# Java 核心技术卷一

Java是一个完整的平台，有一个庞大的库，其中包含了很多可重用的代码，以及一个提供诸如安全性，跨操作系统的可移植性以及自动垃圾收集等服务的执行环境；

简单性，面向对象，分布式，健壮性，安全性，体系结构中立，可移植性，解释性，高性能，多线程，动态性，

类名是以大写字母开头的名词，如果名词由多个单词组成，每个单词的第一个字母应该大写，这个就是驼峰命名法；

Java语言规范：<http://docs.oracle.com/javase/specs>

Java的Bug查询：<http://bugs.java.com/budatabase/index.jsp>

Java虚拟机规范：<http://docs.oracle.com/javase/specs/jvm/se8/html>

Java的函数调用：object.method(parameters)

Java8大数据类型：

byte , short , int , long , char , float , double , boolean

不提倡使用int i , j 这种风格声明变量。逐一声明每一个变量可以提高程序的可读性；

当声明一个变量以后，必须用赋值语句对变量进行显示的初始化，千万不要使用未初始化的变量的值，变量的声明尽可能靠近变量第一次被使用的地方。

Java习惯上用final修饰的常量要全部大写，一旦确定就不能改变，通常定义在方法的外部，可以使一个类中的所有方法都可以使用；

在Math类中，包含了各种各样的数学函数：

Math.sqrt(x) 可以计算一个数值的平方根

Math.pow(x,a) 可以计算x的a次幂

Math.round(x) 强制类型转换

Math.sin

Math.cos

Math.tan

Math.atan

Math.atan2 三角函数

Math.exp 指数函数

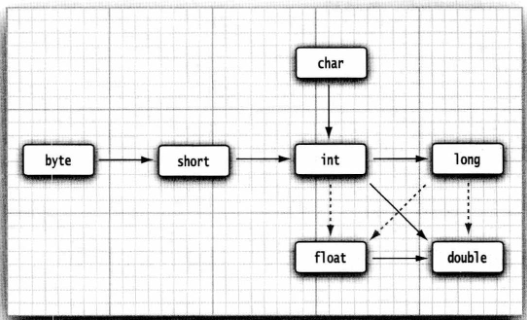
Math.log 自然对数

Math.log10 以10为底的对数

Math.PI 3.1415926……的近似值

Math.E e常量的近似值

数值类型之间的转换：

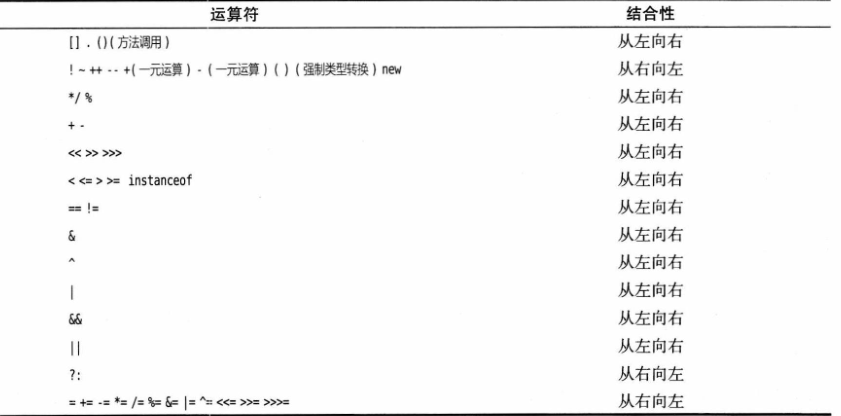


(实线表示无信息丢失，虚线表示可能有精度损失)

三元运算：condition ? expression1 : expression2

如果condition为true，返回expression1，否则返回expression2

运算符优先级：



Java中的字符串：

str.substring(a,b) 提取字符串，包前不包后；

str+”ab” 字符串str拼接ab

String.join(“/”,”a”,”b”,”c”) 把多个字符串放在一起，用界定符分隔

str.repeat(x) 把字符串str重复3次

str1.equals(str2) 检测str1和str2是否相等

str.length() == 0 str.equals(“”) 判断字符串是否为Null

String API

char charAt(int index) 返回给定位置的代码单元。

int codePointAt(int index) 返回给定位置开始的码点

int offsetByCodePoints(int startIndex , int cpCount) 返回从startIndex码点开始，cpCount个码点后的码点索引

int compareTo(String other) 按照字典顺序，如果字符串位于other之前，返回一个负数；如果字符串位于other之后，返回一个正数；如果两个字符串相等，返回一个0；

IntStream codePoints () 将字符串的码点作为一个流返回，可以调用toArray将它们放在一个数组里；

new String(int[] codePoints , int offset , int count) 用数组中从offset开始的count个码点构造一个字符串；

boolean empty()

boolean blank() 如果字符串为空或者由空格组成，返回true

boolean equals(Object other) 如果字符串与other相等，返回true

boolean equalsIgnoreCase (String other) 如果字符串与other（忽略大小写）相等，返回true；

boolean startsWith(String prefix)

boolean endsWith(String suffix)如果字符串是以prefix开头或以suffix结尾，返回true；

int indexOf(String str)

int indexOf(String str , int fromIndex)

int indexOf(int cp)

int indexOf(int cp , int fromIndex) 返回字符串str 或码点cp匹配的第一个子串的开始位置。从索引0或fromIndex开始匹配，如果原始字符串中不存在str，返回-1；

int lastIndexOf(String str)

int lastIndexOf(String str , int fromIndex)

int lastIndexOf(int cp)

int lastIndexOf(int cp , int formIndex) 返回字符串str或码点cp匹配的最后一个子串的开始位置。从原始字符串的末尾或formIndex开始匹配；

int length() 返回字符串的长度

int codePointCount(int startIndex , int endIndex) 返回startIndex和endIndex – 1之间的码点数；

String replace(CharSequence oldString , CharSequence newString) 返回一个新字符串，这个字符串用newString代替原始字符串中所有的oldString；

String substring(int beginIndex)

String substring(int beginIndex , int endIndex) 返回一个新的字符串，这个字符串包含原始字符串从beginIndex到字符串末尾或endIndex-1的所有代码单元；

String toLowerCase()

String toUpperCase()

返回一个新字符串，这个字符串将原始字符串中的字母改为大写字母或小写字母

String trim()

String strip()

返回一个新的字符串，这个字符串将删除原始字符串头部或尾部小于等于U+0020或空格

Java的API文档：<http://docs.oracle.com/javase/9/docs/api>

StringBuilder

Stringbuilder builder = new StringBuilder() ;

builder . append(str) ; 实现字符串的拼接

builder . toString();把多个字符串的拼接转换成一个字符串；

StringBuilder API

StringBuilder () 构造一个空的字符串构建器

int length() 返回构建器或缓冲器中的代码单元数量

StringBuilder append(String str) 追加一个字符串并返回this

StringBuilder appendCodePint (int cp) 追加一个码点，并将其转换为一个或两个代码单元并返回this；

StringBuilder append(char ch) 追加一个代码单元并返回this

void setCharAt(int i , char c) 将第i个代码单元设置为c

StringBuilder insert(int offset , String str) 在offset位置插入一个字符串str并返回this；

StringBuilder insert(int offset , char ch) 在offset位置插入字符ch并返回this

StringBuilder delete(int startIndex , int endIndex) 删除偏移量从startIndex到endIndex – 1的代码单元并返回this；

String toString()返回一个与构建器会缓冲器内容相同的字符串

Java的输入流API

Scanner in = new Scanner(System . in);

Scanner(InputStream in) 用给定的输入流创建一个Scanner对象

String nextLine() 读取输入的下一行内容

String next() 读取输入的下一个单词（以空格作为分隔符）

int nextInt()

double nextDouble() 读取并转换下一个表示整数或浮点数的字符序列

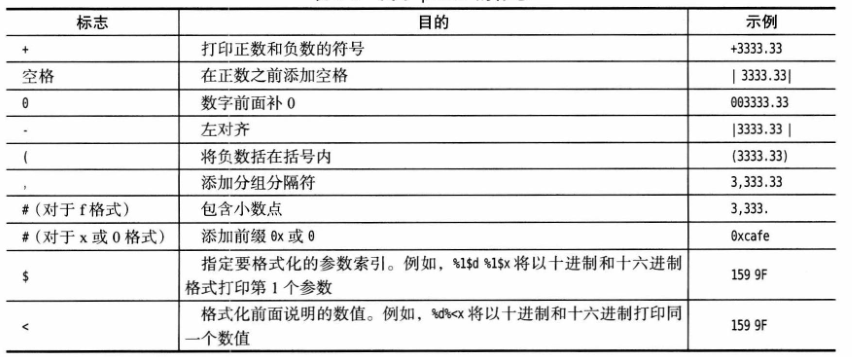
boolean hasNext() 检测输入中是否还有其它单词

boolean hasNextInt()

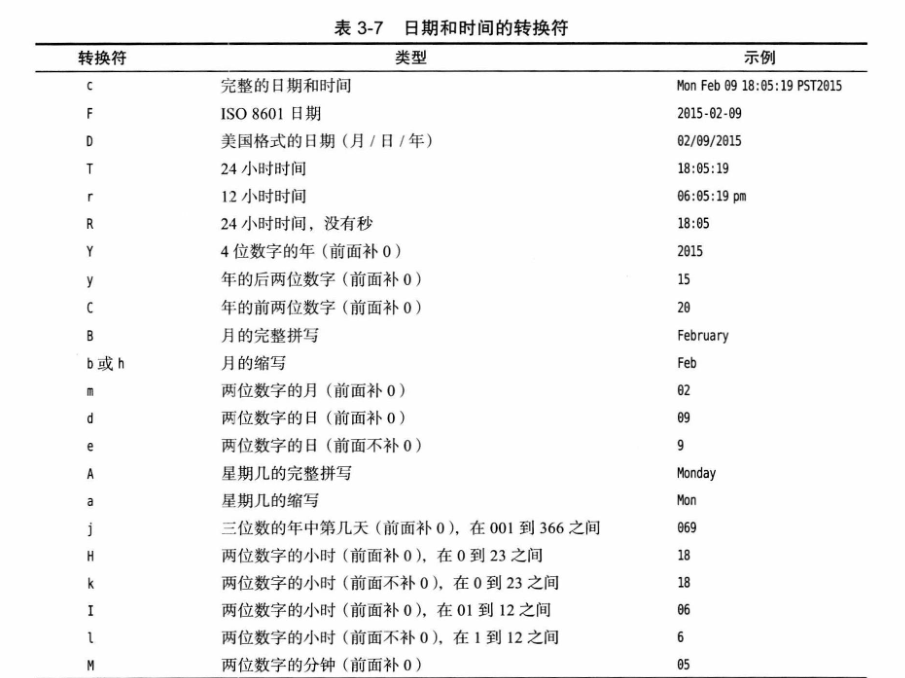
boolean hasNextDouble()检测是否还有下一个表示整数或浮点数的字符序列

Java输出流的占位转换符





可以使用静态的String.format()方法创建一个格式化的字符串，不打印输出；





文件输入与输出

Scanner in = new Scanner(Path.of(“文件地址”),StandardCharsets.UTF\_8);

读取一个文件

PrintWriter out = new PrintWriter(“文件地址”,StandardCharsets.UTF\_8);

写入文件，如果文件不存在，就创建一个文件

数组

int[] copiNumbers = Arrays.copyOf(Numbers2 ,Numbers1.length );

将一个数组的值拷贝到另一个数组中；

Arrays.sort(a) 对数值型数组进行排序（快速排序算法）

数组的API

static String toString(arr[] a) 返回包含a中元素的一个字符串

static arr[] copyOf(arr[] a,int end)

static arr[] copyOfRange(arr[] a,int start , int end) 返回与a类型相同的一个数组，其长度为length或end-start，数组元素为a的值；

static void sort(arr[] a) 使用优化的快速排序算法对数组进行排序

static int binarySearch(arr[] a,v)

static int binarySearch(arr[] a,int start ,int end ,v)使用二分查找算法在有序数组a中查找值v,如果找到v，则返回相应的下表,否则，返回负值r ,-r-1是v应该插入的位置；

static void fill(arr[] a,v) 将数组的所有元素设置为v

static boolean equals(arr[] a , arr[] b)如果两个数组大小相同，并且下标相同的元素都对应相等，返回true；

LocalDate API

static LocalDate now() 构造一个表示当前日期的对象

static LocalDate of(int year , int month , int day) 构造一个表示给定日期的对象

int getYear()

int getMonthValue()

int getDayOfMonth() 得到当前日期的年，月，日

DayOfWeek getDayOfWeek 得到当前日期是星期几

LocalDate plusDays(int n)

LocalDate minusDays(int n) 生成当前日期之后或之前n天的日期

当一个类没有任何其它构造器时，才会得到一个默认的无参构造器；

为了保证包名的绝对唯一性，要使用一个因特网域名以逆序的形式作为包名，然后对不同的工程使用不同的子包

类的设计技巧：

保证数据私有，不要破坏封装性。当数据保持私有时，表示形式的变化不会对类的使用者产生影响，更容易检测出Bug；

对数据进行初始化，不要依赖系统的默认值，应该显示的初始化所有的数据，可以提供默认值，也可以在构造器中设置默认值；

不要再类中使用过多的基本类型，这样会使类更容易修改和理解；

不是所有的字段都需要单独的字段访问器和字段更改器；

分解有过多职责的类；

类名和方法名要能够体现它们的职责；

优先使用不可变的类；

只能再继承层次内进行强制类型转换；

再将超类强制转换成子类之前，应该使用instanceof进行检查；

最好将类中的字段标记为private , 将方法标记为public

仅对本类可见 – private

对外部完全可见 – public

对本包和所有子类可见 – protected

对本包可见 – 默认

基本数据类型判断是否相等用==，对象判断相等用equals , instanceof(不建议)

编写equals方法的建议

1，显式参数命名为otherObject,稍后需要将它强制转换成另一个名为other的变量；

2，检测this和otherObject是否相等：if(this == otherObject ) return true;

3，检测othetObject是否为null，如果为null，返回false：if(otherObject == null) return false ;

4，比较this与otherObject的类，如果equals的语义可以在子类中改变，就使用getClass检测：if(getClass( ) != otherObject.getClass( )) return false ;

5，将otherObject强制转换成为对应类类型的变量：

ClassName other = (ClassName) otherObject

6，根据相等性概念的要求比较字段；

7，对于数组类型的字段，可以使用静态的Arrays.equals方法检测相应的数组元素是否相等；

散列码：hashCode()

数组列表ArrayList API

ArrayList<E> () 构造一个空数组列表；

ArrayList<E> (int initialCapacity) 用指定容量构造一个空数组列表；

boolean add(E obj) 在数组列表的末尾追加一个元素，永远返回true；

int size() 返回当前存储在数组列表中的元素个数

void ensureCapacity(int catacity) 确保数组列表在不重新分配内部存储数组的情况下有足够的容量存储给定数量的元素

void trimToSize() 将数组列表的存储容量消减到当前大小；

E set(int index , E obj) 将值obj放置在数组列表的指定索引位置，返回之前的内容；

E get(int index) 得到指定索引位置存储的值

void add(int index , E obj) 后移元素从而将obj插入到指定索引的位置;

