**第1章------------------------------------------------------------------------------------------------**

**Java特点:**1.简单易学(丰富API) **2.平台无关性(JVM)** 3.面向对象**4.多线程(并行机制)**5.可移植6.安全7.动态

**针对于不同的操作系统，Java提供了不同的虚拟机**

**平台：1.JDK软件开发工具包2.J2ME(Java ME)嵌入式应用开发平台3.J2SE(Java SE)桌面应用程序和低端服务器应用程序or Java Applet；4.J2EE(Java EE)企业级服务应用 上述Java运行平台都包含了相应的 JVM(Java虚拟机) JVM可直接运行字节码文件**

**JRE(Java运行环境) = JVM+核心类库+运行工具(没有编译部分) JDK(Java开发工具)=** **JVM+核心类库+开发工具（javac,java,jdb,jhat)**

**源文件(.java)通过javac编译生成字节码文件(.class),最后通过java（Java解释器）运行 jdb.exe调试器 javap.exe反编译**

**进制:二进制0b,十进制无前缀,八进制0,十六进制0x**

**标识符:数字+字母+下划线+美元符$+人民币￥[不能以数字开头,区分大小写,长度不受限制] 类名可以作为标识符，关键字不行**

**基本数据类型: boolean(1字节) byte(1个字节,8b,-128~127) short(2个字节,16b,[-32768~32767]) int (4个字节32b,-2^31~2^31-1)**

**long(8字节64b) float(4字节) double(8字节) char(2字节0-65535)最高位不是符号位 ‘\uxxx’**

**long变量数据要加L,float变量数据要加F,** **整数默认是int类型，浮点数默认是double类型 浮点数计算会产生舍入误差**

**精度顺序: double > float > long > int > short > byte 从小到大自动转换，从大到小要显式转换 char转short要显式转换**

**byte,short,char**三种类型的数据在**运算的时候**,都会**隐式转换为int**，然后再进行运算，又隐含了**强制转换为原类型**

**输出: %m.nf占m列,小数点保留n位**

**输入: Scanner reader=new Scanner(System.in); reader.nextByte(), nextShort(),nextLong(), nextFloat(), nextDouble(), nextLine()**

**While(reader.hasNextDouble())等待用户从键盘输入数据 X=reader.nextDouble();读取数据**

**数组: 声明方式int[]array= new int[3] / {1,2} / new int[]{1,2}无其他 数组是引用型变量(默认null) 相同类型包含byte等隐式转换**

**数组赋值方法：public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)**

**内存分配:栈**（操作系统分配）：**方法**运行时使用的内存 **堆**（程序员分配）:存储对象或者数组,**new来创建都存储在堆内存**

Java使用的字符集为**Unicode 数组的length方法没有() javadoc注释方式: /\*\* \*/源文件: 有且只有一个公共类,不一定有主类**

