指定的长度，返回复制后的数组。

newlength：复制后新数组的长度，新数组长度大于arr长度，则用0填充，char型暑促用null填充；如果小于arr的长度，则从arr的第一个元素开始截取至满足长度为止。 code:

int arr = new int[]{1,3,2,5,6,4};

int newarr[] = Arrays.copyOf(arr,5);

copyOfRange(arr, int fromIndex, int toIndex); 从数组arr下标fromIndex(包括，值在0到数组长度之间)元素至toIndex(不包括，值可以大于arr长度)元素到新数组，返回复制后的新数组。

code:

int arr = new int[]{2,3,4,1,5,6};

int newarr[] = Arrays.copyOfRange(arr,2,5); //复制下标2~5的元素

字符串

1.创建字符串

使用String()方法初始化一个新创建的String对象

String s = new String();

用一个字符数组a创建String对象

char a[] = {‘g’,’o’,’o’,’d’};

String s= new String(a); 等价于 String s = new String(“good”);

String(char a[], int offset, int length);提取字符串a 中的一部分创建一个字符串对象,offset为开始截取字符串的位置，length为截取长度。

char a[] = {‘a’,’b’,’c’,’d’};

String s = new String(a,2,3); 等价于 String s = new String(“bcd”);

2.字符串连接

String s = s1 + s2;

“+” 运算符，只要“+”运算符的一个操作数是字符串类型，编译器就会将另外的操作数类型转换为字符串类型。

str.length(); 字符串长度

str.indexOf(substr); 返回str中第一次出现substr的索引。

str.lastIndexOf(substr); 返回str中最后出现substr的索引。

str.charAt(int index); 返回指定索引位置的字符。

3.字符串替换

去除字符串中的空格

str.trim(); 返回去除前导和尾部空格后的字符串。

StringTokenizer(String str, String delim); delim为从字符串str中去除掉的字符串，返回去掉后的字符串。

str.replace(String regex, String replacement);用于替换所有与指定字符串匹配的字符串。regex：要替换的字符串；replacement：用来替换的新字符串；返回替换后的字符串。

str.replaceAll(String regex, String replacement);regex：str中要被替换的字符串，replacement：用来替换regex的字符串，返回替换后的字符串。

区别：

replace的参数是char和CharSequence,即可以支持字符的替换,也支持字符串的替换  
 replaceAll的参数是regex,即基于规则表达式的替换,比如,可以通过replaceAll("\\d", "\*")把一个字符串所有的数字字符都换成星号;

str.replaceFirst(String regex, String replacement); 用replacement替换第一个出现的字符串regex。返回替换后的字符串。

4.判断字符串

str.equals(String otherstr); 与otherstr比较，返回boolean类型。

str.equalsIgnoreCase(String otherstr);与otherstr比较，不区分大小写。

“==”比较的是内存位置，卜世怡用来比较字符串。

str.startsWith(String prefix); 判断字符串是否以prefix字符串开始，返回boolean。

str.endsWith(String suffix); 判断字符串是否以suffix字符串结尾，返回boolean。

5.字母大小写转换

str.toLowerCase(); 全部转换为小写。

str.toUpperCase(); 全部转换为大写。

6.字符串分割

str.split(String sign); sign:分割字符串的分割符，可以为正则表达式，返回根据分割符分割后的字符串。

str.split(String sign, int limit);limit：限制分割的份数，返回按分割符分割指定份数后的字符串。

如果想定义多个分割符，使用符号“|”连接，例如：“,|=”、“=|>”。

code:

String s = new string(“abc,def,ghi,jkl”);