E remove(int index) 删除指定索引位置的元素，并将后面所有的元素前移，返回删除的元素；

自动拆箱和装箱 Integer API

int intValue() 将这个Integer对象的值作为一个int返回

static String toString(int i) 返回一个新的String对象，表示指定数值i的十进制表示；

static String toString(int i , int radix) 返回数值i基于radix参数指定进制的表示；

static int parseInt(String s)

static int parseInt(String s,int radix) 返回字符串s表示的整数，指定字符串必须表示一个十进制整数，或者采用radix参数指定的进制；

static Integer valueOf(String s)

static Integer valueOf(String s , int radix) 返回一个新的Integer 对象，用字符串s表示的整数初始化。指定字符串必须表示一个十进制整数，或者采用radix参数指定的进制；

Number parse(String s) 返回数字值，假设给定的String表示一个数值；

Java中方法的参数使用省略号…表示这个方法可以接收任意数量的对象；

枚举类Enum API

static Enum valueOf(Class enumClass , String name) 返回给定类中有指定名字的枚举常量；

String toString( ) 返回枚举常量名

int ordinal( ) 返回枚举常量在enum声明中的位置，位置从0开始；

int compareTo(E other) 如果枚举常量出现在other之前，返回一个负整数，如果this==other，返回0，否则返回一个正整数，枚举常量的出现次序在enum声明中给出；

反射库是提供了一个丰富且精巧的工具集；

继承设计的技巧：

1，将公共操作和字段放在超类中；

2，不要使用受保护的字段；

3，使用继承实现”is-a”的关系；

4，除非所有继承的方法都有意义，否则不要使用继承；

5，在覆盖方法时，不要改变预期的行为；

6，使用多态，而不要使用类型信息；

7，不要滥用反射；

接口

接口中的所有方法都自动是public方法，在接口中声明方法时，不必提供关键字public；

比较API

int compareTo(T other) 用这个对象与other进行比较，如果这个对象小于other则返回一个负整数，如果相等返回0，否则返回一个正数；

static void sort(Object[] a) 对数组a中的元素进行排序，要求数组中的元素必须属于实现Comparable接口的类，并且元素之间必须是可比较的；

static int compare(int x , int y) 如果x < y返回一个负整数，如果x和y相等，返回0，否则返回一个正整数；

如果希望经过一定时间间隔就得到通知，可以使用java.swing.Timer类；

swing API

static void showMessageDialog(Component parent , Object Message) 显示一条包含提示消息个OK按钮的对话框，这个对话框位于parent组件的中央，如果parent为null，对话框将显示在屏幕的中央；

Timer(int interval , ActionListener listener) 构造一个定时器，每经过interval毫秒就通知listener一次；

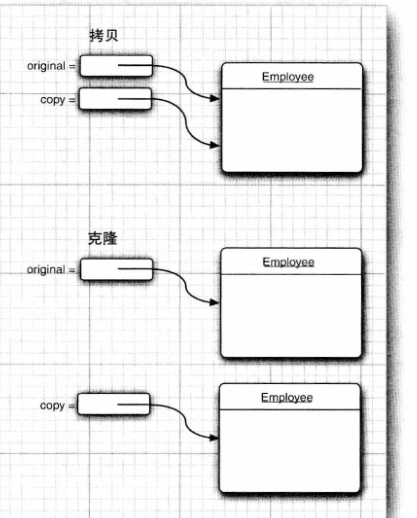
void start() 启动定时器，一旦启动，定时器将调用监听器的actionPerformed；

void stop() 停止定时器，一旦停止，定时器将不再调用监听器的actionPerformed

static Toolkil getDefaultToolkit() 获得默认的工具箱，工具箱包含有关的GUI环境信息；

void deep()发出一声铃响

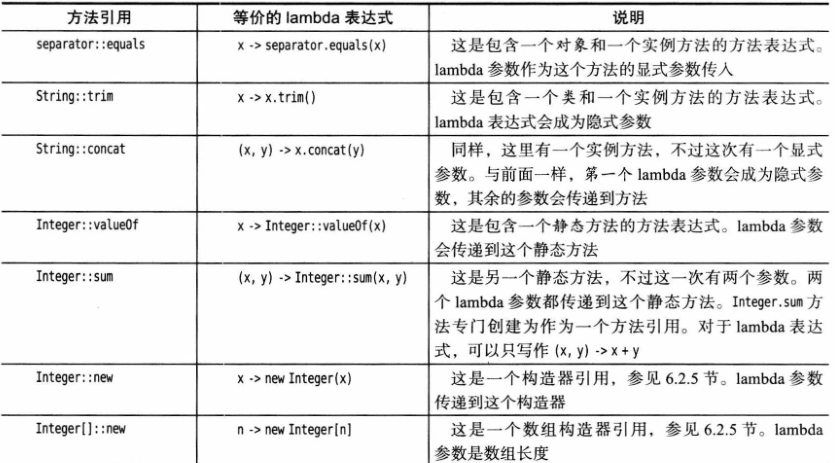
拷贝和克隆



如果要实现对象的克隆，要实现Cloneable接口，重新定义clone方法，并指定public访问修饰符；

lambda表达式

方法引用：



使用lambda表达式的重点是延迟执行。

1，在一个单独的线程中运行代码；

2，多次运行代码；

3，在算法的适当位置运行代码；

4，发生某种情况时执行代码；

5，只在必要的适合运行代码；

常用函数式接口：



调用action.run()时会执行这个lambda的表达式主体；

内部类

内部类可以对同一个包中的其它类隐藏；

内部类方法可以访问定义这个类的作用域中的数据，包括原本私有的数据；

编写一个程序遇到错误的时候最基本做到：

向用户通知错误；

保存所有的工作；

允许用户妥善的退出程序；

如果程序由于出现错误而使得某些操作没有完成，程序应该返回一种安全的状态，并能让用户执行其他的命令，或者允许用户保存所有的工作结果，并以妥善的方式终止程序；

程序遇到的错误：

1，用户输入错误；

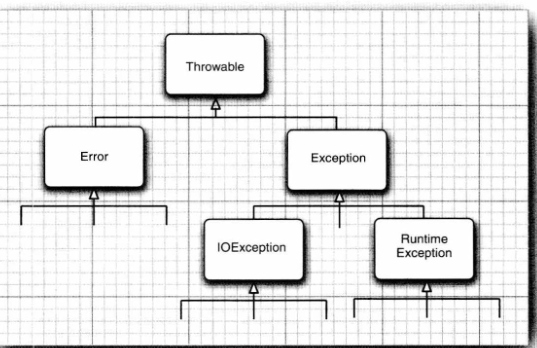
2，设备错误；

3，物理限制；

4，代码错误；

异常的分类

异常对象都是派生与Throwable类的一个类实例。



Error类层次结构描述了Java运行时系统的内部错误和资源耗尽错误（不应该出现）；

派生于RuntimeException的异常：

错误的强制类型转换；

数组访问越界；

访问null指针；

派生于IOException的异常

试图超越文件末尾继续读取数据；

试图打开一个不存在的文件；

试图根据给定的字符串查找Class对象，而这个字符串表示的类并不存在；

如果超类方法没有抛出任何检查型异常，子类也不能抛出任何检查型异常；

抛出异常的方法：

1，找到一个合适的异常类；

2，创建这个类的一个对象；

3，将对象抛出；

当Java提供的异常类不够用时，可以自己定义异常类，这个类要继承IOException：

Throwable( ) 构造一个新的Throwable对象，但没有详细的描述信息；

Throwable(String message) 构造一个新的Throwable对象，带有指定的详细描述信息，通常所有派生类的异常类都支持一个默认构造器和一个带有详细描述信息的构造器；

String getMessage( ) 获得Throwable对象的详细描述信息；

当程序代码捕获到异常后，可以使用该语句获取原始异常：

Throwable original = caughtException.getCause( );

这个包装放在catch(){…}里面，可以在子系统中抛出高层异常，而不会丢失原始异常的细节；

ThrowableAPI

Throwable(Throwable case)

Throwable (String message, Throwable case) 用给定的case构造一个Throwable对象；

Throwable initCase(Throwable case) 为这个对象设置原因，如果这个对象已经有原因，则抛出一个异常，返回this；

Throwable getCase( ) 获得设置为这个对象的原因的异常对象，如果没有设置原因，则返回null；

StackTraceElement[] getStackTrace( ) 获取构造这个对象时调用堆栈的轨迹；

void addSuppressed(Throwable t) 为这个异常增加一个“抑制的”异常，这出现在try-with-resources语句中，其中t是close方法抛出的一个异常；

Throwable getSuppressed( ) 得到这个异常的所有“抑制的”异常，一般来说，这些是try-with-resources语句中close方法抛出的异常；

Exception(Throwable cause)

Exception(String message , Throwable cause) 用给定的cause构造一个Exception对象；

RuntimeException(Throwable cause)

RuntimeException(String message , Throwable cause) 用给定的cause构造一个RuntimeException对象；

static StackWalker getInstance()

static StackWalker getInstance(StackWalker.Option.option)

static StackWalker getInstance(Set<StackWalker.Option> options) 得到一个StackWalker实例，包括StackWalker.Option枚举中的RETAIN\_CLASS\_REFERENCE , SHOW\_HIDDEN\_FRAMES和SHOW\_REFLECT\_FRAMES；

forEach(Consumer<? super StackWalker.StackFrame> action) 在每个栈帧上完成给定的动作，从最近调用的方法开始；

walk(Function<? super Stream<StackWalker.StackFrame>,? extands T> function) 对一个栈帧流应用给定的函数，返回这个函数的结果；

String getFileName( ) 得到包含该元素执行点的源文件的文件名，如果这个信息不可用则返回null；

int getLineNumber( ) 得到包含该元素执行点的源文件的行号，如果这个信息不可用则返回-1；

String getClassName( ) 得到方法包含该元素执行点的类的完全限定名；

String getDeclaringClass( ) 得到该方法包含该元素执行点的类的Class对象，如果这个栈遍历器不是用REAIN\_CLASS\_REFERENCE选项构造的，则会抛出一个异常；

String getMethodName( ) 得到包含该元素执行点的方法的方法名，构造器的方法名为<init>，静态初始化器的方法名为<clinit>，无法区分同名的重载方法；

boolean isNativeMethod( ) 如果这个元素的执行点在一个原生方法中，则返回true；

String toString( ) 返回一个格式化字符串，包含类和方法名，文件名以及行号；

使用异常的技巧：

1，异常处理不能代替简单的测试，只在异常情况下使用异常；

2，不要过分的细化异常，将正常处理与错误处理分开；

3，充分利用异常层次结构，不要只抛出RuntimeException异常，应该寻找一个合适的子类或创建自己的异常类，不要只考虑Thrable异常，否则，会使代码更加难读，更加难以维护；

4，不要压制异常；

5，在检测错误时，苛刻要比放任更好；

6，不要羞于传递异常；

7，对于程序中出现的错误和异常要早抛出，晚捕获；

使用断言（程序的自我保护）：

断言机制允许在测试期间向代码中插入一些检查，而在生产代码中会自动删除这些检查；

日志：

要生成简单的日志记录，可以使用全局日志记录器（global logger）并调用info方法：

Logger,getGlobal( ).info(“File->Open menu item selected”);

在适当的地方（如main的最前面）调用

Logger.getGlobal( ) .setLevel(Level.OFF);将会取消所有的日志；

可以调用getLogger方法创建或获取日志记录器：

private static final Logger myLogger = Logger.getLogger(“com.mycompany.myapp”)

7个日志级别：

SEVERE , WARNING , INFO , CONFIG , FINE , FINER , FINEST;

泛型程序设计

一个ArrayList类就可以收集任何类的对象；

定义简单的泛型类：

泛型类（generic class）就是有一个或多个类型变量的类，设计Java泛型时，主要目标是允许反省代码和遗留代码之间能够互相操作；

类型变量使用大写字母，很简短，Java库使用变量E表示集合的元素类型，K和V分别表示表的键和值的类型，T表示任意类型；

可以用具体的类型替换类型变量来实例化（instantiate）泛型类型；

泛型类相当于普通类的工厂；

对于Java泛型转换：

1，虚拟机中没有泛型，只用普通的类和方法；

2，所有的类型参数都会替换为它们的限定类型；

3，会合成桥方法来保存多态；

4，为保存类型安全性，必要时会插入强制类型转换；

不能用基本类型代替类型参数，其原因在于类型擦除；

集合

队列（queue）：先进先出，队列通常有两种实现方式，一种是使用循环数组，一种是使用链表，每一个实现都可以用一个实现了Queue接口的类表示；

循环数组是一个有界集合，及容量有限，如果程序中要收集的对象数量没有上限，就最好使用链表来实现；

Collection接口：在Java类库中，集合类的基本接口是Collection接口。

Iterator迭代器接口：包含方法：

E next();

boolean hashNext();

void remove();

default void forEachRemaining(Consumer<? super E> action);

调用forEachRemaining方法并提供一个lambda表达式（它会处理一个元素）。将对迭代器的每一个元素调用这个lambda表达式，直到没有元素为止；

iterator.forEachRemaining(element -> do something with element);

Iterator接口的remove方法将会删除上次调用next方法时返回的元素,next方法和remove方法调用之间存在依赖性，如果调用remove之前没有调用next将是不合法的,必须先调用next越过将要删除的元素，否则会抛出一个ILLegaStateException异常；

iterator.remove( );

iterator.next( );

iterator.remove( );

Collection<E> 的API

Iterator<E> iterator() 返回一个用于访问集合中各个元素的迭代器；

init size() 返回当前存储在集合中的元素个数；

boolean isEmpty() 如果集合中没有元素，返回true；

boolean contains(Object obj) 如果集合中包含一个与obj相等的对象，返回true；

boolean containsAll(Collection<?> other) 如果这个集合中包含other集合中的所有元素，返回true；

boolean addAll(Collection<? extends E> other) 将other集合中的所有元素添加到这个集合，如果由于这个调用改变了集合，返回true；

boolean add(E element) 将一个元素添加到集合中，如果由于这个调用改变了集合，返回true；

boolean remove(Object obj) 从这个集合中删除等于obj的对象，如果有匹配对象被删除，则返回true；

boolean removeAll(Collection<?> other) 从这个集合中删除other集合中存在的所有元素，如果这个调用改变了集合，返回true;

default boolean removeIf(Predicate<? super E> filetr) 从这个集合删除filter返回true的所有元素，如果由于这个调用改变了集合，则返回true；

void clear() 从这个集合中删除所有元素；

boolean retainAll(Collection<?> other) 从这个集合中删除所有与other集合中元素不同的元素，如果由于这个调用改变了集合，返回true；