**第2章 运算符和语句-----------------------------------------------------------------------------------------------**

**&&和 || 是短路逻辑运算符**,**第一个不符合条件就不再看第二个** **&和 |** 既是逻辑运算符，也是位运算符，**两边都会判断**

**<< 算数左移 对于byte或short型数据，a<<n的运算结果是int型精度(符号拓展) 一直左移变为0**

**>> 算术右移** **正数不断右移位的结果是0，负数不断右移位的结果是-1**

**位运算符: 1.~非 2.^异或 精度：就高不就低，就长不就短**

**instanceof** 当**左边的对象**是**右边的类**创建的对象时，该运算的结果是true

**switch语句**中表达式的值必须是**整型或字符型（可以是字符串） case后的常量值必须互不相同**

**do-while** 结构结尾的while()**记得加分号;**

**第3章 面向对象-----------**-----------------------------------------------------------------------------------

**三个特性**：**封装(类)，继承，多态(操作名称的多态,继承相关的多态)**

定义类的成员变量时可以**同时赋予初值但不能有语句, 方法和变量和先后位置无关**

如果两个对象有**相同的引用**，那么就具有**相同的实体——垃圾回收机制**  **Java的编译器对空对象不做检查**

每个成员变量都有**默认值**，**引用类型默认null,**所以**无参构造方法的对象是可以直接调用成员变量**，**可以不赋值**

**关键字static修饰**的成员变量称作**静态变量/类变量 局部变量不能声明为static(只能在类中声明)**

**常量：final修饰的成员变量不占用内存**,**final成员变量，只能通过对象访问** **不一定要在声明的时候赋值,只能赋值一次**

**如果一个引用类型的变量被标记为 final，则该引用变量所指向对象的取值可以改变，确保的是引用变量的引用不可变**

**静态和实例： 1.实例变量必须通过对象访问，静态变量对象或类名都可以[实例/静态方法同理]**

**2.实例方法可以调用类中的实例方法、静态方法(类方法),操作实例变量,静态变量，静态方法只能调用类的静态方法,静态变量**

**方法重载:参数类型，参数顺序，参数个数不同**，**方法的返回类型不参与比较**

**方法重写（隐藏）：同名，参数完全相同，内部不同**

**关键字this: 可以在实例和构造方法，但不能在静态方法，局部变量的名字与成员变量的名字相同时，this不能省略**

**无参构造方法调用this(0)**，必须要有有参构造方法，不然会报错

**import语句不会影响程序运行的性能，**先到程序所在目录中寻找程序所使用的类，再到包中找

**访问权限：private：只能在类中访问 protected：允许在同一个包内访问，也允许在不同包的子类中访问**

**默认：只能在同一个包内访问 Public : 所有地方**

**注意：1.不能用protected修饰类 2.不能用private来修饰外部类，只能修饰内部类**

**基本类型封装：java.lang** **Character.toUpperCase Character.isUpperCase Long.size() 输出位数 64**

**第4章 继承---------------------------------------------------------------------------------------------**

**继承** **关键字extends 只能单继承 所有类默认继承于Object 不同包中的类名可以相同**

**继承性：同一个包：除private外** **不同包：只有public和protected继承 权限：子类方法的访问权限不能低于父类方法的访问权限**

**构造方法：子类**的**构造方法先调用父类**的某个**构造方法 父类的构造方法不会被子类继承和覆盖 如果父类只有有参，必须super！**

父类的**没有被继承的**成员变量/方法**分配了内存空间**，但它**不作为子类的成员变量**

**可以通过被子类重写的方法间接操作子类新增的成员变量**

**特殊:**子类D的**protected成员变量和方法**如果是**从父类继承的**,就要一直**追溯到该protected成员的“祖先”类A类**

**如果E类和A类在同一个包中才能访问**

**super关键字: 子类调用父类构造方法和被子类隐藏的成员，必须调用super，其他不用调用super,默认父类无参构造**

**如果调用，super必须是子类构造方法中的第一条语句**

**final类与final方法**：**final类不能被继承，即不能有子类**

**上转型对象：B是A的子类，a=b a能否访问关键看与父类有没有关系 ——作用：多态**

**抽象类:**关键字**abstract** 修饰，**不能创建对象**，**可以有构造方法给子类调用Java 8**之后可以有默认方法和静态方法

**抽象方法必须非静态！！！ 除了不能用new创建，其他和非抽象类一样**

**可以子类是抽象的，父类是具体类，子类是可以将父类中的具体方法重写并定义为抽象的public abstract double getR(); 声明;实现{}**

**接口 ：interface 声明定义 implements声明实现 如果只实现部分方法，则为抽象类**

接口中的**常量**用**public static final来修饰**，但**可以省略public static final**

接口中的**方法**用**public abstract来修饰**，但**可以省略public abstract 实现一定要用public来修饰，不可以省略**

**接口回调 ：实现接口的类**引用传值

**内部类：内部类被声明为static，才可以有静态变量和静态方法，外嵌类可以用内部类声明对象，作为外嵌类的成员**

**匿名类:不可能用匿名类声明对象,但可以直接用匿名类创建一个对象（就是一个子类）一定是内部类->可访问外嵌类成员**

**可以继承类的方法也可以重写类的方法 不可以声明静态成员变量和静态方法 抽象类和接口都可以是匿名类 用途:向方法的参数传值**

**异常类：try-catch-finally 如果有finally,finally一定会被实现**

**Exception子类之间不能有父子关系（子类在前，父类在后）**

**遇到异常不知如何处理->先抛出异常 自定义异常类：1.异常定义 class MyException extends Exception/Throwable**

**2.异常声明：void f(int n) throws MyException // 声明异常{ if(n<0)// 检测异常{MyException ex = new MyEception(n); throw(ex); // 抛出异常} } 3.捕获异常 主方法:try{} catch(MyException e) {} 记得写e**