String[] news = s.split(“,”);//用逗号分割后的字符串

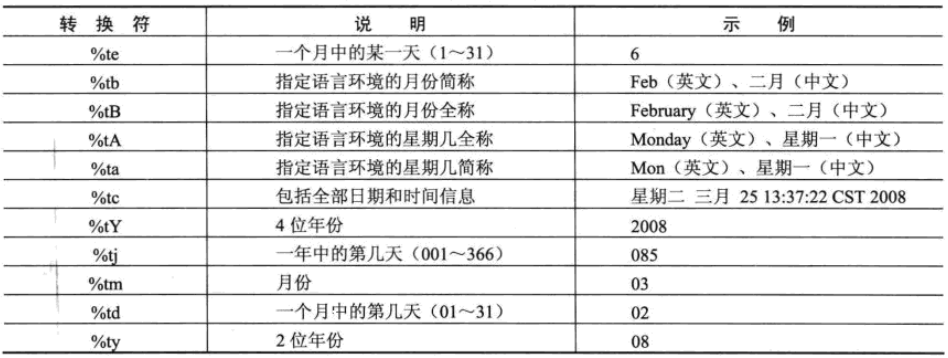
String[] news = s.split(“,”,2);//分割为2份。

7.格式化字符串

String.format(String format, Object…args); format：格式化转换类型；args：格式化字符串中说明符引用的参数，如果还有还有格式化说明符以外的参数，则忽略这些额外的参数，此参数数目可变，可以为0；返回一个格式化的字符串。

String.format(Local l, String format, Object…args); l：格式化过程中要应用的语言环境，如果为null则不进行本地化。

日期格式化：



时间格式化：



日期时间组合格式化：



常规类型格式化：



System.out.println("日期和时间格式化："+String.format("%tc", date));

8.正则表达式

boolean matches(String regex); regex：指定的正则表达式。

正则表达式的元字符：



boolean bs = s.matches("\\p{Upper}\\p{Lower}\\d");

正则表达式的限定符：



方括号中元字符的含义：



String s = “1[3,5,8]\\d{9}”; //第一位是1，第二位是3、5、8中的一个，后面有9位为任意0~9的数字。

9.字符串生成器

StringBuilder append(String str); 将str追加到字符串生成器

StringBuilder append(StringBuffer strbuf); 将字符串缓存StringBuffer的值追加到字符串生成器

String s = “xxxx”;

StringBuilder builder = new StringBuilder();

builder.append(s);

StringBuffer buffer = new StringBuffer(“xxxxx”);

builder.append(buffer);

StringBuilder insert(int offset, String str); 将字符串str添加到偏移offset位置

String s1 = “int”;

String s2 = “ser”;

StringBuilder builder = new StringBuilder(s1);

builder.insert(2,s2);

StringBuilder delete(int start, int end); //删除字符串中从start开始到end结束的字符串

StringBuilder builder = new StringBuilder(“abcdefghijk”);

builder.delete(2,5);

Sring toString(); 将字符串生成器转换为字符串

类和对象

封装、继承、多态

1.构造方法

没有返回值、方法名与类名相同。

this关键字可调用类中的构造方法。

code:

public class test

{

public test()

{

this(“this调用有参数构造方法”);

System.out.println(“无参数构造。”);

}

public test(String s)

{

System.out.println(s+“ 有参数构造.”);

}

}

2.成员方法

权限修饰符 返回值类型 方法名(参数列表)

{

//方法体

return 返回值;

}

3.静态变量、静态常量、静态方法

由static关键字修饰，被称为静态成员，属于类所有。

可以在本类或其他类使用“类名.静态成员”的方法调用。

静态方法中不可以使用this关键字；

静态方法中不可以直接调用非静态方法。

不可将方法体内的局部变量声明为static；

在执行类时，希望先执行类的初始化动作，可以使用static定义一个静态区域，执行时会首先执行static快中的程序，且只会执行一次。

code:

public class test

{

static

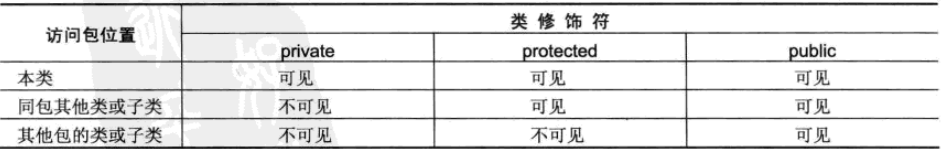
{

xxx.