Object[] toArray() 返回这个集合中的对象数组；

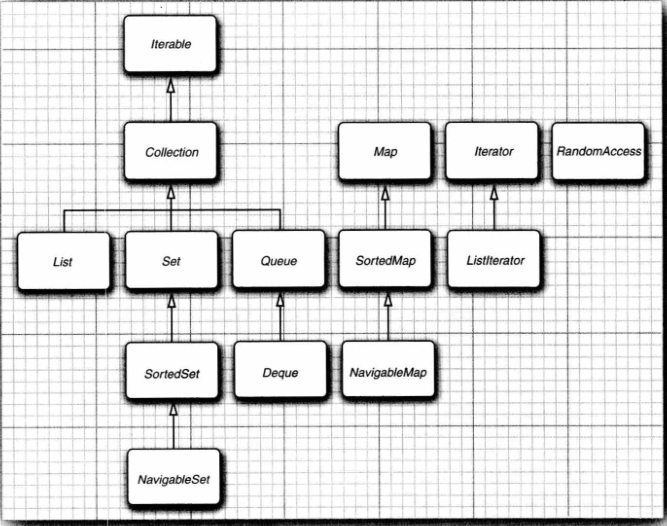
<T> T[] toArray(T[] arrayToFill) 返回这个集合中的对象的数组，如果arrayFill足够大，就将集合中的元素填入数组中，剩余空间填补null，否则，分配一个新数组，将成员类型与arrayToFill的成员类型相同，其长度等于集合的大小，并填充集合元素；

boolean hasNext() 如果存在另一个可访问的对象，返回true；

E next() 返回将要访问的下一个对象，如果已经到达了集合的末尾，将抛出一个NoSuchElementException;

void remove( ) 删除上次访问的对象，这个方法必须紧跟在访问一个元素之后执行，如果上次访问的集合已经发生变化，这个方法将抛出一个ILLegalStateException;

集合框架中的接口：



集合有两个基本接口：Collection 和 Map

boolean add(E element) 在集合中插入一个元素；

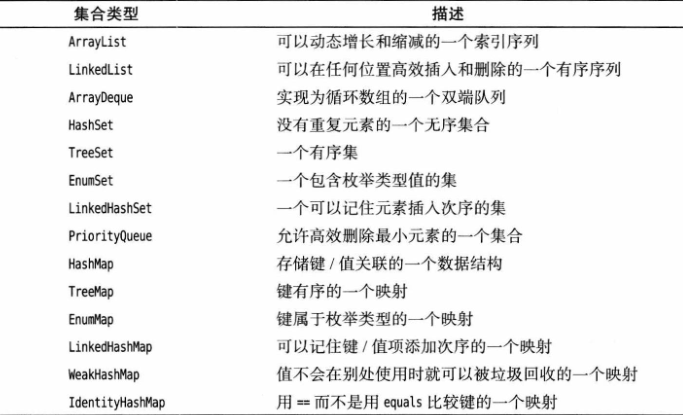
V put(K key , V value) 由于映射包含键/值对，所以要使用put方法插入;

V get(K key) 从映射中读取元素；

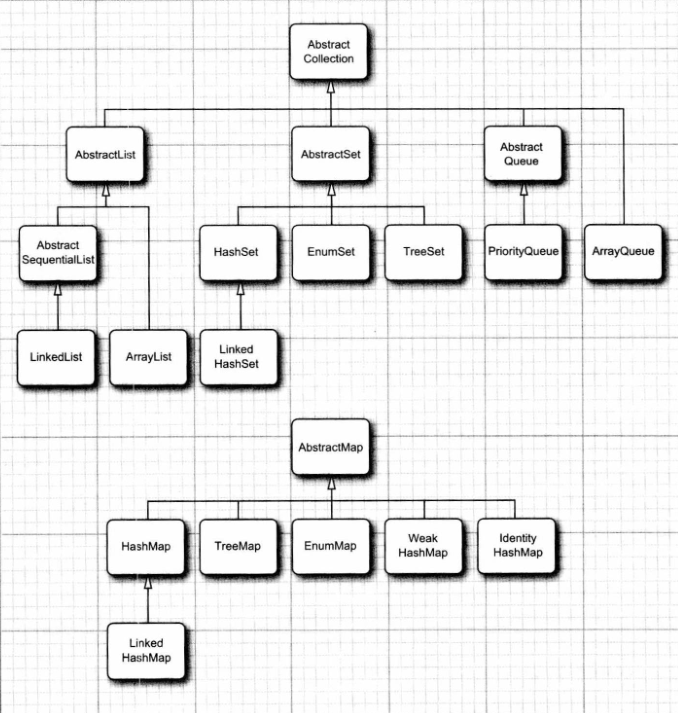
List是一个有序集合，元素会增加到容器的特定位置，采用迭代器访问（必须顺序的访问元素）和整数索引访问（可以按任意顺序访问元素）；

Set接口等同于Collection接口，集和的add方法不允许增加重复的元素

具体的集合：



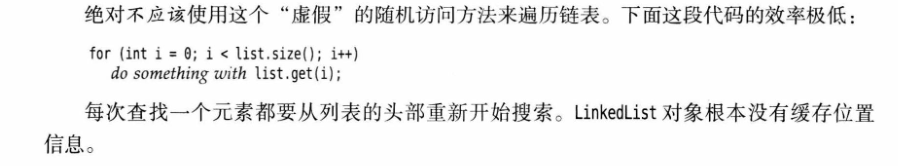
集合之间的关系：



数组和数组列表在数组中间删除或者增加一个元素的时候开销很大；

链表可以将每个对象放在单独的链接中，每个链接中还存放着序列中下一个链接的引用，链表不支持快速随机访问，如果要查看链表中的第n个元素，就必须从头开始，越过n-1个元素，所以需要按整数索引访问元素时，一般都不会使用链表，使用链表的唯一理由就是减少在列表中间插入和删除元素的开销；

链表是一个有序集合，LinkedList.add方法将对象添加到链表的尾部，链表当需要多个迭代器的时候可能会发生异常，为了避免发生并发修改异常，可以根据需要为一个集合关联多个迭代，前提是这些迭代器只能读取集合，或者，可以在关联一个能同时读写的迭代器；



java.util.List<E> API

ListIterator<E> listIterator( ) 返回一个列表迭代器，用来访问列表中的元素；

ListIterator<E> listIterator(int index) 返回一个列表迭代器，用来访问列表中的元素，第一次调用这个迭代器的next会返回给定锁定的元素；

void add(int i , E element) 在给定位置添加一个元素

void addAll(int i , Collection<? extends E> element) 将一个集合中的所有元素添加到给定的位置；

E remove<int i> 删除并返回给定位置上的元素；

E get(int i) 获取给定位置上的元素；

E set(int i ,E element) 用一个新元素替换给定位置上的元素，并返回原来那个元素；

int indexOf(Object element) 返回与指定元素相等的元素在列表中第一次出现的位置，如果没有这样的元素将返回-1；

int lastIndexOf(Object element) 返回与指定元素相等的元素在列表中最后一次出现的位置，如果没有这样的元素返回-1;

java.util.ListIterator<E> API

void add(E newElement) 在当前位置添加一个元素；

void set(E newElement) 用新元素替换next或previous访问的上一个元素，如果在上一个next或previoud调用之后列表结构被修改了，将抛出一个ILLegalStateException异常；

boolean hasPrevious() 当反向迭代列表时，如果还有可以访问的元素，就返回-1；

E previous() 返回前一个对象，如果已经到达了列表的头部，就会抛出一个NoSuchException异常；

int nextIndex() 返回下一次调用next方法时将返回的元素的索引；

int previousIndex() 返回下一次调用previous方法时将返回的元素的索引；

java.util.LinkedList<E> API

LinkedList() 构造一个空链表；

LinkedList(Collection<？ extends E> elements)构造一个链表，并将集合中所有的元素添加到这个链表中；

void addFirst(E element)

void addLast(E element) 将某个元素添加到列表的头部或尾部；

E getFirst()

E getLast() 返回列表头部或尾部的元素；

E removeFirst()

E removeLast() 删除并返回列表头部或尾部的元素；

List接口用于描述一个有序的集合，并且集合中每个元素的位置都很重要，有两种访问元素的协议：一种是通过迭代器，另一种是通过get和set方法随机访问每个元素（不适用于链表）;

ArrayLsit封装了一个动态再分配的对象数组；

Vector类的所有方法都是同步的，可以安全的从两个线程访问一个Vector对象，但是，如果只从一个线程访问Vector，代码就会再同步操作上白白浪费大量的时间。ArrayList方法不是同步的，建议在不需要同步时使用ArrayList；

散列表（HashTable）

java.util.HashSet<E> API

HashSet( ) 构造一个空的散列集；

HashSet(Collection<? extends E> element) 构造一个散列集，并将集合中的所有元素添加到这个散列集中；

HashSet(int initialCapacity) 构造一个空的具有指定容量的散列集；

HashSet(int initialCapacity , float loadFactor)构造一个有指定容量和装填因子（0.0 ~ 1.0 之间的一个数，确定散列表填充的百分比，当大于这个百分比时，散列表进行再散列）的空散列集；

int hashCode() 返回这个对象的散列码，散列码可以是任何数，包括正数或负数，equals和hashCode的定义必须兼容，即如果x.equals(y)返回true，则x.hashCode( )和y.hashCode()必须相等；

树集

树集（TreeSet）是一个有序集合，可以以任意顺序将元素插入到集合中，在对集合进行遍历的时候，值将自动地按照排序后的顺序呈现，排序是用一个树数据结构完成的（红黑树）；

将一个元素添加到树中要比添加到散列表中慢，但是，与检查数组或链表中的重复元素相比，使用树会快很多；

要使用树集，必须能够比较元素，这些元素必须实现Comparable接口，或者构造集时必须提供一个Comparator；

java.util.TreeSet<E> API

TreeSet()

TreeSet(Comparator<? super E> comparator) 构造一个空树集；

TreeSet(Collection<? extends E> element)

TreeSet(SortedSet<E> s) 构造一个树集，并增加一个集合或有序集合中的所有元素（对于后一种情况，需要使用同样的顺序）;

java.util.SortedSet<E> API

Comparator<? super E> comparator() 返回用于对元素进行排序的比较器，如果元素用Comparable接口的compareTo方法进行比较则返回null;

E first()

E last() 返回有序集中的最小元素或最大元素；

java.util.NavigableSet<E> API

E higher(E value)

E lower(E value) 返回大于value的最小元素或小于value的最大元素，如果没有这样的元素则返回null；

E ceiling(E value)

E floor(E value) 返回大于等于value的最小元素或小于等于value的最大元素，如果没有这样的元素则返回null；

E pollFirst()

E pollLasr() 删除并返回这个集合中的最大元素或最小元素，这个集合为空时返回null;

Iterator<E> descendingIterator() 返回一个按照递减顺序遍历集合中元素的迭代器；

双端队列

队列允许高效地在尾部添加元素，并在头部删除元素。双端队列允许在头部和尾部都高效地添加或删除元素。两者都不支持在队列中间添加元素；

java.util.Queue<E> API

boolean add(E element)

boolean offer(E element) 如果队列没有满，将给定的元素添加到这个队列的队尾并返回true，如果队列已经满了，第一个方法将抛出一个ILLegealStateException,第二个方法将返回false；

E remove()

E poll()

如果队列不为空，删除并返回这个队列对头的元素，如果队列是空的，第一个方法抛出NosuchElementException，第二个方法返回null；

E element()

E peek() 如果队列不为空，返回这个队列队头的元素，但不删除，如果队列为空，第一个方法抛出一个NosuchElementException，第二个方法返回null；

java.util.Deque<E> API

void addFirst(E element)

void addLast(E element)

boolean offerFirst(E element)

boolean offerLast(E element) 将给定的对象添加到双端队列的队头或队尾。如果这个双端队列已满，前面两个方法将抛出一个IllegalStateException，而后面两个方法返回false;

E getFirst()

E getLast()

E peekFirst()

E peekLast() 如果这个双端队列不为空，返回双端队列的队头元素，但是不删除，如果双端队列为空，前面两个方法将抛出一个NoSuchElementException，后面两个方法返回null;

E removeFirst()

E removeLast()

E pollFirst()

E pollLast() 如果这个双端队列不为空，删除并返回双端队列队头的元素。如果双端队列为空，前面两个方法将抛出一个NoSuchElementException，而后面两个方法返回null;

java.util.ArrayDeque<E> API

ArrayDeque()

ArrayDeque(int initialCapacity) 用初始容量16或给定的初始容量构造一个无限定双端队列；

优先队列（Priority queue）中的元素可按照任意的顺序插入，但是会按照有序的顺序进行检索，优先队列使用了堆（heap）的自组织二叉树的数据结构，它添加（add）和删除（remove）操作可以让最小的元素移动到根，不必花费时间对元素进行排序；

优先队列的典型用法是任务调度（每一个任务都有一个优先级，任务以随机顺序添加到队列中。每当启动一个新的任务时，都将优先级最高的任务从队列中删除）；

java.util.PriorityQueue API

PriorityQueue( )

PriorityQueue(int initialCapacity) 构造了一个存放Comparable对象的优先队列；

PriorityQueue(int initialCapacity,Comparator<? super E> c) 构造一个优先队列，并使用指定的比较器对元素进行比较；

映射

Java中为映射（键值对）提供了两种实现HashMap和HashTable，都实现了Map接口；

散列映射对键进行散列，树映射根据键的顺序将元素组织为一个搜索树。散列或比较函数只用于键，与键关联的值不进行散列或比较；如果不需要按照有序的顺序访问键，最好选择散列映射；

每当往映射中添加一个对象时，必须同时提供一个键。想要检索一个对象，必须使用键。如果映射中没有存储与给定键对应的信息，get将返回null，有时候返回null并不方便，可以使用一个默认值getOrDefault方法；键必须是唯一的，不能对同一个键存放两个值。如果对一个键存放两个值，第二个值就会取代第一个值。

java.util.Map<K,V> API

V get(Object key) 获取与键关联的值，返回与键关联的对象，如果映射中没有这个对象，则返回null，实现类可以禁止返回null；

default V getOrDefault(Object key,V defaultValue) 获得与键关联的值，返回与键关联的对象，如果在映射中没有找到这个键，则返回defaultValue；

V put(K key , V value) 将关联的一对键和值放到映射中，如果这个键已经存在，新的对象将取代与这个键关联的旧对象，这个方法将返回键关联的旧值，如果之前没有这个键，则返回null，实现类可以禁止键或值为null；

void putAll(Map<? extends K,? extends V> entries) 将给定映射中的所有映射条目添加到这个映射中；

boolean containsKey(Object key) 如果在映射中已经有这个键，返回true；

boolean containsValue(Object value) 如果映射中已经有这个值，返回true；

default void forEach(BiConsumer<? super K,? super V> action) 对这个映射中的所有键/值对应用这个动作；

java.util.HashMap<K,V> API

HashMap()

HashMap(int initialCapacity)

HashMap(int initialCaoacity , float loadFactor) 用给定的容量和装填因子构造一个空散列映射，默认的装填因子是0.75;

java.util.TreeMap<K,V> API

TreeMap( )为实现Comparable接口的键构造一个空的树映射；

TreeMap(Comparator<? super K> c) 构造一个树映射，并使用一个指定的比较器对键进行排序；

TreeMap<Map<? extends V> entries> 构造一个树映射，并将某个映射中的所有映射条目添加到树映射中；

TreeMap(SortedMap<? extends K, ? extends V> entries) 构造一个树映射，将某个有序映射中的所有映射条目添加到树映射中，并使用与给定的有序映射相同的比较器；

java.util.SortedMap<K,V> API

Comparator<? super K> comparator( ) 返回对键进行排序的比较器，如果键是用Comparable接口的compareTo方法比较，返回null；

K firstKey()