**前后关系：1.try一旦异常后面语句不会执行 2.没有异常try执行完程序结束**

**3.进入catch(Exception e ) {} statement4; {}外面的语句也会执行,不会执行其他catch**

**泛型类：class A<E>; E可以是任何对象或接口(只能是引用类型），但不能是基本类型数据。 可以定义多个类型参数 泛型数据只能调用Object类中的方法**

**泛型接口interface Computer<E>; 泛型类+泛型接口 class Chorus<E,F> implements Computer<E,F>**

**建立具有类型安全的数据结构，它将运行时类型检查提前到编译时执行**

**泛型类的类型参数并不会在编译时都被替换成Object[替换成原始类型(原始类型通常是Object，除非有上界限制)]**

**相同参数类型的泛型基类（接口）和泛型派生类之间可以互相转换**

**String 所有都是java.lang----------------------------------------------------------**

**String常见方法：1.equals(String s) 实体(内容)是否相同 <-> s1==s2 引用是否相同 equalsIgnoreCase()忽略大小写匹配**

**String s1, s2; s1 = "How"; s2 = "How"; s1, s2具有相同的引用**

**2.length()长度 3.startsWith(String s)前缀 endsWith(String s)后4.int compareTo(String s)字典序比较大小相同0,大于正数,小于负数**

**5.int indexOf(String s)查找s位置(失败返回-1，成功返回位置) 6.String substring(int startPoint)求子串(start到最后)**

**7.String replaceAll(String s1, String s2)字符串的s1部分替换成s2 8.String trim()去除前后空格 9.charAt(int index)**

**转换：1.基本数据到String: Integer.parseInt(String s) Long.parseLong(String s) 2.转换为String String.valueOf(int/long i);**

**3.所有对象调用toString()获得字符串表示 4.类型+字符串，会自动把类型转换成字符串【底层实现 StringBuilder】**

**String不是字符数组,也不是引用类型，无法修改String的值(传值调用) 所有对字符串的修改返回新的字符串,而不会修改原字符串**

**字符数组:char a[]=s.toCharArray();字节:byte d[]= s.getBytes();**

**StringBuffer类:可修改 初始容量16个字符自动增加 方法: 1.length()长度 capacity()容量 2.charAt(int index)获得单个字符**

**3.append 其它类型数据转化为字符串再追加到StringBuffer对象 4.setCharAt(int index, char ch)设置单个字符【还是不能用数组】**

**返回对象引用——5.insert(int index, String str)插入 6.reverse()翻转 7.delete(int start, int end) 8.replace(int start, int end, String str)**

**StringBuffer是线程安全（处理多线程）的，StringBuilder不是线程安全的 append返回对象是StringBuffer,要强转回String**

**字符串分析器StringTokenizer(String s, String delim) 方法:1.nextToken()获取下一个单词,计数器自动-1 2.countTokens()获得计数值**

**While(st.hasMoreTokens()){String str=st.nextToken();}**

**Scanner充当分析器: 默认分隔符(空格,\n,\r,\t,\f) Scanner scanner = new Scanner(String s); while(scanner.hasNext() ){}**

**try{double p=scanner.nextDouble;} catch(InputMismatchException e){String t=scanner.next();} 异常可以跳过不符合部分**

**scanner.useDelimiter()**方法将一个**正则表达式**作为**分隔标记**

**模式匹配:1.正则表达式创建Pattern对象 Pattern p = Pattern.compile("A\\d"); 2.p调用matcher(CharSequence input)方法返回Matcher对象m,String类和StringBuffer类都实现了CharSequence接口 Metcher m = p.matcher(input);**

**方法: 返回boolean:1.find() 寻找和p匹配的下一子序列 2.matches()判断是否完全匹配 3.lookingAt()判断从开始位置是否有匹配的**

**4.find(int start) 返回String：5.replaceAll(String str) 匹配的部分都替换为str(input不变) 6. replaceFirst(String s)只替换第一个 while(m.find()){String str=m.group();返回匹配字符串}**

**正则表达式: 元字符：1.\\d代表0~9的任何一个数字 2.\\D代表任何一个非数字字符 3. \\s代表空格类字符,‘\t’,‘\n’,‘\x0B’,‘\f’,‘\r’**

**4.\\S代表非空格类字符 5.\\w代表可用于标识符的字符（不包括美元符号）6.\\W 代表不能用于标识符的字符**

**7.\\p{Punct}表示标点符号 !"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~**

**限定符号：1. X? X出现0次或1次 2. X\* 出现0次或多次3. X+ 出现1次或多次4. X{n} X恰好出现n次5. X{n,} X至少出现n次**

**6. X{n, m} X出现n次至m次7.[ ]代表方括号中的任何一个字符**

**[abc]：代表a, b, c中的任何一个 [^abc]：代表除了a, b, c以外的任何字符 [a-d]：代表 a至d中的任何一个**

**方括号**里允许**嵌套方括号，还可以使用“|”位运算符进行逻辑“或”运算得到一个新模式 pattern=pattern1|pattern2;**

**字符串分解:String[] split(String regex)**

**第7章 常见类--------------------------------------------------------------------------------------------**

**Date类——获取本地当前时间** **创建对象：更常用默认构造Date d = new Date(); System.out.println(d); 当前时间**

**Date(long time) time取值为毫秒 相对1970年1月1日8点 public long currentTimeMillis()获取系统当前时间的毫秒数**

**DateFormat的子类SimpleDateFormat来实现日期的格式化SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd ");**

**将时间对象转为字符串** **String time=sdf.format(Date d)**

**Calendar类 Calendar calendar = Calendar.getInstance();** **初始化一个日历对象** **calendar.setTime(new Date()); 将日历翻到一个时间**

**calendar.get(Calendar.DAY\_OF\_WEEK)-1; //获取星期 注意要-1 月份Calendar.MONTH** **从零开始 calendar.getTimeInMillis()毫秒数**

**Math类 public static double random()：产生一个0到1之间的随机数，范围 是[0,1)**  **两个静态常量,E,PI**

**BigInteger类 任意精度的整数运算 BigInteger(String val) 非数字字符会抛出异常 BigInteger.add(b)、subtract(b)、multiply(b)、divide(b)、remainder(b)、abs()、compareTo(b)（大等小于依次返回1,0,-1），pow(exp) toString()、toString(int p) p为进制**