}

}

权限修饰符：



4.this关键字

可以调用成员变量，成员方法，作为方法返回值。

code:

public Book getBook()

{

return this;

}

方法的返回值类型为Book，用this返回Book类的对象。

5.对象

一个类的引用：

Book book;

创建对象：

new Book(); 调用构造方法实例化对象

引用与对象的关联：

Book book = new Book();

对象的比较：

“==”运算符：比较两个对象引用的地址是否相等；

equals()方法：比较两个对象引用所指的内容是否相等。

接口、继承、多态

1.接口的定义

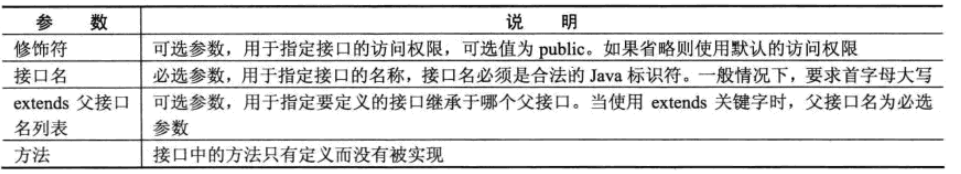
[修饰符] interface [final] 接口名 [extends 父接口名列表]

{

[public] [static] [final] 变量;

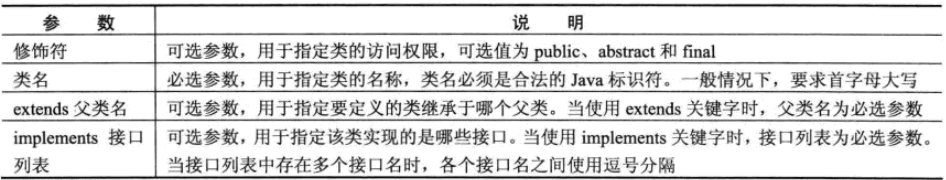
[public] [abstract] 方法;

}



2.接口的实现

[修饰符] class <类名>[extends 父类名] [implements 接口列表]{}



类只支持单继承，接口可多实现。

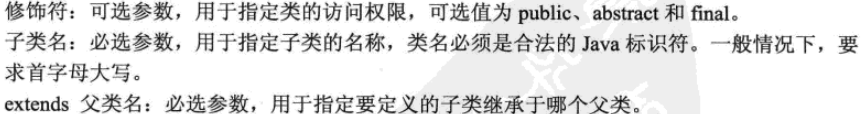
3.类的继承

[修饰符] class 子类名 extends 父类名

{

类体

}



4.重写（覆盖）

父子类之间，子类继承父类中所有可能被子类访问的成员方法时，如果子类的方法名与父类的方法名相同，子类不能继承父类的方法，称子类的方法重写了父类的方法。

5.super关键字

作用：

调用父类构造方法：在子类的构造方法中使用super关键字调用。

super([参数列表]);

如果父类的构造方法中包括参数，则参数列表为必选，用于指定父类构造方法的入口参数。

操作被隐藏的成员变量和被重写的成员方法

super.成员变量名

super.成员方法名([参数列表])

6.多态

使用方法的重载(overloading)和重写(overriding)实现多态

重载：

在一个类中出现多个方法名形同，参数个数或参数类型不同的方法。

类的高级特性

1.抽象类

抽象类：只声明方法而不实现。

抽象类不能被实例化，即不能创建对象；

定义抽象类时，在class前加关键字 abstract

abstract class test

{

}

抽象方法： 在抽象类中创建的，必须在子类中实现的方法，用abstract关键字修饰

abstract <方法返回值类型> 方法名(参数列表)

public abstract void test();

抽象方法不能使用private 或static关键字修饰。

包含一个或多个抽象方法的类必须声明为抽象类。

2.内部类

一个类中使用内部类，可以在内部类中直接存取其所在类的私有成员变量，但内部类的成员只有在内部类中可访问。

如果在外部类和非静态方法之外实例化内部类对象，需要使用“外部类.内部类”的形式指定该对象的类型。

code:

public static void main(String[] args)

{

OuterClass out = new OuterClass();