K lastKey()返回映射中的最小或最大键；

java.util.Map<K,V> API

default V merge(K key , V value , BiFunction<? super V , ？super V , ? extends remappingFunction>) 如果key与一个非null值v关联，将函数应用到v和value，将key与结果关联，或者如果结果为null，则删除这个键，否则，将key与value关联，返回get(key)；

defult V compute(K key,BiFunction<? super K,? super V,? extends V> remappingFunction) 将函数应用到key和get(key)，将key与结果关联，或者如果结果为null,则删除这个键，返回get(key)；

default V computIfPresent(K key,BiFunction<? super K,? super V,? extends V> remappingFunction) 如果key与一个非null值v关联，将函数应用到key和v，将key与结果关联，或者如果结果为null，则删除这个键，返回get(key)；

default V computeIfAbsent(K key,Function<? super K,? super V,? extends V> mappingFunction) 将这个函数应用到key，除非key与一个非null值关联。将key与结果关联，或者如果结果为null，则删除这个键，返回get(key)；

default void replaceAll(BiFunction<? super K,? super V,? extends V> function)在所有映射条目上应用这个函数，将键与非null结果关联，对于null结果，则将相应的键删除；

default V putIfAbsent(K key , V value) 如果key不存在或者与null关联，则将它与value关联，并返回null,否则返回关联的值；

Set<Map.Entry<K,V>> entrySet() 返回Map.Entry对象（映射中的键/值对）的一个集试图，可以从这些集中删除元素，他们将从映射中删除，但是不能添加任何元素；

Set<K> keySet() 返回映射中所有键的一个集试图，可以从这个集中删除元素，键和相关联的值将从映射中删除，但是不能添加任何元素；

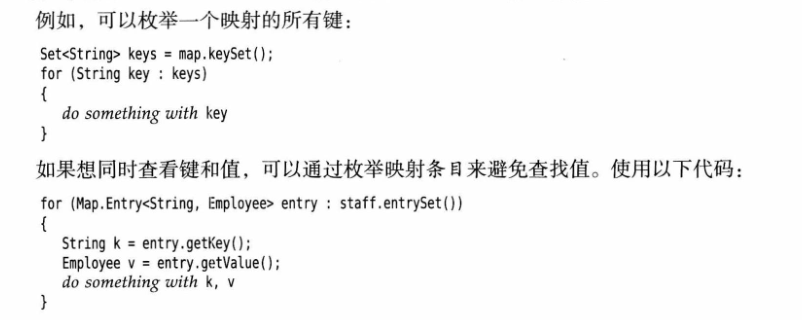
Collection<V> values() 返回映射中所有值的一个集合视图，可以从这个集中删除元素，所删除的值及相应的健将从映射中删除，不能添加任何元素；

java.util.Map.Entry<K,V> API

K getKey()

V getValue() 返回这个映射条目的键和值；

V setValue(V newValue) 将相关映射中的值改为新值，并返回原来的值；



链接散列集与映射

LinkedHashSet和LinkedHashMap类会记住插入元素项的顺序；

java.util.WeakHashMap<K , V> API

WeakHashMap( )

WeakHashMap(int initialCapacity)

WeakHashMap(int initialCapacity , float loadFactor) 用给定的容量和填充因子构造一个空的散列映射；

java.util.LinkedHashSet<E> API

LinkedHashSet( )

LinkedHashSet(int initialCapacity)

LinkedHashSet(int initialCapacity , float loadFactor) 用给定的容量和填充因子构造一个空的链接散列集；

java.util.LinkedHashMap<K,V> API

LinkedHashMap( )

LinkedHashMap(int initialCapacity)

LinkedHashMap(int initialCapacity , float loadFactor)

LinkedHashMap(int initialCapacity , float loadFactor , boolean accessOrder)

用给定的容量，填充因子和顺序构造一个空的链接散列映射，accessOrder参数为true时表示访问顺序，为false时表示插入顺序；

protected boolean removeEldestEntry(Map.Entry<K , V> eldest) 如果想要删除eldest元素，就要覆盖为返回true，eldest参数是预期可能删除的元素，这个方法在向映射中添加一个元素之后调用，其默认实现会返回false，即默认情况下，老元素不会被删除，不过，可以重新定义这个方法，以便有选择地返回true。

java.util.EnumSet<E extends Enum<E>> API

static <E extends Enum<E>> EnumSet<E> allOf(Class<E> enumType) 返回一个包含给定枚举类型的所有值的可变集；

static <E extends Enum<E>> EnumSet<E> noneOf(Class<E> enumType) 返回一个初始为空的可变集；

static <E extends Enum<E>> EnumSet<E> range(E from,E to) 返回一个包含from ~ to之间的所有值（包括两个边界元素）的可变集；

static <E extends Enum<E>> EnumSet<E> of(E e)

static <E extends Enum<E>> EnumSet<E> of(E e1,E e2,E e3,E e4,E e5)

static <E extends Enum<E>> EnumSet<E> of(E first , E… rest) 返回包括不为null的给顶元素的可变集；

java.util.EnumMap<K extends Enum<K> , V> API

EnumMap(Class<K> keyType) 构造一个键为给定类型的空的可变映射；

java.util.IdentityHashMao<K , V> API

IdentityHashMap( )

IdentityHashMap(int expectedMaxSize) 构造一个空的标识散列映射集，其容量是大于1.5 \* expectedMaxSize的2的最小幂值（expectMaxSize的默认值是21）.

java.lang.System API

static int identityHashCode(Object obj) 返回Object.hashCode计算的相同散列码（根据对象的内存地址得出），即使obj所属的类已经重新定义了hashCode方法；

同步视图

如果多个线程访问集合，就必须确保集合不会被意外地破坏；

类库的设计者使用试图机制来确保常规集合是线程安全的，而没有实现线程安全的集合类；

Collections类的静态方法synchronizedMap方法可以将任何一个映射转换成有同步访问方法的Map：



这样就可以从多线程访问这个map对象，类似get和put等方法都是同步的，即每个方法调用必须完全结束，另一个线程才能调用另一个方法；

检查类型视图可以探测将错误类型的元素混入泛型集合中的情况：



这个是他的add方法将检查插入的对象是否属于给定的类，如果不属于给定的类，就立即抛出一个ClassCastException的异常。

java.util.List API

static <E> List<E> of()

static <E> List<E> of(E e1)

static <E> List<E> of(E e1,E e2,E e3,E e4,E e5,E e6,E e7,E e8,E e9,E e10)

static <E> Set<E> of(E … elements) 生成给定元素的一个不可变的列表，元素不能为null；

java.util.Set API

static <E> Set<E> of()

static <E> Set<E> of(E e1)

static <E> Set<E> of(E e1,E e2,E e3,E e4,E e5,E e6,E e7,E e8,E e9,E e10)

static <E> Set<E> of(E … elements) 生成给定元素的一个不可变的集，元素不能为null;

java.util.Map API

static <K,V> Map<K,V> of()

static <K,V> Map<K,V> of(K k1,V v1)

static <K,V> Map<K,V> of(K k1,V v1,K k2,V v2,K k3,V v3,K k4,V v4, K k5,V v5,K k6,V v6,K k7,V v7,K k8,V v8,K k9,V v9,K k10,V v10)

生成给定键和值的一个不可变的映射，键和值不能为null;

static <K,V> Map.Entry<K,V> entry(K k,V v)

生成给定键和值的一个不可变的映射条目，键和值不能为null；

static<K,V> Map<K,V> offEntries(Map.Entry<? extends K,? extends V> … entries)

生成给定映射条目的一个不可变的映射；

java.util.Collections

static <E> Collection unmodifiableCollection(Collection<E>)

static <E> List unmodifiableList(List<E> c)

static <E> Set unmodifiableSet(Set<E> c)

static <E> SortedSet unmodifiableSortedSet(SortedSet<E> c)

static <E> SortedSet unmodifiableNavigableSet(NavigableSet<E> c)

static <K,V> Map unmodifiableMap(Map<K,V> c)

static <K,V> SortedMap unmodifiableSortedMap(SortedMap<K,V> c)

static <K,V> SortedMap unmodifiableNavigableMap(NavigableMap<K,V> c)

构造一个集合视图，视图的更改器方法抛出一个UnsupportedOperationException；

static <E> Collection<E> synchronizedCollection(Collection<E> c)

static <E> List synchronizedList(List<E> c)

static <E> Set synchronizedSet(Set<E> c)

static <E> SortedSet synchronizedSortedSet(SortedSet<E> c)

static <E> NavigableSet synchronizedVavigableSet(NavigableSet<E> c)

static <K,V> Map<K,V> synchronizedMap(Map<K,V> c)

static <K,V> SortedMap<K,V> synchronizedSortedMap(SortedMap<K,V> c)

static <K,V> NavigableMap<K,V> synchronizedableNavigableMap(Navigable<K,V> c)

构造一个集合视图，视图的方法是同步的；

static <E> Collection checkedCollection(Collection<E> c,Class<E> elementType)

static <E> List checkedList(List<E> c,Class<E> elementType)

static <E> Set checkedSet(Set<E> c,Class<E> elementType)

static <E> SortedSet checkedSortedSet(SortedSet<E> c,Class<E> elementType)

static<E> NavigableSet checkedNavigableSet(NavigableSet <E> c,Class <E> elementType)

static <K,V> Map checkedMap(Map<K,V> c,Class<K> keyType,Class<V> valueType)

static <K,V> SortedMap checkedSortedMap(SortedMap<K,V> c,Class<K> keyType,Class<V> valueType)

static <K,V> NavigableMap checkedNavigableMap(NavigableMap<K,V>,Class<K> keyType,Class<V> valueType)

static <E> Queue<E> checkedQueue(Queue<E> queue,Class<E> elementType)

构造一个集合视图，如果插入一个错误的元素，视图的方法抛出一个ClassCastException；

static <E> List<E> nCopies(int n,E value) 生成一个不可变的列表，包含n个相等的值；

static <E> List<E> singletonList(E value)

static <E> Set<E> singleton(E value)

static <K,V> Map<K,V> singletonMap(K,key,V value) 生成一个单例列表，集或映射；

static <E> List<E> emptyList()

static <T> Set<T> emptySet()

static <E> SortedSet<E> emptySortedSet()

static NavigableSet<E> emptyNavigableSet()

static <K,V> Map<K,V> emptyMap()

static <K,V> SortedMap<K,V> emptySortedMap()

static <K,V> NavigableMap<K,V> emptyNavigableMap()

static <T> Enumeration<T> emptyEnumeration()

static <T> Iterator<T> emptyIterator()

static <T> ListIterator<T> emptyListIterator()

生成一个空集合，映射或迭代器；

java.util.Arrays API

static <E> List<E> asList(E … array) 返回一个数组中元素的列表视图，这个数组时可修改的，但是其大小不可变；

java.util.List<E> API

List<E> subList(int firstIncluded ,int firstExcluded) 返回给定位置范围的所有元素的列表视图；

java.util.SortedSet<E> API

SortedSet<E> subSet(E firstIncluded,E firstExcluded)

SortedSet<E> headSet(E firstExcluded)

SortedSet<E> tailSet(E firstIncluded) 返回给定范围内元素的视图；

java.util.NavigableSet<E> API

NavigableSet<E> subSet(E from,boolean fromIncluded ,E to,boolean toIncluded)

NavigableSet<E> headSet(E to,boolean toIncluded)

NavigableSet<E> tailSet(E from.boolean fromIncluded)

返回给定范围内元素的视图，boolean 标志决定视图是否包含边界；

java.util.SortedMap<K,V> API

SortedMap<K,V> subMap(K firstIncluded , K firstExcluded)

SortedMap<K,V> headMap(K firstExcluded)

SortedMap<K,V> tailMap(K firstIncluded)

返回键在给定范围内的映射条目的映射视图

java.util.NavigableMap<K,V> API

NavigableMap<K,V> subMap(K from,boolean fromIncluded,K to,boolean toIncluded)

NavigableMap<K,V> headMap(K from ,boolean fromIncluded)

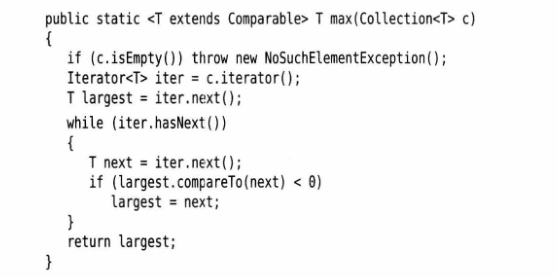
NavigableMap<K,V> tailMap(K to,boolean toIncluded)

返回键在给定范围内的映射条目的映射视图，boolean标志决定视图是否包含边界

Java集合中提供的算法

1，泛型集合算法：

计算返回链表，数组列表或数组中的最大元素：



2，排序与混排算法：

Collections类中的sort方法可以对实现了List接口的集合进行排序；

java.util.Collection API

static <T extends Comparable<? super T>> void sort(List<T> elements)

使用稳定的排序算法对列表中的元素进行排序，这个算法的时间复杂度为O(n log n),

n为列表的长度；

static void shuffle(List<?> elements)

static void shuffle(List<?> elements,Random r)随机地打乱列表中元素的顺序，这个算法的时间复杂度是O(n a(n))，n是列表的长度，a(n)是访问元素的平均时间；

default void sort (Comparator<? super T> comparator) 使用给定比较器对列表排序；

static <T extends Comparable<? super T>> Comparator<T> reverseOrder( )

生成一个比较器，将逆置Comparable接口提供的顺序；

default Comparator<T> reversed() 生成一个比较器，将逆置这个比较器提供的顺序；

3，二分查找

二分查找的前提是，待查找的元素集合必须是有序的，

static <T extends Comparable<? super T>> int binarySearch(List<T> elements,T key)

static <T> int binarySearch(List,T> elements , T key,Comparator<? super T> c)

从有序列表中搜索一个键，如果元素类型实现了RandomAccess接口，就是用二分查找，其他情况下都是用线性查找，这个方法的时间复杂度是O(a(n)logn)，n是列表的长度，a(n)是访问这个元素的平均时间，这个方法将返回这个键在列表中的索引，如果列表中不存在这个键将返回负值i，在这种情况下，这个键应该插入到索引-i-1的位置上，以保持列表的有序性；

4，简单算法

static <T extends Comparable<? super T>> T min(Collection<T> elements)

static <T ectends Comparable<? super T>> T max(Collection<T> elements)

static <T> min(Collection<T> elements , Comparator<? super T> c)

static <T> max(Collection<T> elements , Comparator<? super T> c)

返回集合中最小或最大的元素

static <T> void copy(List<? super T> to,List<T> from)

将原列表中的所有元素复制到目标列表的相应位置上，目标列表的长度至少与原列表一样；

static <T> void fill(List<? super T>l,T value)

将列表中所有位置设置为相同的值；

static <T> boolean addAll(Collection<? super T> c , T…value)

将所有的值添加到给定的集合中，如果集合改变了，则返回true；

static <T> boolean replaceAll(List<T> l,T oldValue , T newValue)

用newValue替换所有值为oldValue的元素；

static int indexOfSubList(List<?> l , List<?> s)

static int lastIndexOfSubList(List<?> l , List<?> s)