**数字格式化 NumberFormat类** **NumberFormat f = NumberFormat.getInstance();//实例化对象**

**设置参数:f.setMaximumFractionDigits(5);** **//设置小数的最大位数（四舍五入）** **f.setMinimumIntegerDigits(3);//设置整数的最小位数**

**调用public final String format(double number)方法来格式化数字number**

**List有序集合: ArrayList<E> 动态数组 LinkedList<E>泛型类 链表**

**LinkedList<String> mylist = new LinkedList<String>(); 共有方法：**

**boolean add(E element) 末尾增加节点 void add(int index, E element) 指定位置添加节点 E get(int index): 获取对象**

**boolean addAll(int index, Collection<E> c) 指定位置添加集合**

**int indexOf(E e):返回含有数据e的节点在链表中首次出现的位置,无返-1 int lastIndexof(E e) 最后出现下标**

**E remove(int index) 移除指定下标元素，同时返回该元素 E set(int index, E e) index的对象设置为参数e指定对象，并返回**

**List<E> subList(int fromIndex, int toIndex) 返回from-(to-1)的子List**

**size()大小contains(Object obj)查找 Object clone()浅拷贝(需强转)**

**LinkedList 特有方法: void addFirst(E element) 头添加新节点 removeFirst() removeLast() getFirst() getLast()**

**iterator用法：Iterator<E> iter=mylist.iterator()，然后while(iter.hasNext()) { E temp=iter.next() }**

**Set: 无序集合HashSet<E>** **add进集合中的元素称作元素** **不允许有相同的元素（重复添加是无效的）**

**集合对象的容量会随着添加元素而增加：若集合添加的元素超过总容量的75%，集合容量加一倍**

**boolean add(E o):添加元素** **void clear()清空集合boolean contains(Object o)判断包含** **boolean isEmpty()判断空**

**boolean remove(Object o)删除**指定元素 **size()**集合**个数Object[] toArray()** 将集合**存到数组**，并返回数组 **要强转**

**boolean containsAll(HanshSet set)** 判断**集合包含集合** **Object clone():克隆**不影响原集合,**返回Object,要强转**

**boolean addAll(HashSet set) 并 retainAll(HashSet set) 交 removeAll(HashSet set) 差**

**TreeSet<E>泛型类 有序树集** **比hashmap多一个返回对象first()第一个(最小) last()最后一个,最大**

**存放到树集中的对象按对象的字符串表示升序排列 ——规定顺序两种方法: Comparable or Comparator 泛型接口**

**一：实现Comparator** **新写一个类重写public int compare(Object o1, Object o2)** **Arrays.sort(students, new StudentComparator());**

**二：实现Comparable** **原类直接重写public int compareTo(Object o) Arrays.sort(students);**

**小的在前 return this.score - stu.score; 大的在前return s.age - age**

**Stack泛型类 push() pop() empty()** **peek()获取栈顶数据** int search(Object data)返回位置,**最顶端位置是１,向下递增**,**不含数据返回-1**

**映射Map**,**存储key/value集合的容器** **无序HashMap<K,V>泛型类 散列映射 V put(K key, V value) 不是add！**

**clear()清空映射Object clone()返回克隆containsKey(Object key)包含key返回true** **containsValue(Object value)包含value返回true** **V get(Object key)返回key对应值boolean isEmpty()** **不含任何键/值对返回true int size()返回键/值对数目**

**V remove(Object key)**：**删除key指定的键/值对**,**并返回对应值 当散列映射的容量被使用了75%时，它就把容量增加到原始容量的２倍**

**遍历键值对: E key：map.keySet() 遍历key，V value:map.values() 所有的值**

**foreach遍历: HashMap<<K, V> hm = new HashMap<>(); for(Map.Entry<K, V> e : hm.entrySet()) { e.getKey(); e.getValue(); }**

**有序：TreeMap<K.V>() 默认字符串表示字典序（或K实现Comparable）or (Comparator<K> cmp) 按cmp接口规定的大小来排序**

**iterator遍历Collection<Student> collection = treemap.values(); //方法HashMap一样**

**Iterator iter = collection.iterator();**  **while(iter.hasNext()) { Student te=iter.next(); }**

**字典序**：**大写在前，小写在后**

**! HashMap.keySet() 是一个视图，不是副本，HashSet<>(G.keySet())是一个副本**

**第8章 线程------------------------------------------------------------------------------------------------**

**1个进程有多个线程，线程是最小的执行单位**——线程的**4种状态**（完整的生命周期）：

**1.新建:**此时**已经有了相应的内存空间和其他资源**

**2.运行:**线程创建后仅占有了内存资源，需要调用**start()**通知JVM，告诉JVM有个新的线程排队等候切换；JVM将CPU使用权切换给线程时，如果线程是Thread的子类，**run()**方法立刻执行。**必须在子类中重写run方法；**