OuterClass.InnerClass in = out.new InnerClass();//实例化内部类对象

}

用this关键字获取内部类和外部类的引用

code:

public class test

{

private int x;

private class Inner

{

private int x = 0;

publi void dot(int x)

{

x++; //形参

this.x++; //内部类变量

test.this.x++; //外部类变量

}

}

}

局部内部类：在类的方法中定义的内部类

匿名内部类：用对象名来代替

return new A()

{

//内部类体

}

匿名内部类编译后，会产生以“外部类名$序号”为名称的class文件，序号从1~n排列。

code:

A a = new A()

{

//匿名内部类体

}

静态内部类：在内部类前添加static关键字

在静态内部类中可以声明static成员，但在非静态内部类中不可以声明static成员。

静态内部类不可以使用外部类的非静态成员。

创建静态内部类的对象，不需要其外部类的对象；

不能从静态内部类的对象中访问非静态外部类的对象。

内部类的继承：

code:

class Calss A

{

class ClassB

{

}

}

public class OutputInnerClass extends ClassA.ClassB

{

public OutputInnerClass(ClassA a)

{

a.super();

}

}

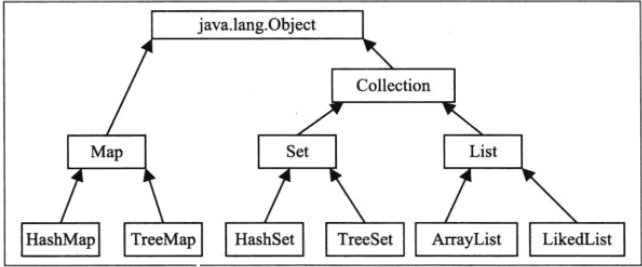
某个类继承内部类时，必须硬性的给这个类一个带参的构造方法，并且该构造方法的参数要继承内部类的外部类的引用，同时，在构造方法中使用a.super()语句，这样才为继承提供了必要的对象引用（这么麻烦，用个毛线啊。。。）

3.Class类与Java反射

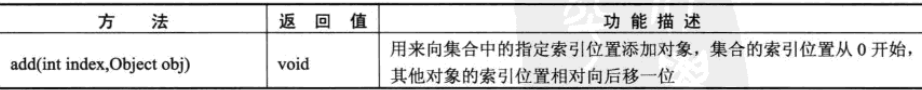
通过Java反射机制，可以在程序中访问已经装载到JVM中的Java对象的描述，实现访问、检测和修改Java对象本身信息的功能。在java.lang.reflect包中提供支持。

（内容太多，以后再学）

Java集合类



1.List接口的常用方法





List集合可以包含重复的对象

code：

String a = “a”,b = “b”;

List<String> list = new ArrayList<String>(); //创建List集合对象

list.add(a);

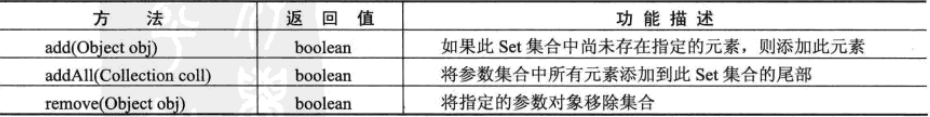
list.add(b);

list.add(a);

int firstIdx = list.indexOf(a); //获取第一次出现索引

int lastIdx = list.lastIndexOf(a); //获取最后一次出现索引

2.Set接口的常用方法



Set集合不允许出现重复值，可以使用addAll()方法，将Collection集合添加到Set集合中并去掉重复元素。

code:

List<String> list = new ArrayList<String>();

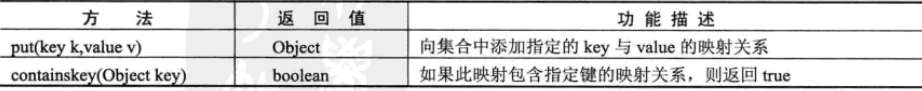
list.add(“a”);