返回l中第一个或最后一个等于s的子列表的索引，如果l中不存在等于s的子列表，则返回-1；

static void reverse(List<?> l) 逆置列表中元素的顺序；

static void swap(List<?> l , int i , int j) 交换给定偏移位置的两个元素；

static void rotate(List<?> l , int d) 旋转列表中的元素，将索引i的元素移动到位置(i + d) % l.size();

static int frequency(Collection<?> c , Object o) 返回c中与对象o相等的元素；

boolean disjoint(Collection<?> c1,Collection<?> c2) 如果两个集合没有共同的元素，则返回true；

default boolean removeIf(Predicate<? super E> filetr) 删除所有的匹配元素；

default void replaceAll(UnaryOperator<E> op) 对这个列表的所有元素用这个操作；

5，批操作算法：

coll1.removeAll(coll2) 从coll1中删除coll2中出现的所有元素；

coll1,retainAll(coll2) 从coll1中删除所有未在coll2中出现的元素；

6，集合与数组的转换：

toArray方法返回的数组创建为一个Object[]数组，不改变它的类型；

如果自己编写算法，应该尽可能的使用接口，而不要使用具体的实现；

属性映射（Property Map）是一个特殊类型的映射结构：

键与值都是字符串

这个映射可以很容易地保存到文件以及从文件加载

有一个二级表存放默认值

java.util.Properties API

Properties() 创建一个空属性映射；

Properties(Properties defaults) 用一组默认值创建一个空属性映射；

String getProperty(String key) 获得一个属性，返回与键关联的值，或者如果这个键未在表中出现，则返回默认值表中与这个键关联的值，如果键在默认值表中也未出现，则返回null；

String getProperty(String key , Strubf defaultValue) 如果键未找到，获得有默认值的属性，返回与键关联的字符串，或者如果键在表中未出现，则返回默认字符串；

Object setProperty(String key , String value) 设置一个属性，返回给定键之前设置的值；

void load(InputStream in) throw IOException 从一个输入流加载一个属性映射；

void store(OutputStream out , String header) 将一个属性映射保存到一个输出流，header是所存储文件的第一行；

Properties getProperties() 获取所有系统属性，应用必须有权限获取所有属性，否则会抛出一个安全异常；

7，栈

java.util.Statck<E> API

E push(E item) 将item压入栈并返回item

E pop() 弹出并返回栈顶的item，如果栈为空，不要调用这个方法；

E peek() 返回栈顶元素，但是不弹出，如果栈为空，不要调用这个方法；

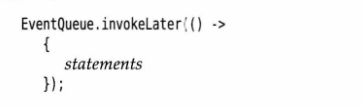
图形用户界面程序设计

Swing提供了强大的用户界面组；

JavaFX：<http://horstmann.com/corejava>

1，显示窗体（frame）

所有的Swing组件必须由事件分派线程配置这是控制线程，它将鼠标点击和按键事件传递给用户接口组件；



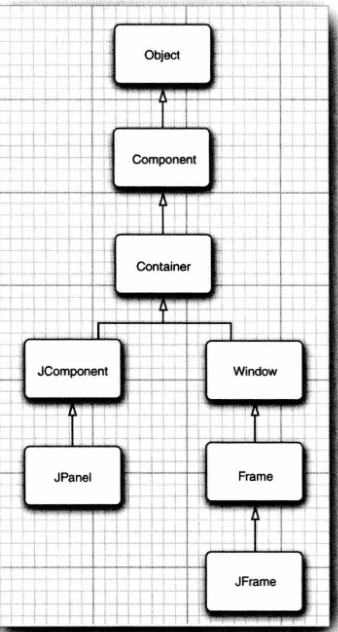
用户关闭这个窗体的相应动作：



为了显示窗体，main方法需要调用窗体的setVisible方法；

2，窗体属性

Jframe类的继承层次：



setLocation 方法和setBounds方法用于设置窗体的位置；

setIconImage方法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方法用于改变标题栏的文字；

setResizeable利用boolean值确定是否允许用户改变窗体的大小；

boolean isVisible( )

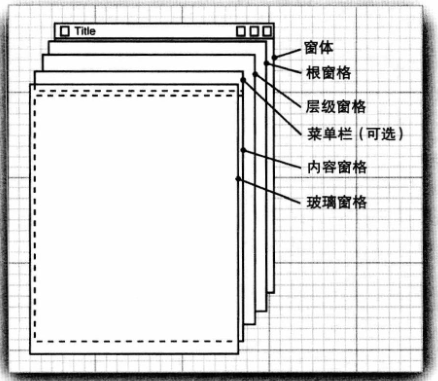
boolean setVisible(boolean b)获取或设置visible属性，组件最初是可见的；

void setSize(int width , int height) 将组件大小调整为给定的宽度和高度；

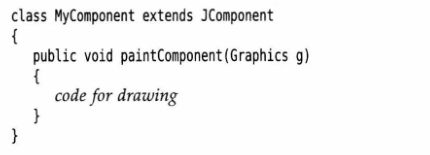
void setLocationByPlatform(boolean b) 获取或设置locationByPlatform属性，这个属性在窗口显示之前设置时，由平台选择一个合适的位置；

3，组件中显示的信息

Jframe的内部结构：



要在一个组件上绘制，需要定义一个扩展Jcomponent的类，并覆盖其中的paintComponent方法；



Component add(Component c) 讲一个给定的组件添加到该窗体的内容窗格中，并返回这个组件；

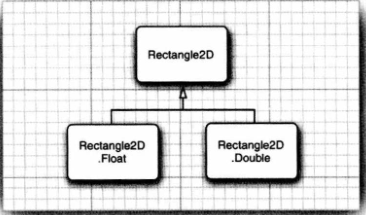
void repaint() “尽可能快的”重新绘制组件；

Dimension getPreferredSize() 覆盖这个方法，以返回这个组件的首选大小；

void paintComponent(Graphics g) 覆盖这个方法来描述需要如何绘制组件；

void pack() 调整窗口大小，要考虑其组件的首选大小；

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)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，构造一个直线

4，使用颜色

Color(int r,int g,int b) 用给定的红，绿，蓝分量（取值为0~255）创建一个颜色对象；

Paint getPaint()

void setPaint(Paint p)获取或设置这个图形上下文的绘制属性，Color类实现了Paint接口；

void fill(Shape s) 用当前的颜料填充图形；

Color getForeground()

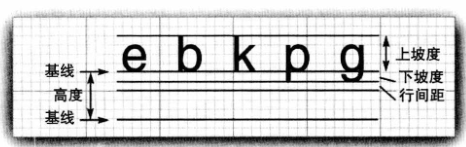
Color getBackground()

void setForeground(Color c)

void setBackground(Color c)

获取或设置前景或背景颜色；

5，使用字体



Font(String name , int style , int size) 创建一个新字体对象，字体名字可以是具体的字体名，或者逻辑名，字体风格也可以设置（Font.PLAIN , Font.BOLD , Font.ITALIC , Font.BOLD + Font.ITALIC）;

String getFontName() 获得字体名；

String getFamily() 获得字体族名；

String getName() 如果采用逻辑字体名创建字体，将获得逻辑字体名；

Rectangle2D getStringBounds(String s , FontRenderContext context) 返回包围这个字符串的矩形；

LineMetrice getLineMetrice(String s , FontRenderContext context) 返回确定字符串宽度的一个度量对象；

Font dervieFont(int style)

Font dervieFont(float size)

Font dervieFont(int style , float size) 返回一个新字体，除了有给定的大小和字体风格外，其余与原字体一样；

float getAscent() 获得字体的上坡度——从基线到大写字母顶端的距离；

float getDescent() 获得字体的下坡度——从基线到坡底的距离；

float getLeading() 获得字体的行间距——从一行文本低端到下一行文本顶端的空隙；

float getHeight() 获得字体的总高度——两条文本基线之间的距离；

FontRenderContext getFontRenderContext() 获得这个图形上下文中指定文字特征的字体绘制上下文；

void drawString(String str , float y) 采用当前的字体和颜色绘制一个字符串；

FontMetrics getFontMetrics(Font f) 获取给定字体的字体度量对象；

FontRenderContext getFontRenderContext() 返回字体绘制上下文

6，显示图像

使用ImageIcon类从文件中读取图像

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

boolean drawImage(Image img,int x,int y,int with,int height,ImageObserver observer) 绘制一个缩放或不缩放的图像；

void copyArea(int x,int y,int with,int height,int dx,int dy)复制屏幕的一个区域，dx和dy是原始区域到目标区域的距离；

事件处理：

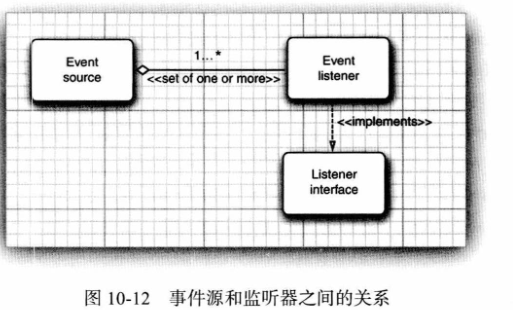
AWT事件处理机制：

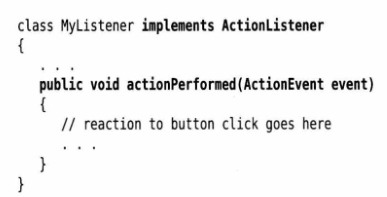
事件监听器是一个实现了监听器接口（listener interface）的类的实例；

事件源对象能够注册监听器对象并向其发送事件请求；

当事件发生时，事件源将事件对象发送给所有注册的监听器；

监听器对象再使用事件对象中的信息决定如何对事件做出响应；





JButton(String label)

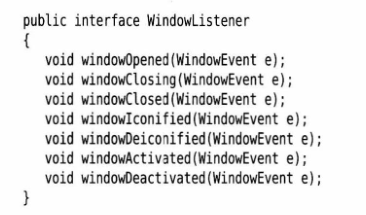
JButton(Icon icon)

JButton(String label , Icon icon) 构造一个按钮，标签字符串可以是常规的文本或HTML；

Component add(Component c) 将组件c添加到这个容器中；

适配器：

窗口监听器必须是实现了WindowListener接口的类的一个对象；

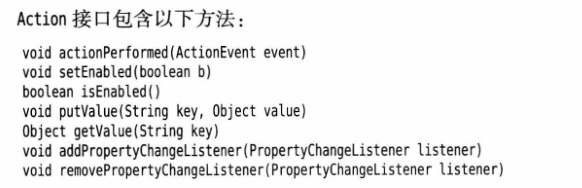


动作

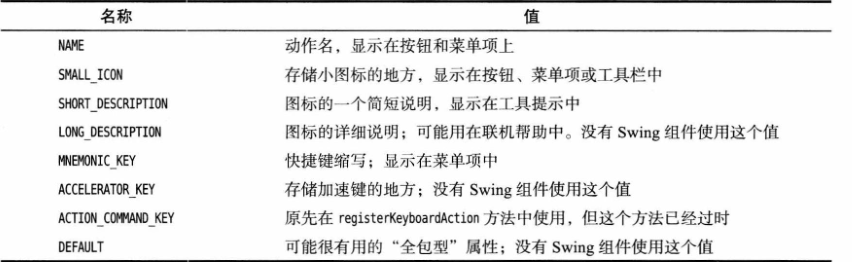
Swing包提供了一种非常实用的机制来封装命令，并将它们关联到多个事件源，这就是Action接口，动作（action）是封装以下内容的一个对象：

命令的说明（一个文本字符串和一个可选的图标）

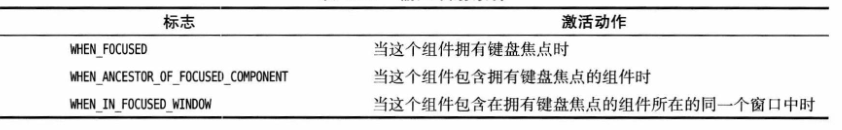
执行命令所需要的参数



预定义的动作表名：



输入映射：



鼠标事件：

mouseEntered和mouseExited两个方法在鼠标进入或移出组件时被调用；

int getX()

int getY()

Point getPoint() 返回事件发生时点（鼠标点击位置）相对于事件源组件的左上角的x和y坐标；

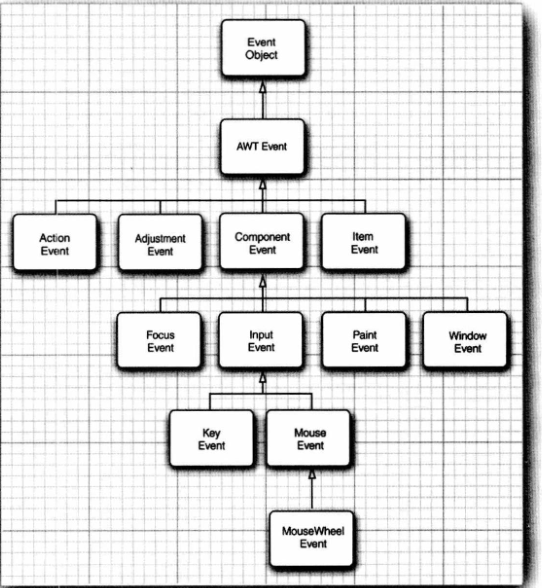
int getClickCount() 返回与事件关联的鼠标点击次数；

public void setCursot(Cursor cursor) 为指定光标设置光标图像；



AWT事件继承层次：

EventObject 类有一个子类AWTEvent，它是所有AWT事件类的父类；



常用的语义事件：

ActionEvent(对应按钮点击，菜单选择，选择列表项或在文本域中按回车)；

AdjustmentEvent(用户调节滚动条)

ItemEvent(用户从复选框或列表框中选择一项)

常用的底层事件类：

keyEvent(一键按下或释放)

MouseEvent(鼠标键按下，释放，移动或拖动)

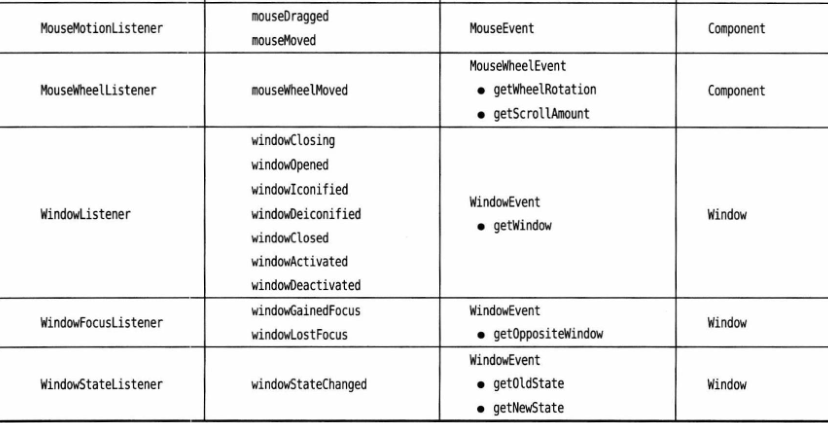
MouseWheelEvent(鼠标滚轮转动)

FocusEvent(某个组件获得焦点或失去焦点)

WindowEvent(窗口状态改变)