**start()方法可以是一个线程成为可运行，但不一定立刻运行**

线程没有结束run()方法之前，不要能再调用start()方法

**3.中断: (1)被动中断：JVM将CPU资源从当前线程切换给其他线程**

**(2)休眠：**线程执行了**sleep(int millsecond)**立刻让出CPU——**重新进到**线程队列中**排队等待**CPU资源

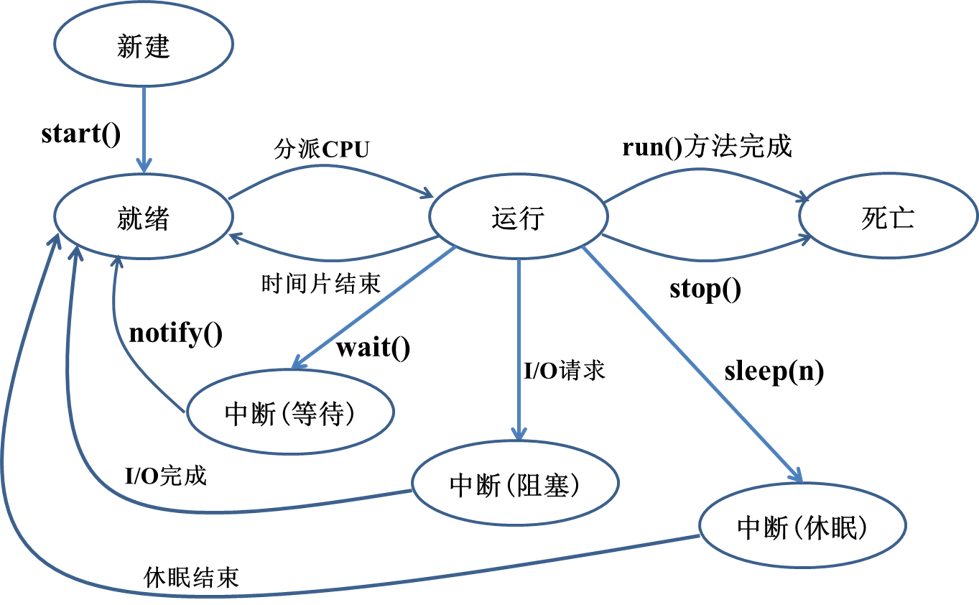
**(3)主动中断：**执行了**wait()**方法，当前线程进入**中断（等待）状态**，等待状态的线程**不会主动**进到线程队列中排队等待 CPU资源，必须由其他线程调用**notify()方法**通知它

**(4)阻塞状态 进入等待状态==重新根据优先级分配**

**4.死亡:**——释放了线程对象的内存 **1.**执行完run()方法中的全部语句

**2.**stop()被提前**强制终止**，即强制run()方法结束，让线程主动调用sleep方法让出CPU的使用权

**线程是分时的，类中定义变量(实例的域变量)是全局变量，唯一共享的，run()方法里定义是局部变量（local定义域），不共享**

使用 wait()方法可以中断方法的执行，使本线程等待，暂时让出 CPU 的使用权，并允许其它线程使用这个同步方法。其它线程如果在使用这个同步方法时不需要等待，那么它使用完这个同步方法的同时，应当用 notifyAll()方法通知所有由 于使用这个同步方法而处于等待的线程结束等待。

**线程联合：B.join()**如果线程 A 在占有 CPU 资源期间一旦联合 B 线程，那么 A 将立刻中断执行，**一直等到它联合的线程 B 执行完毕**，A 线程再重新排队等待 **主要是为了实现线程间的同步**

**守护线程：设置：thread.setDaemon(true)**;当程序中的所有用户线程都已结束运行时，即使守护线程的 run() 方法中还有需要执行的语句，守护线程也**立刻结束运行**

**优先级：**JVM中的**线程调度器**负责管理线程把线程的优先级 **10个级别（1-10），**优先级**默认为5（Thread.NORM\_PRIORITY）**；优先级可以通过**setPriority(int grade)方**法**调整**，**getPriority**方法返回，**高优先级的线程能始终运行**（若A和B的级别高于C和D，等到A和B都轮流执行完毕进入死亡状态，才会在C和D之间轮流切换,**有些操作系统只能识别3个级别：1（Thread.MIN\_PRIORITY）5和10 Thread.MAX\_PRIORITY**

**创建线程的两个途径： run方法必须是public**

**1 用Thread子类创建线程:写run()方法class Left extends Thread -> Left left=new Left();left.start();**

**2 用Thread类直接创建线程对象:**构造方法中的参数是**一个实现Runnable接口的类【implements Runnable，实现public void run()】的实例(接口回调)**，（**先创对象，再将对象传给thread对象**）