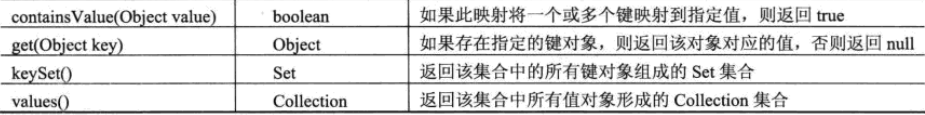
list.add(“b”);

Set<String> set = new HashSet<String>();

set.addAll(list);

3.Map接口的常用方法





Map集合中的元素是通过key、value 进行存储的，要获得集合中指定的key值或value值，需要先通过相应的方法，获取key集合或value集合，在遍历key集合或value集合获取指定值。

code:

Map<String,String> map = new HashMap<String,String>();

map.put(“1”,”a”);

map.put(“2”,”b”);

for(int i = 1; i <= 2; i++ )

System.out.println(“第”+i+”个元素是：”+map.get(“”+i+””));

4.集合类接口的实现

要实现List集合，通常需要声明为List类型，然后通过List接口的实现来对集合进行实例化。通常有ArrayList和LinkList类。

ArrayList类：实现可变数组，允许所遇元素，包括null，可以根据索引位置对集合进行快速的随机访问。缺点：向指定的索引位置插入或删除对象速度较慢。

List<String> list = new ArrayList<String>();

LinkList类：采用链表结构保存对象，优点在可以快速向集合中添加或删除对象，但对于随机访问集合中的对象较慢。

List<String> list = new LinkList<String>();

5.Set接口的实现类

要使用Set集合，通常声明为Set类型，然后通过Set接口的实现来对集合进行实例化。通常有HashSet和TreeSet类。

Set集合中的对象是无序的，遍历Set集合的结果和插入Set集合的结果顺序不同。

Set<String> collset = new HashSet<String>();

Set<String> collset = new TreeSet<String>();

6.Map接口的实现类

Map接口常用有HashMap和TreeMap，一般建议使用HashMap类实现Map集合。

HashMap类实现的Map集合对于添加和删除效率较高；TreeMap中的映射关系存在一定的顺序。

HashMap类基于哈希表的Map接口实现，允许使用null值和null键，但必须保证键的唯一性。

TreeMap不允许键对象为null。

可以通过HashMap类创建Map集合，当需要顺序输出时，再创建一个完全相同映射关系的TreeMap类实例。

6.迭代器

利用Iterator接口创建迭代器，位于java.util包下。

方法

hasNext():如果还有元素可迭代，返回ture

next()：返回迭代的下一个元素

remove()：从迭代器指向的collection中移除迭代器返回的最后一个元素。

ListIterator接口继承了Iterator接口。

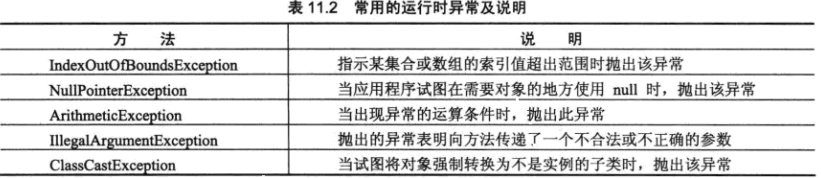
异常处理

1.异常的分内

可控式异常：可以预知的错误

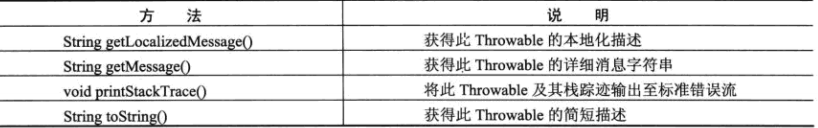


运行时异常：不能被编译器检测到的，在运行时会发生的异常



2.获取异常信息

java.lang.Throwable类是所有异常类的超类，提供获得异常的方法



3.处理异常

使用try…catch处理异常

try

{

//需要正常执行但可能发生异常的语句

}

catch(Exception ex)