事件处理总结：





Swing 用户界面组件

Swing和模型-视图-控制器设计模式

设置用户界面组件的三大特征： 内容 外观 行为

设计模式 ：模型-视图-控制器模式，实现这个模型要求我们提供三个对象

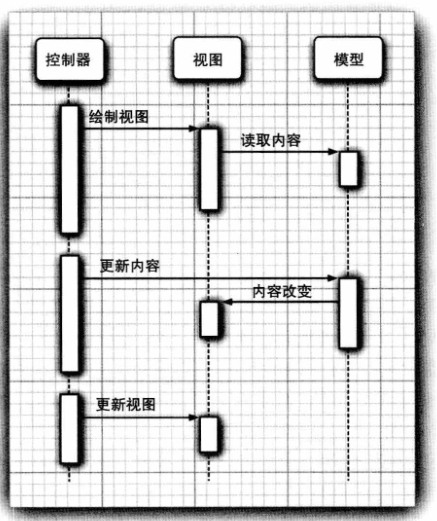
1，模型（存储内容，用户不可见），必须实现改变内容和查找内容的方法；

2，视图（显示内容）

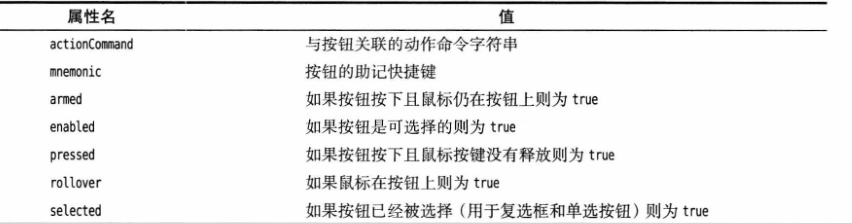
3，控制器（处理用户输入）

一个模型可以有多个视图，其中每个视图可以显示全部内容的不同部分或不同方面。

模型，视图，控制器对象之间的交互：



ButtonModel接口的属性：

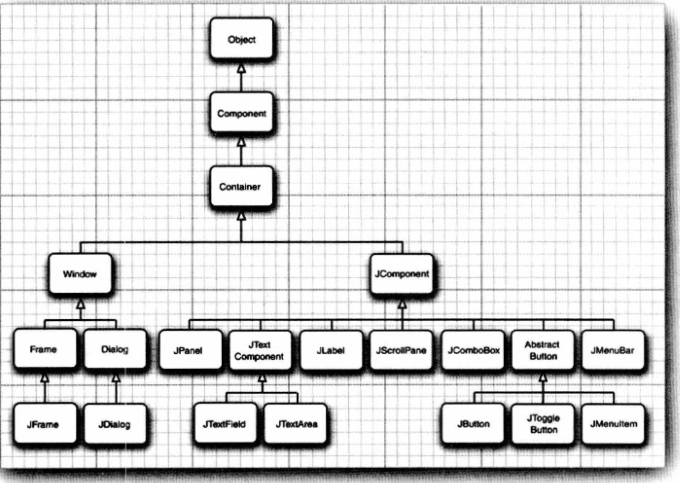


布局管理：

面板的默认布局管理器为：流布局管理器（flow layout manager）,按钮总是位于面板的中央，即使用户调整了大小;

组件放置在容器中，布局管理器决定了容器中组建的位置和大小

Component类的继承层次结构：



java.awt.Container API

void setLayout(LayoutManager m) 为容器设置布局管理器；

Component add(Comonent c)

Component add(Component c , Object constraints) 将组件添加到容器中，并返回组件引用；

java.awt.FlowLayout API

FlowLayout()

FlowLayout(int align)

FlowLayout(int align , int hgap , int vgao) 构造一个新的FlowLayout对象，align参数可以是LEFT,CENTER或者RIGHT

边框布局：

边框布局管理器（border layout manager）是每个JFrame的内容窗格的默认布局管理器，边框布局会扩展所有组件的尺寸以便填满可用空间，如果要指定组件在底面板上的放置位置，可以创建一个JPanel对象，将其添加到底面板中，然后再将组件添加到Jpanel面板上。

java.awt.BorderLayout API

BorderLayout()

BorderLayout(int hgap,int vgap) 构造一个新的BorderLayout对象；

网格布局：

网格布局就像电子数据表一样，按行列排列所有的组件，所有的组件都是一样大小的；

java.awt.GridLayout API

GridLayout(int rows , int colums)

GridLayout(int rows,int colums,int hgap,int vgap) 构造一个新的GridLayout对象，rows或者colums可以为0，但是不能同时为0，指示每行或每列任意的组件数量；

文本输入：

可以使用文本域（JtextField）和文本区（JTextArea）组件输入文本；

文本域只能输入单行文本，文本区可以输入多行文本；

JpasswordField也只能接受单行文本的输入，但是不会将输入的内容显示出来；

javax.swing,text.JtextComponent API

String getText()

void setText(String text)获取或设置文本组件的文本；

boolean isEditable()

void setEditable(boolean b)获取或设置editable特性，这个特性决定了用户是否可以编辑这个文件组件的内容；

文本域：

把文本域添加到窗口的常用办法是将它添加到一个面板或者其他容器中；

如果要将文本中内容的前后空格去掉，可以对getText的返回值应用trim方法；

javax.swing.JtextField API

JTextField(int cols) 构造一个有给定列数得空JtextField对象；

JTextField(String text , int cols) 构造一个给定列数和初始字符串的JTextField对象；

int getColumns( )

void setColumns(int cols) 获取或设置文本域使用的列数

java.awt.Jcomponent API

void revalidate( ) 重新计算组件的位置和大小；

void setFont(Font f)设置这个组件的字体；

java.awt.Component API

void validate( ) 重新计算组件的位置和大小，如果组件是容器，容器中包含的所有组件的位置和大小也会重新计算；

Font getFont( ) 获取组件的字体；

标签和标签组件：

标签是容纳文本的组件，它们没有任何的修饰，也不能响应用户的输入，可以利用标签标识组件；

要想用标识符标识这种本身不带标签的组件：

1，用正确的文本构造一个Jlable组件；

2，将它放置在距离需要标识的组件足够近的地方，以便用户看出这个标签标识的组件；

javax.swing.JLabel API

JLabel(String text)

JLabel(Icon icon)

JLable(String text , int align)

JLabel(String text,Icon icon,int align) 构造一个标签，align参数是一个SwingConstans常量：LEFT（默认），CENTER或者RIGHT；

String getText()

void setText(String text) 获取或设置标签的文本；

Icon getIcon()

void setIcon(Icon icon) 获取或设置标签的图标；

密码域：

密码域是一种特殊类型的文本域，用户输入的字符不真正显示出来，每个输入的字符都用回显字符（echo character）表示，典型的回显字符是\*；

javax.swing.JpasswordField API

JPasswordField(String text , int colums) 构造一个新的密码域；

void setEchoChar(char echo) 为密码设置回显字符；

char[] getPassword() 返回密码域中包含的文本，为了得到更好的安全性，在使用之后应该覆盖所返回数组的内容；

滚动窗格：

在Swing中，文本区没有滚动条，如果需要滚动条，可以将文本区放在滚动窗格（scroll pane）中;

滚动窗格管理文本区的视图，如果文本超出了文本区可以显示的范围，滚动条就会自动地出现，删除部分文本后，如果文本能够显示在文本区域范围内，滚动条会再次自动地消失；

javax.swing.JTextArea API

JTextArea()

JTextArea(int rows,int cols)

JTextArea(String text , int rows , int cols) 构造一个新的文本区；

void setColumns(int cols) 设置文本区应该使用的首选列数；

void setRows(int rows) 设置文本区应该使用的首选行数；

void append(String newText) 将给定的文本追加到文本区中已有文本的末尾；

void setLineWrap(boolean wrap) 打开或者关闭换行；

void setWrapStyleWord(boolean word) 如果word为true，超长的行会在单词边界换行，如果为false，超长的行会被截断而不考虑单词边界；

void setTabSize(int c) 将制表符（tab stop）设置为c列，制表符不会被转换为空格，但可以让文本对齐到下一个制表符处；

javax.swing.JscrollPane API

JscrollPanel(Component c)创建一个滚动窗格，用来显示指定的内容，当组件内容超过显示范围时，滚动条会自动出现；

选择组件：

复选框（“是”或“不是”），自带有标识标签；

JCheckBox(String lable)

JCheckBox(String lable , Icon icon) 构造一个复选框；

JCheckBox(String lable , boolean state) 用给定的标签和初始状态构造一个复选框；

boolean isSelect( )

void setSelected(boolean state) 获取或设置复选框的选择状态；

单选框 当用户选择另一项的时候，前一项就自动地取消选中，允许用户在多个选择中选择一个字体大小，每次用户只能选择一个；

javax.swing.JradioButton API

JRadioButton(String label , Icon icon) 构造一个初始没有选中的单选按钮；

JRadionButton(String label ,boolean state) 用给定的标签和初始状态构造一个单选按钮；

javax.swing.ButtonGroup API

void add(AbstractButton b) 将按钮添加到组件中；

ButtonModel getSelect( ) 返回被选中的按钮的按钮模型；

javax.swing.ButtonModel API

String getActionCommand() 返回按钮模型的动作命令；

javax.swing.AbatractButton API

void setActionCommand(String s) 设置按钮及其模型的动作指令；

边框

常用的边框选项：

凹斜面，凸斜面，蚀刻，直线，蒙版，空（只是在组件周围创建一些空白空间）；

可以给边框添加标题，实现方法是将边框传递给BorderFactory.createTitledBorder;

如果想全力使用边框，可以调用方法把边框组合起来：BorderFactory.createCompoundBorder;

调用Jcomponent类的setBorder方法将得到的边框添加到组件；

SoftBevelBorder类用于构造有柔和拐角的斜面边框，LineBorder用来构建圆角边框；

javax.swing.BorderFactory API

static Border createLineBorder(Color color)

static Border createLineBorder(Corder color , int thickness) 创建一个简单的直线边框；

static MatteBorder createMatteBorder(int top,int left,int bottom,int right,Color color);

static MattBorder createMatteBorder(int top,int left,int bottom,int right,Icon titleIcon) 创建一个用颜色或重复图标填充的粗边框；

static Border createEmptyBorder()

static Border createEmptyBorder(int top,int left,int bottom,int right) 创建一个空边框；

static Border createEtcheBorder()

static Border createEtcheBorder(Color highlight , Color shadow)

static Border createEtchedBorder(int type)

static Border createEtchedBorder(int type,Color highlight,Color shadow) 创建一个具有3D效果的直线边框，type参数的常量EtchedBorder.RAISED和EtchedBorder.LOWERED之一；

static Border createBevelBorder(int type)

static Border createBevelBorder(int type,Color highlight,Color shadow)

static Border createLoweredBevelBorder()

static Border createRaisedBevelBorder() 创建一个具有凹面或凸面效果的边框，type参数是BevelBorder.LOWERED和BevelBorder.RAISED之一；

static TitleBorder createTitleBorder(String title)

static TitleBorder createTitleBorder(Border border)

static TitleBorder createTitleBorder(Border border,String title)

static TitleBorder createTitleBorder(Border border,String titlr,int justification ,int position)

static TitleBorder createTitledBorder(Border border,String title,int justification,int position,Font font)

static TitleBorder createTitledBorder(Border border,String title,int justification , int position,Font font,Color color)

创建一个有指定属性的带标题的边框，justification参数是TitledBorder常量LEFT,CENTER,RIGHT,LEADING,TRAILING,DEFAULT\_JUSTIFICATION之一，position是ABOVE\_TOP，TOP,BELOW\_TOP,ABOVE\_BOTTOM,BOTTOM,BELOW\_BOTTOM,DEFAULT\_POSITION之一；

static CompoundBorder createCompoundBorder(Border outsideBorder,Borde insideBorder) 将两个新边框组合成一个新的边框；

javax.swing.border.SoftBevelBorder API

SoftBevelBorder(int type)

SoftBevelBorder(int type,Color highlight,Color shadow) 创建一个带有柔和边角的斜面边框，typt的参数是BevelBorder.LOWERED和BevelBorder.RAISED之一；

javax.swing.border.LineBorder API

public LineBorder(Color color , int thickness , boolean roundedCorners)

用指定的颜色和粗细创建一个直线边框，如果roundedCorners为true，则边框有圆角；

javax.swing,Jcompoment API

void setBorder(Border border) 设置这个组件的边框；

组合框

当有多个选择项，就可以使用组合框，当用户点击这个组件时，会下拉一个选择列表，用户可以选择其中的一项；如果下拉框设计为可编辑的，它将文本域的灵活性与一组预定义的选项组合起来，JComboBox类提供了组合组件；

JComboBox类是一个泛型类；

如果需要往组合框中添加大量的选项，addItem方法的性能会很差，可以构造一个DefaultComboxBoxModel，并调用addElement方法填充这个模型，然后调用JcomboBox的setModel方法；

当用户从组合框中选择一个选项时，组合框就将产生一个动作事件。

boolean isEditable()

void setEditable(boolean b) 获取或设置组合边框的editable特性；

void addItem(Object item) 把一个选项添加到选项列表中；

void insertItemAt(Object item,int index) 将一个选项插入到选项列表中的指定索引位置；

void removeItem(Object item) 从选项列表中删除一个选项；

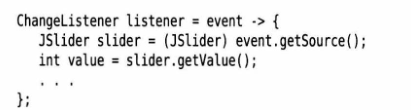
void removeItemAt(int index) 删除指定索引位置的选项；

void removeAllItems() 从列表中删除所有的选项；

Object getSelectedItem() 返回当前选择的选项；

滑动条：

滑动条允许用户从连续值中选择；



当用户滑动滑动条时，滑动条的值就会在最小值和最大值之间变化，当值发生变化时，ChangeEvent就会发送给所有变更监听器，为了得到这些变更通知，需要调用addChangeListener方法并且安装一个实现了ChangeListener接口的对象；可以通过显示刻度（tick）对滑动条进行修饰；

在滑动条真正对齐之前，变更监听器报告的滑动条值并没有对应刻度，如果点击滑动条附近，这个动作通常会让滑动条向点击的方向移动一小段距离，“对齐刻度”的滑动条并不移动到下一个刻度；

如果刻度的标记或者标签没有显示，请检查确认是否调用了setPaintTicks(true)和setPaintLabels(true)；

javax.swing.JSLider API

JSlider()

JSlider(int direction)

JSlider(int min,int max)

JSlider(int min,int max,int initialValue)

JSlider(int direction,int min,int max,int initValue)用给定的方向，最大值，最小值和初始值构造一个水平滑动条，direction参数时SwingConstants.HORIZONTAL或SwingConstants.VERTICAL之一，默认为水平，滑动条的最小值，初始值和最大默认值为0，50，100；

void setPaintTicks(boolean b)如果b为true，显示刻度

void setMajorTickSpacing(int units)

vois setMinorTickSpacing(int units) 用给定的滑动条单位的倍数设置最大可度和最小刻度；

void setPaintLabels(boolean b)如果b为true，显示刻度标签；

void setLabelTable(Dictionary table) 设置用作刻度标签的组件，表中的每一个键/值对都采用new Integer(value)/component的格式

void setSnapToTicks(boolean b) 如果b是true，每一次调整后滑块都要对齐到最接近的刻度；

void setPainTrack(boolean b) 如果b是true，显示滑动条滑动的轨迹；

菜单：

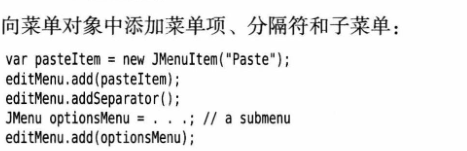
1，构建一个菜单栏：JMenuBar menuBar = new JmenuBar();

2,菜单栏可以添加到任何位置，通常放在窗体的顶部：frame.setJMenuBar(menuBar);

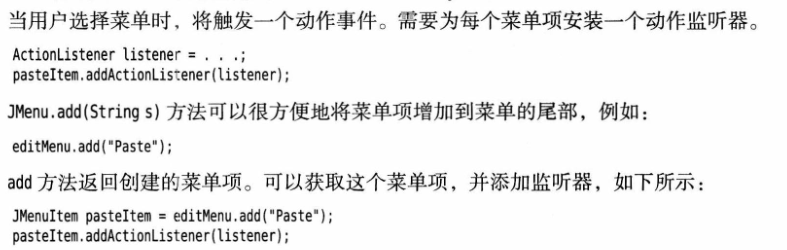
3，为每个菜单创建一个对象：JMenu editMenu = new Jmenu(“Edit”);