**常用方法：**

**currentThread()**方法是Thread类中的静态方法，返回**当前正在使用CPU资源的线程。**

**setName()设置名字,getName()返回名字，Thread.currentThread().getName().equals(String s)名字判断哪个线程**

**使用sleep()方法：try{Thread.sleep(800); }catch(InterruptedException e){}**

**isAlive():**判断生死——**垃圾实体：**重复分配实体会导致先前的实体变为垃圾，**不会被垃圾收集机制收集**

**interrupt():**让休眠的线程发生异常,**结束休眠,重新排队等待CPU资源**

**！！！线程同步：synchronized修饰的方法 public synchronized void run(){}**

**协调线程同步：wait(),notify(),notifyAll()是Object类中的final方法，是被所有的类继承、且不允许重写的方法**

**上面三个方法必须在synchronized代码块中使用**

**Java线程模型:代码操作的数据，CPU执行的代码，一个虚拟处理器**

**第9章 GUI--------------------------------------------------------------------------------------------------**

Java**最早**的页面库**java.awt**包,包中的类创建的组件称为**重组件(有同位体),Java2(JDK1.2)增加javax.swing包，提供Swing组件,大部分为轻组件(没有同位体)-显示和事件处理交给UI完成**->**不依赖平台,不同平台上外观相同,更高性能.**

**Object->awt.Component->awt.Container: Component**称为**组件**;**Container**称为**容器(容器也是组件,容器可嵌套)**

**添加组件add()；去除所有组件removeAll()/remove(Component a)；添加和删除后validate()确保正确显示.**

**javax.swing四个最重要的类(Container的子类)：JComponent(它的子类都是轻组件,所有轻组件都是容器)**

**[JFrame(窗口),JApplet(小应用程序),JDialog(对话框)]都是重组件,可以和操作系统交互信息,称Swing的顶层/底层容器. 默认BorderLayout 特殊：JApplet直接父类(Panel)**

**JPanel:轻组件,中间容器 默认 FlowLayout**

**JFrame:JFrame(String s),含有一个称为内容面板的重容器,应将组件添加到这.不能为JFrame窗体设置布局**

**应设置内容面板,默认布局BorderLayout 方法:getContentPane()得到内容面板；必须调用setVisible(true)窗口可见**

**中间容器：1.JPanel:创建一个面板,再向这个面板添加组件,然后把这个面板添加到底层容器或其他中间容器中, 默认BorderLayout 2.JScrollPane滚动窗格 3.JSplitPane拆分窗格:水平拆分和垂直拆分JSplitPane(int a, Component b, Component c)** **4.JLayeredPane分层窗格:add(JComponent com, int layer)添加组件com,layer有5层**

**DEFAULT\_LAYER是最底层，DRAG\_LAYER层是最上面的层，添加到同一层发生重叠,后添加的会覆盖**

**JButton JTextField JTextArea JPanel JCheckBox JRadioButton JLabel JComboBox---JComponent**

**JPasswordField getPassword()获得密码，和JText不一样**

**六个布局管理器 只能添加一个 所有布局管理器都实现了LayoutManager接口**

**1.FlowLayout布局:居中对齐,组件按照加入的先后顺序从左向右排列，一行排满之后就转到下一行继续从左至右排列 2.BorderLayout布局:Window型容器(JFrame和JDialog)的默认布局,五个区域(东西南北中) EAST, WEST, SOUTH, NORTH, CENTER表示,每个区域只能放置一个组件,若重复,后面覆盖前面**