{

//对异常进行处理的语句

}

使用try…catch…finally处理异常

Java图形程序设计

1、框架定位

处理框架大小和位置的方法：

setLocation和setBounds方法 用于设置框架的位置。

setIconImage 用于告诉窗口系统在标题栏、任务切换窗口等位置显示那个图标。

setTitle 用于改变标题栏的文字

setResizable 利用一个boolean值确定框架的大小是否允许用户改变

JFrame的继承层次

Object

Component

Container

JComponent

Window

Jpanel

Frame

JFrame

1.1框架属性

属性的获取/设置方法，一般以get/set加上属性名，属性名的第一个字母大写；针对类型为boolean的属性，获取方法有is开头。

public String getTitle()

public void setTitle()

public boolean isLocationByPlatform()

publiv void setLocationByPlatform()

1.2确定合适的框架大小

boolean isVisible()

void setVisible(boolean b)

获取或设置visible属性。

void setSize(int width, int height)

使用给定的宽度和高度，重新设置组件的大小。

void setLocation(int x, int y)

将组件一道一个新的位置上。如果这个组件不是顶层组建，x和y坐标（或p.x和p.y）是容器坐标；否则是屏幕坐标。

void setBounds(int x, int y, int width, int height)

移动并重新设计组件大小。

Dimension getSize()

void setSize(Dimension d)

获取或设置当前组建的属性。

void toFront()

将这个窗口显示在其他窗口前面。

void toBack()

将这个窗口移到左面窗口栈的后面，并重新排列所有的可见窗口。

boolean isLocationByPlatform()

void setLocationByPlatform()

获取或设置locationByPlatform属性，这个属性在窗口显示之前被设置，有平台选择一个合适的位置。

boolean isResizable()

void setResizable（boolean b）

获取或设置resizable属性，这个属性设置后，用户可以重新设置框架的大小。

String getTitle()

void setTitle(String s)

获取或设置title属性，这个属性确定框架标题栏中的文字。

Image getIconImage()

void setIconImage(Image image)

获取或设置iconImage属性，这个属性确定框架的图标。窗口系统可能会将图标作为框架装饰或其他部位的一部分显示。

boolean isUndecorated()

void setUndecorated(boolean b)

获取或设置undecorated属性。这是属性设置后，框架显示中将没有标题栏或关闭按钮这样的装饰。在框架显示前，必须调用这个方法。

int getExtendedState()

void setExtendedState(int state)

获取或设置窗口状态，状态如下：

frame.NORMAL

frame.ICONIFIED

frame.MAXMIZED\_HORIZ

frame.MAXMIZED\_VERT

frame.MAXMIZED\_BOTH

static Toolkit getDefaultToolkit()

返回默认的工具箱。

Dimension getScreenSize()

返回用户屏幕尺寸。

ImageIcon(String filename)

构造一个图标，其图标村粗在一个文件中。

Image getImage()

获取该图标的图像。

2、在组件中显示信息

Container getContentPane()

返回这个JFrame的内容窗格对象。

Component add(Component c)

将一个给定的组件添加到该框架的内容窗格中。

void repaint()

“尽可能快的”重新绘制组件

Dimension getPreferredSize()

要覆盖这个方法，返回这个组件的首选大小。

void paintComponent(Graphics g)

覆盖这个方法来描述应该如何绘制自己的组件。

void pack()

调整窗口大小，要考虑到其组建的首选大小。

3、处理2D图形

double getCenterX()

double getCenterY()

double getMinX()

double getMinY()

double getMaxX()

double getMaxY()

返回闭合矩形的中心，以及最小、最大x和y坐标值

double getWidth()

double getHeight()

返回闭合矩形的宽和高

double getX()

double getY()

返回闭合矩形左上角的x和y坐标

Rectangle2D. Double (double x, double y, double w, double h)

Rectangle2D.Float(double x, double y, double w, double h)

利用给定的左上角、宽和高，构造一个矩形

Ellipse2D. Double (double x, double y, double w, double h)

利用给定的左上角、宽和高的外接矩形，构造一个椭圆

Point2D.Double(double x, double y)

利用给定的坐标构造一个点

Line2D.Double(Point2D start, Point2D end)

Line2D.Double(double startX, double startY, double endX, double endY)

使用给定的起点和终点构造一条直线。

绘制图形，首先需要创建一个实现了Shape接口的类的对象，然后调用Graphics2D类中的draw方法，例如：

画一个矩形：

Graphics2D g2 = (Graphics2D) g;

Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);

g2.draw(rect);

4、使用颜色

将调用draw绘图的方法换成fill方法，可以用一种颜色填充一个封闭图形。fill方法会在右侧和下方少绘制一个像素点，例如绘制一个 new Rectangle2D.Double(0, 0, 10, 20)，填充的矩形不会绘制x=10,y=20的像素。

Graphics2D g2 = (Graphics2D) g;

Rectangle2D rect = new Rectangle2D.Double(leftX, topY, width, height);

g2.setPaint(Color.RED);

g2.fill(rect);

Color类定义颜色，定义了13个常量表示13中标准的颜色：

BLACK, BLUE, CYAN , DARD\_GRAY, GRAY, GREEN, LIGHT\_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW

可以通过提供红、绿、蓝三种颜色成分来创建一个Color对象，以达到定制颜色的目的。三种颜色的值都是0~255之间的整形数：

Color(int redness, int greenness, int blueness)

例如： g2.setPaint(new Color(0,128,128));

g2.drawString(“welcome!”,75,75);

设置背景颜色，需要使用Component类中的setBackground方法。

MyComponent p = new MyComponent();

p.setBackground(Color.PINK);

Color类中的brighter()和darker()方法，分别加亮或变暗当前的颜色，要达到耀眼的效果，需要调用三次brighter()方法。

SystemColor类中预定义了很多颜色的名字。

p.setBackground(SystemColor.window) 将把面板的背景颜色设定为用户桌面上所有窗口使用的默认颜色。

Color getColor()

void setColor(Color c)

获取或改变当前的颜色。

Paint getPaint()

void setPaint(Paint p)

获取或设置这个图形环境的绘制属性。Color类实现了Paint接口，因此可以使用这个方法将绘制属性设置为纯色。

void fill(Shape s)

用当前的颜料填充该图形。

Color getBackground()

void setBackground(Color c)

获取或设置背景颜色。

Color getForeground()

void setForeground(Color c)

获取或设置前景颜色。

5、显示图形

boolean drawImage(Image img, int x, ing y, ImageObserver boserver)

绘制一副非比例图像。注意：这个调用可能会在图像还没有绘制完毕就返回。

参数：img 将要绘制的图像

x 左上角的x坐标

y 左上角的y坐标

observer 绘制进程中以通告为目的的对象，可以为null

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

绘制一副比例图像，系统按照比例将图像放入给定宽和高的区域。注意：这个调用可能会在图像还没有绘制完毕就返回。

参数： width 描述图像的宽度

height 描述图像的高度

void copyArea(int x, int y, int width, int height, int dx, int dy)

拷贝屏幕的一块区域。

参数： x 原始区域左上角x坐标

y 原始区域左上角y坐标

width 原始区域的宽度

height 原始区域的高度

dx 原始区域到目标区域的水平距离

dy 原始区域到目标区域的垂直距离

事件处理

1、处理按钮点击事件

整体过程：

a）设置窗体尺寸：setSize(Width,Height);

b）创建Button对象：JButton MyButton = new JButton(“MyButton”);

c）创建Panel对象：private JPanel buttonPanel = new JPanel();

d）将按钮添加至面板：ButtonPanel.add(MyButton);

e）将面板添加至窗体：add(buttonPanel);

f）创建监听器类（内部类，其方法拥有访问外部面板的权限）：

class ButtonAction implements ActionListener

{

privater Color backgroundColor;

public ButtonAction(Color c) //构造方法和其他

{

backgroundColor = c;

}

public void actionPerformed(ActionEvent event)

{

buttonPanel.setBackground(gackgroundColor);

}

}

g）构造对象，并将对象设置为按钮监听器：

ButtonAction buttonAction = new ButtonAction(Color.RED);

buttonAction.addActionListener(buttonAction);

JButton(String label)

JButton(Icon icon)

JButton(String label, Icon icon)

构造一个按钮，标签可以是常规的文本，也可以是HEML。

Component ad(Component c)

将组件c添加到这个容器。