4，将顶层菜单添加到菜单栏中：menuBar.add(editMenu);

5，



6，



javax.swing.JMenu API

JMenu(String label) 用给定标签构造一个菜单

JMenuItem add(JMenuItem item) 添加一个菜单项（或一个菜单）

JMenuItem add(String label) 将一个有给定标签的菜单项添加到菜单中，并返回这个菜单项；

JMenuItem add(Action a)将一个有给定动作的菜单项添加到菜单中，并返回这个菜单项；

void addSeparator( ) 将一个分隔行（separator line）添加到菜单中；

JMenuItem insert(JMenuItem menu , int index)将一个新菜单项（或子菜单）添加到菜单的指引索引位置；

JMenuItem insert(Action a , int index) 将有指定动作的新菜单添加到菜单的指定索引位置；

void insertSeparator(int index) 将一个分隔符添加到菜单中；

void remove(int index)

void remove(JMenuItem item) 从菜单中删除指定的菜单项；

javax.swing,JMenuItem API

JMenuItem(String label) 用给定标签构造一个菜单项；

JMenuItem(Action a) 为给定动作构造一个菜单项；

javax.swing.AbstractButton API

void setAction(Action a) 为这个按钮或菜单设置动作；

javax.swing.JFram API

void setJMenuBar(JmenuBar menubar) 为这个窗体设置菜单栏；

javax.swing.JmenuItem API

JMenuItem(String label , Icon icon) 用给定的标签和图标构造一个菜单；

javax.swing.AbstractButton API

void setHorizontalTextPosition(int pos) 设置文本相对于图标的水平位置，pos参数是SwingConstants.RIGHT或者SwingContants.LEFT;

javax.swing.AbstractAction API

AbstractAction(String name , Icon smallIcon)用给定的名字和图标构造一个抽象动作；

复选框和单选按钮菜单项

复选框和单选按钮菜单项会在菜单名旁边显示一个复选框或一个单选框按钮，当用户选择一个菜单项时，菜单项就会自动地在选择和未选择间进行切换；

单选按钮菜单项与普通单选按钮的工作方式一样，必须将它们加入到按钮组中，当按钮组中的一个按钮被选中时，其他按钮就自动地变为未选中；

javax.swing.JCheckBoxMenuItem API

JCheckBoxMenuItem(String label) 用给定的标签构造一个复选框菜单项；

JCheckBoxMenuItem(String label , boolean state) 用给定的标签和给定的初始状态构造一个复选框菜单；

javax.swing.JRadioButtonMenuItem API

JRadioButtonMenuItem(String label) 用给定的标签构造一个单选按钮菜单项；

JradioButtonMenuItem(String label , boolean state) 用给定的标签和给定的初始状态构造一个单选按钮菜单项；

javax.swing.AbstractButton API

boolean isSelected()

void setSelected(boolean state) 获取或设置这个菜单项的选择状态；

弹出菜单

弹出菜单（pop-up menu）是不固定在菜单栏中随处浮动的菜单；

弹出菜单并不像常规菜单栏那样总是显示在窗体的顶部，必须调用show方法显示地显示弹出菜单，需要指定父组件，并使用父组件的坐标系统指定弹出菜单的位置；

当用户点击某个鼠标时弹出菜单，这是弹出式触发器（pop-up trigger）。

javax.swing.JPopupMenu API

void show(Component c , int x , int y) 在组件c上显示弹出菜单，组件c的左上角坐标为（x,y）;

boolean isPopupTrigger(MouseEvent event) 如果鼠标事件是弹出菜单触发器，则返回true；

java.awt.event.MouseEvent API

boolean isPopupTrigger( )如果鼠标事件是弹出菜单触发器，则返回true；

javax.swing.JComponent API

JPopupMenu geComponentPopupMenu( )

void setComponentPopupMenu(JpopuMenu popup) 获取或设置用于这个组件的弹出菜单

boolean getInheritsPopupMenu()

void setInheritsPopupMenu(boolean b) 获取或设置inheritsPopupMenu特征，如果这个特征已设置或这个组件的弹出菜单为null，就使用其父组件的弹出菜单；

键盘助记符和加速器

可以通过在菜单项的构造器中制定一个助记字母来为菜单设置键盘助记符。键盘助记符会在菜单中自动显示，助记字母下面有一条下划线。只能在菜单项的构造器中提供助记字母，而不是在菜单构造器中。

利用键盘助记符，可以从当前打开的菜单中选择一个子菜单或者菜单项，键盘加速器是在打不开菜单的情况下选择菜单项的快捷键。

加速器只能关联到菜单项，不能关联到菜单，加速器键并不真正打开菜单，它会直接触发与菜单关联的动作事件；

javax.swing.JMenuItem API

JMenuItem(String label , int mnemonic) 用给定的标签和助记符构造一个菜单项；

void setAccelerator(KeyStroke k) 将k键设置为这个菜单项的加速器，加速器键显示在标签旁边；

javax.swing.AbstractButton API

void setMnemonic(int mnemonic) 设置按钮的助记字符，该字符会在标签中加下划线显示；

void setDisplayedMnemonicIndex(int index) 设置按钮文本中加下划线字符的索引，如果不希望第一个出现的助记字符带下划线，就可以使用这个方法；

启动和禁用菜单项

启动和禁用菜单项有两种策略。

1，每次环境发生变化时，就可以对相关的菜单项或者动作调用setEnabled;

2，在显示菜单之前禁用这些菜单项，这个必须为“菜单选中”事件注册一个监听器；

在显示菜单之前禁用菜单项是一种明智的选择，但这种方法不适用于带有加速键的菜单项。因为在按下加速键时从来没有打开菜单，动作并没有被禁用，所以加速键还会触发这个动作；

javax.swing.JMenuItem API

void setEnabled(boolean b) 启用或禁用菜单项；

javax.swing,event.MenuListener API

void menuSelected(MenuEvent e) 在菜单被选择但尚未打开时调用；

void menuDeselected(MenuEvent e) 在菜单被取消选择并且已经关闭之后调用；

void menuCanceled(MenuEvent e) 当菜单被取消时调用；

工具条

工具条是一个按钮，通过它可以快速访问程序中最常用的命令，工具条的特殊之处在于可以将它随处移动，可以将它拖拽到窗体的四个边框上，释放鼠标按钮以后，工具条将会落在新的位置上；

只有当工具条位于采用边框布局（或者任何支持Morth,East,South,West约束的其它布局管理器）的容器内才能拖拽；

工具条可以完全脱离窗体

按钮是工具条中最常用的组件，对于工具条中可以增加那些组件并没有任何限制；

工具提示

当光标在一个按钮上停留片刻时，工具提示就会被激活，工具提示文本显示在一个有颜色的矩形里，当用户移开鼠标时，工具提示就会消失；

javax.swing.JToolBar API

JToolBar( )

JToolBar(String titleString)

JToolBar(int orientation)

JToolBar(String titleString , int orientation) 用给定的标题字符串和方向构造一个工具条，orientation可以是SwingConstants.HORIZONTAL或者SwingConstants.VERTICAL

JButton add(Action a) 在工具条中用给定动作的名称，图标，简要说明和动作回调构造一个新按钮，并把这个按钮增加到工具条末尾；

void addSeparator( )将一个分隔符添加到工具条的末尾；

javax.swing.JComponent API

void setToolTipText(String text) 设置当鼠标停留在组件上是要作为工具提示显示的文本；

复杂的布局管理

网格包布局：在网格包布局中，行和列的大小可以改变，可以将相邻的单元合并以适应较大的组件，组件不需要填充整个单元格区域，而且可以指定它们在单元格的对齐方式；

网格包管理布局的设置步骤：

1，创建一个GridBagLayout类型的对象，不需要指定底层网格的行数和列数，布局管理器会根据后面所给的信息猜测行数和列数；

2，将这个GridBagLayout对象设置为组建的布局管理器；

3，对于每个组件，创建一个GriaBagConstraints类型的对象，设置GridBagConstraints对象的字段值来指定组件在网格包中如何布局；

4，通过add(component , constraints);的调用为各个组件增加约束；

网格包布局技巧：

1，在纸上画出组件布局草图；

2，找出一个网格，小组件分别包含在一个单元格内，更大的组件会跨越越多个单元格；

3，用0，1，2，3…表示网格的行和列，现在可以得出gridx,gridy,gridwidth,gridheight;

4，对于每个组件，需要考虑以下问题：是否需要水平或者垂直填充他所在的单元格？如果不需要，希望如何对齐？

5，将所有的增量都设置为100，如果希望某行或某列始终保持默认大小，就将这行或者列中所有组件的weightx和weighty设置为0；

6，编写代码，仔细检查GridBagContraints的设置；

Java . awt . GridBagConstraints API

int gridx , gridy 指定单元格的起始行和列，默认值是0；

int gridwidth , gridheight 指定单元格行和列的范围。默认值是1；

double weightx , weighty 指定单元格扩大的容量，默认值是0；

int anchor 表示组件在单元格内的对齐方式，可以选择的绝对位置包括：

NORTHWEST NORTH NORTHEAST WEST

CENTER EAST SOUTHWEST SOUTH SOUTHEAST

或者各个方向上的相对位置：

FIRST\_LINE\_START LINE\_START FIRST\_LINK\_END PAGE\_START

CENTER PAGE\_END LAST\_LINK\_START LINE\_END LAST\_LINE\_END

如果应有要从右往左或者从上往下排列文本，就应该使用后者，默认值是CENTER；

int fill 指定在单元格内的填充行为，取值为NONE ，BOTH , HORIZONTAL ，VERTICAL，其默认值为NONE；

int ipadx , ipady 指定组件周围的“内部”填充，默认值为0；

Insets insets 指定组件边框周围的“外部”填充。默认为不填充；

GridBagConstraints(int gridx , int gridy ,int gridwidth , int gridheight , double weightx , double weighty , int anchor , int fill , Insets instes , int ipadx , int ipady) 用参数中给定的所有字段值构造GridBagConstraints ，这个构造器只用于自动代码生成器，因为它会让你的源代码很难阅读；

定制布局管理器：

可以设计自己的LayoutManager定制化管理组件类，必须实现LayoutManager接口，并且覆盖以下5个方法：

void addLayoutComponent (String s , Component c)

void removeLayoutComponent (Cpmponent c)

添加或者删除一个组件；

Dimension preferredLayoutSize (Container parent)

Dimension minimumLayoutSize (Container parent)

计算组件的最小布局和首选布局所需要的空间；

void layoutContainer (Container parent)

具体完成工作，会调用所有组件的setBounds方法；

Java . awt . LayoutManger API

void addLayoutComponent (String name , Component comp) 将组件添加到布局中；

void removeLayoutComponent (Component comp) 从布局删除一个组件；

Dimension preferredLayoutSize (Container cont) 返回这个布局中容器的首选尺寸；

Dimension minimumLayoutSize (Container cont) 返回这个布局中容器的最小尺寸；

void layoutContainer (Container cont) 在容器中摆放的组件；

对话框

AWT分为模式对话框和无模式对话框；

模式对话框：在结束对它的处理之前，不允许用户与应用程序的其余窗口进行交互，主要用于在程序继续运行之前获取用户提示的信息；

无模式对话框：允许用户同时在对话框和应用程序的其他部分输入信息；

选项对话框

showMessageDialog ：显示一条消息并等待用户点击OK，无返回值；

showConfirmDialog：显示一条消息并等待用户点击（OK/Cancel），返回所选项的一个整数;

showOptionDialog：显示一条消息并获取用户在一组选项中的选择，返回所选项的一个整数；

showInputDialog：显示一条消息并获取用户输入的一行文本，返回用户提示或者选择的字符串；

消息类型的图标：

ERROR\_MESSAGE , INFORMATION\_MESSAGE , WARNING\_MESSAGE

QUESTION\_MESSAGE , PLAIN\_MESSAGE

构造一个消息的步骤：

1. 选择对话框的类型（消息，确认，选项，输入）
2. 选择图标（错误，信息，警告，问题，无，自定义）
3. 选择消息（字符串，图标，自定义组件，集合）
4. 对于确认对话框，选择选项类型（默认，Yes/No ，Yes/No/Cancel , OK/Cancel）
5. 对于选择对话框，选择选项（字符串，图标，自定义组件）和默认选项；
6. 对于输入对话框，选择文本或者组合框；
7. 调用JobjectPanel API中的相应方法；

static void\int showMessageDialog\showConfirmDialog (Component parent , Object message , String title , int meaageType , Icon icon)

static void\int showMessageDialog\showConfirmDialog (Component parent , Object message , String title , int messageType)

static void \int showMeaageDialog\showConfirmDialog (Component parent , Object message);

static void\int showInternalMessageDialog\showComfirmDialog (Component parent , Object message , String title , int meaageType , Icone icone)

static void showInternalMessageDialog(Component parent , Object message , String title

, int messageType

static void showInternalMessage(Component parent , Object message);

显示一个消息对话框或者一个内部消息对话框（内部消息对话框完全显示在其父组件窗体内），父组件可以为null，显示在对话框中的消息可以是字符串，图标，组件或者它们的一个数组；

创建对话框

要想实现一个对话框，需要扩展Jdialog类；

步骤：

1. 在对话框构造器中，调用超类JDialog的构造器；
2. 添加对话框的用户界面组件；
3. 添加事件处理器；
4. 设置对话框的大小；

在调用超类构造器时，需要提供所有者窗体，对话框标题，模式特征；

所有者窗体控制对话显示的位置，如果提供null作为所有者，那么这个对话框将属于一个隐藏窗体；

模式特征将指定显示这个对话框时阻塞应用程序的那些其他窗口；

Javax . swing . JDialog API

public JDialog (Frame parent , String title , boolean model) 构造一个对话框，在显式显示这个对话框之前，这个对话框是不可见的；

数据交换

# 并发

1. 概念：

多线程程序在更低一层扩展了多任务的概念：单个程序看起来在同时完成多个任务。每个任务在一个线程中执行，线程是控制线程的简称；如果一个程序可以同时运行多个线程，则这个程序就是多线程的；

多进程和多线程的本质区别在于每个进程都有自己的一套变量，而线程共享数据。共享变量使线程之间的通信比进程之间的通信更加的有效，更加的容易。在有些操作系统中，线程更“轻量级”，创建，撤销一个线程比启动一个进程开销要小得多；

1. 线程：

在一个单独的线程中运行一个任务的简单过程：

1. 将执行这个任务的代码放在一个类的run方法中，这个类要实现Runable接口。Runnable接口非常简单，只有一个方法：

Public interface Runable

{

void run();

}

由于Runable是一个函数式接口，可以用一个lambda表达式创建一个实例：

Runable r = ( ) -> {task code}

1. 从这个Runable构造一个Thread对象

var t = new Thread(r) ;

三，启动线程：t . start ( ) ;

不要调用Thread类或Runable对象的run方法，直接调用run方法只会在同一个线程中执行这个任务，而没有启动新的线程，用当调用Thread . start方法，这会创建一个执行run方法的新线程；