**3.CardLayout布局:多个组件同一时刻选出一个显示占据所有空间[轻容器JTabbedPane选项卡窗格默认布局是CardLayout**

**4.GridLayout布局:若干行乘若干列的网格区域 GridLayout(int m, int n) m行n列 顺序添加**

**5.BoxLayout布局:盒式容器的默认布局是盒式布局，而且不允许更改盒式容器的布局**

**行型createHorizontalBox():上沿 列型createVerticalBox():左沿**

**6.null布局(任意布局)**

**二、ActionEvent事件处理: 事件源、监视/监听器和处理事件的接口 所有事件类的父类 AwtEvent/EventObject**

**1.事件源：**能够产生事件的对象

**2.监视/监听器：addActionListener(ActionListener listener)**

如果用户触发事件源,**Java运行系统就自动用ActionEvent类创建一个对象**,即发生了**ActionEvent事件**

**3.处理事件的接口:实现ActionListener接口**的类创建的对象,**必须提供处理事件的特定方法**

**public void actionPerformed(ActionEvent e){…}**

**public Object getSource()返回发生ActionEvent事件的对象的引用 (强转事件源类型)e.getSource()**接口回调这一过程对程序是不可见的，Java在设计组件事件时已经设置好了这一回调过程

**文本区:addDocumentListener() 组件上拖动和移动鼠标时，触发的事件是MouseMotionEvent，其他是MouseEvent**

**事件类型：WindowEvent, MouseEvent, FocusEvent, KeyEvent**

**监视器接口：WindowListener, MouseListener/MouseMotionListener, FocusListener, KeyListener**

**适配器：WindowAdapter, MouseAdapter/MouseMotionAdapter 不是所有事件都有适配器**

**事件触发：实现监视器接口or适配器的对象，接口里实现具体操作的处理方法，将组件添加到监视器 可以添加到不止一个监视器**

**窗口事件：1.WindowListener接口**：**JFrame类和JDialog是Window类的子类**，Window型对象都能触发WindowEvent事件

**addWindowListener()方法获得监视器**

**2.WindowAdapter适配器:WindowAdapter类(实现了WindowListener接口),使用WindowAdapter子类创建对象作为监视器**

**方便:接口多于一个方法,提供适配器的类,这个适配器是已经实现了相应接口的类,只是相应方法的实现内容为空.**

**AWT线程：两个重要的线程：AWT-EventQueue和AWT-Windows**

**AWT-EventQueue线程**负责处理GUI事件，通知其它线程开始运行、挂起、恢复或死亡

**AWT-Windows线程**负责将窗体或组件绘制到桌面

**事件触发：实现监视器接口or适配器的对象，接口里实现具体操作的处理方法，将组件添加到监视器**

**线程安全三个线程：1.初始化线程 2.事件调度线程 3.工作线程**

**事件冒泡机制：事件处理模型中，事件默认是从 子组件 向 父组件 传播的 有些组件不支持冒泡**

**显示图像组件：最好用Canvas**

**第10章网络编程------------------------------------------------------------------------------------------------------**

**URL：**包含最基本的三部分信息:**协议、地址、资源** 常用的**http、ftp、file协议**都是JVM支持的协议

**地址**必须是能连接的**有效IP地址或域URL类的getHost方法返回指定URL的主机名 getPort()端口号 getFile()文件名getPath()路径**

**资源**可以是主机上的任何一个**文件**

**构造方法1 public URL(String spec)throws MalformedURLException**

**try {url=new URL("http://yahoo.com.cn");} catch(MalformedURLException e){…}**

**协议:http协议**，即用户按这种协议和指定的服务器通信；**地址:**yahoo.com.cn；**资源:**默认资源(主页)

**构造方法2 public URL(String protocol, String host, String file) throws MalformedURLException**

**protocol:协议 host:地址 file:资源**

**读取URL资源：URL对象调用 InputStream openStream() 返回输入流(指向对象包含的资源)，通过输入流将服务器资源读入到客户端 （URL资源的读取可能会引起堵塞） InputStream in =url.openStream(); in.read(byte[]b);**

**显示URL资源(HTML文件):javax.swing包中的JEditorPane类可以解释执行html文件**

**public JEditorPane(URL initialPage / String url) throws IOException**

**调用public void setPage(URL page) throws IOException可以显示新的URL中的资源**

**处理超链接(HyperlinkEvent事件):->显示新的URL中的资源**

**JEditorPane对象调用addHyperlinkListener(HyperlinkListener listener)获得监视器**

**监视器需要实现HyperlinkListener接口,该接口中的方法是void hyperlinkUpdate(HyperlinkEvent e)**

**InetAddress类:对象含有一个Internet主机地址的域名和IP地址 getHostName()获得域名 ; getHostAddress()获得IP地址**

**域名服务器DNS负责将域名转化为IP地址,这样才能和主机建立连接**

**获取Internet上主机的地址:** **InetAddress.getByName(String s); InetAddress类没有构造方法 可以实现正向名称解析**

**通过静态方法获得对象 InetAddress address\_1=InetAddress.getByName("www.sina.com.cn");**

**抛出异常UnknownHostException**

**TCP套接字Socket： 1.端口号 16位的整数**0~65535,0~1023被预先定义的服务通信占用,应该使用**1024~65535**这些端口中的某一个进行通信，以免发生**端口冲突** **通过IP地址+端口号唯一确定同一个主机上运行的不同的应用程序**

**2.Socket连接 （1）服务器建立ServerSocket对象 ServerSocket对象负责等待客户端的请求，进而建立套接字连接**

**构造方法ServerSocket(int port(端口号)) try{ServerSocket a = new ServerSocket(1880); } catch(IOException e){}**

**当对象建立后，就可以使用方法accept()接收客户端的套接字连接请求** **Socket socketAtServer = a.accept();**

**返回一个Socket对象，称作服务器端的套接字对象**

**(2) 客户端创建Socket对象:构造方法 Socket(String host, int port) 指定服务器主机名称和端口**

**Socket socketAtClient = new Socket(localhost, 1880);**

**Also:无参构造public Socket() + 调用public void connect(InetAddress endpoint) throws IOException**

过程就是**向服务器发出套接字连接请求**，**如果服务器端相应的端口**上有**套接字对象正在使用accept()方法等待客户端**，那么双方的套接字对象（即socketAtClient和socketAtServer）就都**诞生**了

**（3）流连接：getOutputStream()获得输出流，getInputStream()获得输入流**

**in = new DataInputStream(socketAtServer.getInputStream());**

**out=new DataOutputStream(socketAtServer.getOutputStream());**

**服务器向这个输出流out写入信息时，客户端通过相应的输入流in读取，反之亦然**

**UDP通信： (1) 将数据打包，称为数据包，然后将数据包发往目的地。 (2) 接收别人发来的数据包，然后查看数据包中的内容**

**1.发送数据：(1)创建DatagramPacket对象将数据打包(数据包)**

**DatagramPacket(byte data[], int length, InetAddress address, int port)**

**含有data数组指定长度的数据，发送到地址是address、端口号是port的主机上**

**byte data[ ]="注意时间".getByte();**

**InetAddress address = InetAddtress.getName("www.sina.com.cn");**

**DatagramPacket pack = new DatagramPacket(data, data.length, address, 5678);**

**(2)发送数据 用DatagramSocket类的不带参数的构造方法DatagramSocket()创建一个对象a，该对象负责发送数据包 a.send(pack)**

**2.接收数据：DatagramSocket类的另一个构造方法DatagramSocket(int port)创建一个对象b，其中的参数必须和待接收的数据包的端口号相同 DatagramSocket mail\_in = new DatagramSocket(5678);**

**对象mail\_in使用方法receive(DatagramPacket pack)接收数据包 必须预备一个数据包以便收取数据包**

**使用 DatagramPack类的另外一个构造方法：DatagramPacket(byte data[],int length)创建一个数据包，用于接收数据包**

**特殊IP a.b.c.d 一个具有A、B或C类地址的主机要广播数据或接收广播，必须加入到同一个D类地址**

**1.当a小于128，那么b.c.d就用来表示主机，这类地址称做A类地址**

**2.当a大于等于128并且小于192，则a.b表示网络地址，而c.d表示主机地址，这类地址称做B类地址**

**3.当a大于等于192，则网络地址是a.b.c，d表示主机地址，这类地址称做C类地址。**

**4.224.0.0.0与239.255.255.255之间的地址称做D类地址，D类地址并不代表某个特定主机的位置**

**流---输入流读取,输出流写入 传递方式：串行-----------------------------------------------------------------------------**

**java.io 4个最重要的abstract class** **InputStream字节输入流OutputStream字节输出流Reader字符输入流Writer字符输出流**

**File类** **获取文件信息，不涉及读写** **File file= new File("c:\\myletter");** //**文件路径两种形式 (C:\\ or C:/)**

**Scanner解析File Scanner scanner = new Scanner(file)**;默认分隔符,若正则表达式**scanner.useDelimiter(“”);**

**用法：while(scanner.hasNext()){try{scanner.nextInt()} catch(InputMismatchException e) {scanner.next();}} nextInt要抛出异常**

**调用 next()方法依次返回 file 中的单词,抛出IOException int read(char b[ ], int off, int len) 读和写都可加上off和len控制范围**

**getAbsolutePath()得到绝对路径名 getAbsoluteFile()等价于new File(this.getAbsolutePath())；getName()得到不包含路径的文件名public File getParentFile()** **得到文件对象父路径名**

**文件字符流FileReader r /FileWriter w (String path/ File file) r : int read()读取一个字符 int read(char b[ ])读取b.length个字符b中**

**w:** **void write(char b[])**：**写b.length个字符到输出流** **void write(String str)字符串写入到输出流**

**缓冲流(按行读取)BufferedReader/Writer 模板**：**try{BufferedReader input = new BufferedReader(new FileReader("input.txt"));**

**BufferedWriter output = new BufferedWriter(new FileWriter("output.txt"));** String s=null; int i=0; **while((s = input.readLine())!=null)**

{ i++;**output.write**(i + ": " + s); **output.newLine();** **} output.close(); input.close();}catch(IOException e){}**

**文件字节流** **FileOutputStream**/**FileInputStream(String name||File file) 读取和写入字节(一个汉字两个字节) output.write(byte b[] )**

**while(n=input.read(b,0,2))!=-1) readInt(Boolean.../Line/UTF)()对应类型 writeByte(int) Bytes(String) writeUTF(String)**

**数据流**：**处理所有基本数值类型数据DataIn(Out)putStream(In(Out)putStream** **s) 上面write方法会隐式转换**

**FileOutputStream fs** = new FileOutputStream("jerry.dat"); **DataOutputStream output = new DataOutputStream(fs);**

**对象流：ObjectInputStream** **writeObjece(obj) readObject() 该类要实现Serializable接口才能被序列化**,**不用实现额外方法**

**数组字节流ByteArrayOutputStream** out = new ByteArrayOutputStream( || int size) 指向**一个默认大小为32个字节(or size)的缓冲区**

**数组字节流读写操作不会发生IOException异常ByteArrayInputStream** in = new ByteArrayInputStream(**out.toByteArray()**);

**数组字符流 CharArrayReader(Writer) 可能发生IOException异常 output.toCharArray());**

**RandomAccessFile** 在文件**任意位置**进行读写 **(String name, String mode)** **mode取r（只读）或rw（可读写）**，

**f.seek(f.length()-1)** 将文件指针指向**最后一个字符前面**

**BufferedInputOutputStream** **提高处理二进文件的效率**

文本

描述已自动生成图标

描述已自动生成文本

描述已自动生成图片包含 文本

描述已自动生成文本

描述已自动生成**文本

描述已自动生成**文本

中度可信度描述已自动生成文本

描述已自动生成文本

描述已自动生成文本

描述已自动生成**图形用户界面, 文本, 应用程序

描述已自动生成文本

描述已自动生成**图标

描述已自动生成文本

中度可信度描述已自动生成文本

描述已自动生成文本

描述已自动生成**文本

中度可信度描述已自动生成**文本

描述已自动生成