Java.lang.Thread API

Thread(Runable target) 构造一个新线程，调用指定目标的run( )方法；

void start( ) 启动这个线程，从而调用run( )方法，这个方法会立即返回。新线程会并发运行；

void run( ) 调用相关Runable的run方法；

static void sleep (long millis) 休眠指定的毫秒数；

Java.lang.Runable API

void run( ) 必须覆盖这个方法。提供你希望执行的任务指令；

1. 线程状态

一般线程可以有6种状态：

New（新建）

Runable（可运行）

Blocked（阻塞）

Waiting（等待）

Time waiting（计时等待）

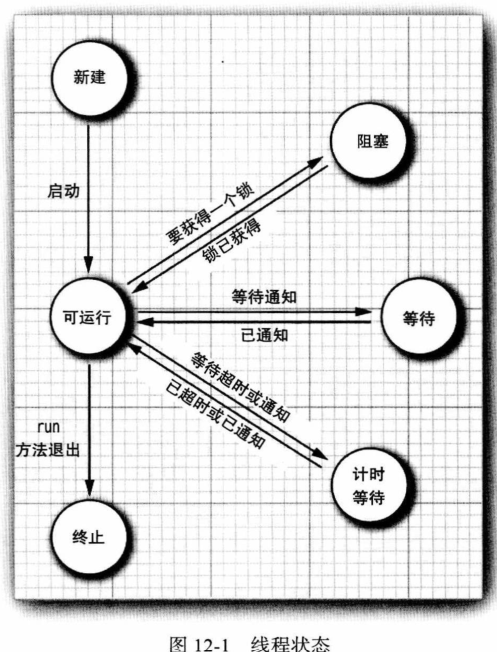
Terminated（终止）

Java.lang.Thread API

static void yield ( ) 使当前正在执行的线程向另一个线程交出运行权。这是一个静态方法；

1. 阻塞和等待线程

在线程处于阻塞或等待状态时，它暂时是不活动的。他不运行任何代码，而且消耗最少的资源，要由线程调度器重新激活这个线程；



当一个线程视图获取一个内部的对象锁，而这个锁目前被其他线程占用，该线程就会被阻塞，当所有其他线程都释放了这个锁，并且线程调度器允许该线程持有这个锁时，它将变成非阻塞状态；

当线程等待另一个线程通知调度器出现一个条件时，这个线程就会进入等待状态（调用Object . wait方法或Thread . join方法，或者是等待java . util . concurrent库中的Lock或Condition时）；

有几个方法有超时参数，调用这些方法就会让线程进入计时等待状态，这一状态将一直保持到超时期满或者接收到适当的通知（带有超时参数的方法有： Thread . sleep , Object . wait , Thread . join , Lock . tryLock , Condition . await）；

终止线程：run方法正常退出，线程自然停止；

因为一个没有捕获的异常终止了run方法，使线程意外终止；

Java.lang.Thread API

void join( ) 等待终止指定的线程；

void join (long millis) 等待指定的线程终止或等待经过指定的毫秒数；

Thread . State getState ( ) 得到这个线程的状态；

1. 线程的属性
2. 中断线程

当线程的run方法执行方法体中最后一条语句再执行return语句返回时，或者出现了方法中没有捕获的异常时，线程就会终止；

当对一个线程调用interrupt方法时，就会设置线程的中断状态。这是每个线程都有的boolean标志，每个线程都应该检查这个标志，以判断线程是否被中断；

要想得出是否设置了中断状态，首先调用静态的Thread.currentThread方法获取当前线程，然后调用isInterrupted方法：

While (!Thread . currentThread( ) . isInterrupted( ) && more work to do)

{

do more work

}

如果线程被阻塞，就无法检查中断状态。当一个被sleep 或 wait 调用阻塞的线

上调用interrupt方法时，那个阻塞调用将被一个InterruptedException异常中断。被中断的线程可以决定何时相应这个中断，而不是立即终止；

线程的run方法相应中断请求：

Runnable r = ( ) ->{

Try{

…

while(!Thread.currentThread.isInterrupted( ) && more work to do)

{

do more work

}

}catch(InterruptException){

//thread was interrupted during sleep or wait

}

finally{

cleanup , if required

}

//exiting the run method terminates the thread

};

有两个类似的方法：

interrupted : 是一个静态方法，检查当前线程是否被中断，会清理该线程的中断状态；

isInterrupted: 是一个实例方法，可以用来检查是否有线程被中断，不会改变中断状态；

使用catch 捕获线程中断异常：

在catch子句中调用Thread.currentThread( ).interrupt( )来设置中断状态，这样一来调用者就可以检测出中断状态

try {sleep (delayThread) ; }

cache(InterruptedException e) {Thread . currentThread( ) . interrupt( ); }

用throws InterruptedException 标记方法，去掉try语句块，这样调用者就可以捕获这个异常

void myThreadMethod ( ) throws InterruptedException {

sleep(delayThread);

}

Java.lang.Thread API

void interrupt( ) 向线程发送中断请求，线程的中断状态将被设置成为true。如果当前该线程被一个sleep调用阻塞，则抛出一个InterruptedException异常；

static boolean interrupted ( ) 测试当前进程是否被中断（副作用：它将当前线程的中断状态重置为false）

boolean isInterrupted( ) 测试线程是否被中断，这个调用的时候不改变线程的中断状态；

static Thread currentThread( ) 返回表示当前正在执行的线程的Thread对象；

1. 守护线程

可以通过调用t . setDaemon (true) 将一个线程转换为守护线程（daemon thread）。守护线程的唯一作用就是为其他线程提供服务。

Java . lang . Thread API

void setDaemon (boolean isDaemon) 标识该线程为守护线程或用户线程，这个方

法必须在线程启动之前调用；

1. 线程名

可以用setName方法为线程设置任何名字：

var t = new Thread(runable) ;

t.setName(“Thread-1”);

这些在线程转储的时候可能很有用；

1. 未捕获异常的处理器

在线程死亡之前，异常会传递到一个用于处理未捕获异常的处理器，这个处理器必须属于一个实现了Thread . UncaugtExceptionHandler接口的类，这个接口类只有一个方法：void uncaughtException (Thread t , Throwable e)

可以用setUncaugtExceptionHandler方法为任何线程安装一个处理器。也可以使用Thread类的静态方法setDefaultUncaughtExceptionHandler为所有线程安装一个默认的处理器。

1. 线程优先级

每个线程都有一个优先级，默认情况下，一个线程会继承构造它的那个线程的优先级，可以使用setPriority方法提高或降低任何一个线程的优先级，可以将优先级设置为MIN\_PRIORITY（1）与MAX\_PRIORITY（10）之间的任何值；

每当线程调度器有机会选择新线程时，他首先选择具有较高优先级的线程。线程优先级高度依赖于系统。

1. 同步

在多线程的应用中，两个或两个以上的线程需要共享对同一个数据的存取，当每个线程分别调用了一个修改该对象状态的方法，线程会互相覆盖，导致对象数据被破坏，这种情况一般叫竞争条件（race condition）

1. 锁对象：

使用ReentrantLock保护代码块的基本结构：

myLock . lock( );

try{

critical section;

}

finally{

myLock.unlock( ) ;

}

这个结构确保了任何时刻只有一个线程进入临界区，一旦一个线程锁定了锁对象

其他线程都无法通过lock语句，当其他线程调用lock时，它们会暂停，直到第一个线程释放锁对象。

要把unlock操作包括在finally子句中，这一点至关重要，如果在临界区的代码抛出一个异常，锁必然释放，否则，其它线程将永远阻塞；

Java.util.concurrent.locks.lock API

void lock( ) 获得这个锁，如果锁当前被另一个线程占有，则阻塞；

void unlock( ) 释放这个锁；

java.util.concurrent.locks.ReentranLock API

ReentrantLock ( ) 构造一个重入锁，可以用来保护临界区；

ReentrantLock (boolean fair) 构造一个采用公平策略的锁，一个公平锁倾向于等待时间最长的线程（公平锁可能严重影响性能，所以一般不要求锁是公平的）；

条件对象

线程在进入临界区以后却发现只有满足了某个条件之后才能执行，可以使用一个条件对象来管理那些已经获得一个锁却不能做有用工作的线程；

一个锁对象可以有一个或多个相关联的条件对象，可以使用newCondition方法获得一个条件对象；

synchronized关键字

锁用来保护代码片段，一次只能有一个线程执行被保护的代码；

锁可以管理试图进入被保护代码段的线程

一个锁可以有一个或多个相关联的条件对象；

每个条件对象管理那些已经进入被保护代码段但是还不能运行的线程；

如果一个方法声明了synchronized关键字，那么对象的锁将保护整个方法；

内部锁和条件存在一些限制：

1. 不能中断一个正在尝试获取锁的线程；
2. 不能指定尝试获得锁时的超时时间；
3. 每个锁仅有一个条件可能是不够的；

代码中锁的使用建议：

1. 最好既不是用Lock \ Condition 也不使用 synchronized关键字，在许多情况下，可以使用java . util . concurrent包中的某种机制，他会为你处理所有的锁定；
2. 如果synchronized关键字适合你的程序，就尽量使用这种做法，这样可以减少代码的编写量，还能减少出错的概率；
3. 如果特别需要Lock / Condition结构提供的额外能力，则使用Lock / Condition 。

java . lang . Object API

void notifyAll ( ) 解除在这个对象上调用wait方法的那些线程的阻塞状态，该方法只能在同步方法或同步代码块中调用，如果当前线程不是对象锁的所有者，该方法会抛出一个IllegalMonitorStateException异常；

void notify ( ) 随机选择一个在这个对象上调用wait方法的线程，解除其阻塞状态，该方法只能在一个同步方法或同步代码块中调用。如果当前线程不是对象锁的所有者，该方法会抛出一个IllegalMonitorStateException 异常；

void wait ( ) 导致一个线程进入等待状态，知道它得到通知，该方法只能在一个同步方法或同步代码块中调用，如果当前线程不是对象锁的所有者，该方法会抛出一个异常：IllegalMonitorStateException；

void wait(long millis)

void wait (long millis , int nanos)导致一个线程进入等待状态，直到它得到通知或者经过了指定的时间。这些方法只能在一个同步方法或同步代码块中调用，如果当前线程不是对象锁的所有者，这些方法会抛出IllegalMonitorStateException异常，纳秒数不能超过1000000。

同步块

当线程进入一个同步块时：

Synchronizde (obj) {

Critical section

}

会获得obj的锁；

使用一个对象的锁来实现额外的原子操作，这种方法叫客户端锁定（clientside locking）;

监视器概念

特性：

1. 监视器是只包含私有字段的类；
2. 监视器类的每个对象有一个关联锁；
3. 所有方法由这个锁锁定；
4. 锁可以有任意多个相关联的条件；

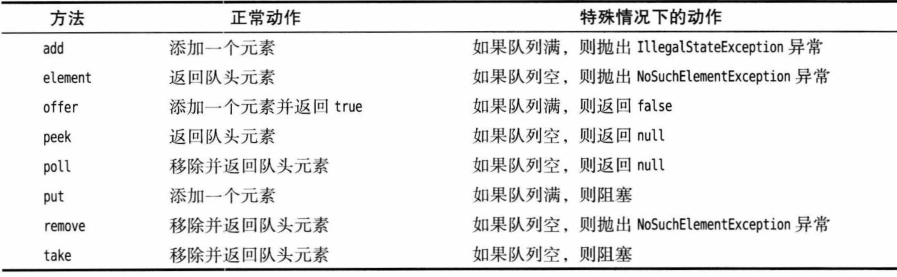
线程安全的集合

1. 阻塞队列

生产者向队列插入元素，消费者从队列获取元素。使用队列，可以安全的从一个线程向另一个线程传递数据。当尝试像一个已经满了的队列添加元素，或是从队列移除元素但是队列为空的时候，阻塞队列将导致线程阻塞；

工作线程可以周期性的将中间结果存储在阻塞队列中。

1. 阻塞队列的方法：



带有超时时间的offer方法和poll方法：

boolean success = q . offer (x , 100 , TimeUnit . MILLISECONDS) ;

尝试在100毫秒的时间内在队尾插入一个元素，如果成功返回true，否则，如果超时，则返回false；

Object head = q . poll(100,TimeUnit . MILLISECONDS);

尝试在100毫秒的时间内移除队头元素，如果成功返回队头元素，如果超时则返回null;

如果队列满了，就用put方法阻塞，如果队列为空，则用take方法阻塞，它们与不带参数的offer和poll方法等效；

LinkedBlockingQueue的容量没有上限，但是可以指定一个最大容量，是一个双端队列,实现为一个链表；

ArrayBlockingQueue 需要指定容量，有一个可选的参数来指定是否需要公平性。若设置了参数，那么等待了最长时间的线程会优先得到处理，实现为一个循环数组。

PriorityBlockingQueue是一个优先队列，不是先进先出队列，元素按照它们的优先级顺序移除，队列容量没有上限，但是如果队列为空，获取元素的操作就会阻塞；

LinkedTransferQueue 允许生产者线程等待，直到消费者准备就素可以接收元素。如果生产者调用q.transfer(item); 这个调用会阻塞，直到另一个线程将元素（item）移除。

DelayQueue( ) 构造一个包含Delayed元素的无上限阻塞队列，只有那些延迟已经到期的元素可以从队列中移除；

void transfer (E element)

boolean tryTransfer (E element , long time , TimeUnit unit) 传输一个值，或者尝试在给定的超时时间内传输这个值，这个调用将阻塞，直到另一个线程将元素删除，第二个方法会在调用成功时返回true ;

高效的映射，集合和队列

并发散列映射可以高效地支持大量阅读器和一定数量的书写器。

Java.util.concurrent.ConcurrentLinkedQueue<E> API

ConcurrentLinkedQueue<E> ( ) 构造一个可以被多线程安全访问的无上限非阻塞的队列；

Java.util.concurrent.ConcurrentSkipListSet<E> API

ConcurrentSkipListSet<E> ( )

ConcurrentSkipListSet<E>(Comparator <? super E> comp) 构造一个可以被多线程安全访问的有序集，第一个构造器要求元素实现Comparable接口；

ConcurrentHashMap<K , V>( )

ConcurrentHashMap<K , V>(int initialCapacity)

ConcurrentHashMap<K , V>(int initialCapacity , float loadFactor , int concurrencyLevel )

构造一个可以被多线程安全访问的散列引射表，默认的初始容量为15，如果每个桶的平衡负载超过装载因子，表的大小会重新调整，默认值为0.75，并发级别是估计的并发书写器的线程数；

ConcurrentSkipListMap<K , V> ( )

ConcurrentSkipListSet<K , V>(Comparator <? super K> comp) 构造一个可以被多线程安全访问的有序映像，第一个构造器要求键实现Comparable接口；

映射条目的原子更新

对并发散列映射的批操作

1. search（搜索）为每个键或值应用一个函数，直到函数生成一个非null的结果，然后搜索终止，返回这个函数的结果；
2. reduce（归约）组合所有键或值，这里要使用所提供的一个累加函数；
3. forEach为所有键或值应用一个函数；

每个操作都有4个版本：

operationKeys：处理键。

operationValues：处理值。

operation：处理键和值。

operationEntries：处理Map.Entry对象；

对于上述各个操作，需要指定一个参数阈值。如果映射包含的元素多于这个阈值，就会并行地完成批操作，如果希望批操作在一个线程中运行，可以使用阈值Long.MAX\_VALUE。如果希望用尽可能多的线程运行批操作，可以使用阈值1；