Oracle 勇于通过代价高昂的法律诉讼维护其知识产权——它与 [Google](http://www.yidianzixun.com/channel/w/google) 围绕 Android 的官司就是证明。那次官司的结果是法律不支持 Java API 被复制或分支（copy/fork），也不支持通过封装或重命名的方式移为他用。Java Community Process 是目前唯一可以改变该语言核心或标准 API 的方式。第三方供应商若想开发 Java 工具并大量发售，必须获得（大多是以购买）Oracle 的许可。

java中建议数据成员命名的时候不要加上m\_，类的构造函数中通过this.value=value来进行赋值。

idea java gradle项目中获取resources文件夹下文件：

xxxx.class.getClassLoader().getResourceAsStream("dura\_dist.txt"); 获取内容

xxxx.class.getClassLoader().getResource("dura\_dist.txt").getPath(); 获取绝对路径

idea打开gradle项目，要么open-》xxx.gradle不能是项目文件夹，要么import project

-1. 从语言排名、发展情况、人才需求、公司使用、平台通用性的情况来看，开源的Java没有过时

但是Java已经存在了21年，写起来复杂难受，比如getter、setter函数，不少操作都需要编写比其他语言冗长的代码。相比较Java我更喜欢scala、kotlin，100%兼容Java。在我看来这就是一门过时的语言，因为语法过时，但是除此之外有很多能够被借鉴的东西来给新语言参考。python3就解决了不少python2中的语法问题，当时3不兼容2，有不少人吐槽这件事情，但是时间证明这就是对的。

0.Java applet：Java在web里的小程序

Android会使用到Java里面常用的库，但是只是这样。所以我就没有怎么学习JavaFX，这个时下流行的java窗口。说C++：用过JNI的人都知道本地的C/C++库，至少要编译多次。不同的框架就代表者不同的编译环境。而且直接用C++编译运行的话。C++并不能很好的屏蔽掉硬件差异。所以对于开发来说调试来说，都是一个比较艰巨的任务(说到底还是编译组的锅)。所以C++没有被选为Android的开发语言。

Java SE 是客户端。Java EE 是服务器端。Java ME是微型版，嵌入式开发。

JavaFX 界面的制作比Java Swing好

Java9 引入jShell，这种做法方便初学者或者来复习一些东西。

1. .java文件-》.class文件然后和库文件一起转到JVM里面（类加载器），同时会对代码进行字节码验证和安全审计。

2.java的import java.xxxx导入类就好比C++里面的#inlcude<头文件> using xxxxx一样，之后的使用可以简化名字，否则要使用全名。import 使用通配符就好比C++ using了命名空间一样。java包的思想比C++命名空间好多了，前者import只需要一步就好，就可以满足解析的需求。

java会默认导入java.lang和当前java所在文件夹的包

3.java从键盘缓冲区读取输入：

/\*在idea中，在主函数中写一个Scanner，然后就会自动加上对应的import\*/

import java.util.Scanner; //这是一个用于扫描输入文本的类

public class ComputerAverage { //类名的命名规则

public static void main(String[] args) { //args.length 基本遇到的都是length函数，这里却是成员

//Integer.parseInt(args[0])+1

//void main通过System.exit(flag ? 1:0);返回不同的结果值，不能通过return

Scanner input =new Scanner(System.in);//不要同时声明多个这个，又没有锁。扫描输入流

System.out.print("Enter three number");//和System.in一样，估计是个对象

double number1 =input.nextDouble();//方法是从第二个单词开始大写

double number2 =input.nextDouble();

double number3 =input.nextDouble();

double average =(number1+number2+number3)/3;

System.out.println("The average of three number is"+average);//默认就是String类型，而不是char\*，所以可以直接+。

if(args.length!=2) { //不是4

System.out.println("Usage: java Exercise12\_11 John filename");

System.exit(1);

}

}

}

4.使用final来修饰这是一个变量设定为常量！如果修饰一个类表示这个类不能被继承。修饰方法表示该方法不会被覆盖。即做出不会被改写的。

Math.PI 对pi的较长的一个final double 定义。

对于数字常量，进制的设置在前面：

这倒是和C格式上一样，但是不同汇编和一般写法。

0B11111111

0377

0x(0X)EE

5.Java基本类型：

byte 1

short 2

int 4

long 8

//以上做法纠正了C里面多余的一种类型，去掉了long long

float 4 小数点后有效位数7-8 后缀是f F

double 8 小数点后有效位数15-17 后缀是d D

/\*没有long double\*/

boolean 布尔类型，显示输出就是True或者Flase。

char 两字节，因为使用了Unicode。

所以对应的读取方法就是：

.nextByte()

.nextShort()

.nextInt()

.nextLong()

.nextFloat()

.nextDouble()

.nextBoolean() 输入true、false

6.标识符（命名规则）：

由字母、数字、下划线(\_)和美元符号($)构成的字符序列，而且不以数字开头，书上写着是任意长度。

6.5返回系统分隔符：

System.getProperty("line.separator"); 底层猜测使用map实现

7.伪随机数：

java.lang.Math.Random; 默认可以直接使用

Math.random()返回一个[0,1)小数。所以random\*101就是得到一个[0,100]之间的数字。'a'+Math.random\*('z'-'a'+1)得到一个小写字母随机数。

System.currentTimeMillis() //话说形容词在time后面，java中的随机数不需要初始化种子

8.范围大的类型转换为范围小的类型必须要使用显示强转。

9.Math类在java.lang包里面，而这个包是自动导入的。

三角函数(静态方法)：

输入弧度sin() cos() tan() asin()

acos() atan() toDegrees()

输入角度转换为弧度toRadians()

10.Java支持Unicode。Unicode一开始被设计为16位的字符（所以Java里的char就是两字节）后来Unicode加入了大量的其他字符，成为supplementary character。Unicode兼容0x7F以前的ASCII编码。Java中16位的Unicode编写格式是\uFFFF这样子，也有\b \t \n \f \r \\ \"。

11.Character类的静态方法： char是基本类型而不是类，所以按照java的思想就需要一个新的类来装载这些方法。

java.lang.Character 所以可以直接使用

isDight()

isLetter()

isLetterOrDight()

isLowerCase()

isUpperCase()

使用举例：

System.out.println(""+isUpperCase('a'));

12.静态方法也可以使用对象来调用（Java里面声明一个对象就可以了）。Java里面直接调用是类名.静态方法，C++中间是::。

13.String类：

"xxx"包括的就是String类的一个对象

""是空串

通过构造函数将一个char[]转为String，那么这个char[]有多大，转换出来的String就会有多长。

信息获取：

.length()

.charAt(index) //代替索引 并且char可以和string相加 不存在[]

判断：

.equals(s1)

.equalsIgnoreCase(s1)

.compareTo(s1) //String类型，内容的比较要用equals、compareTo函数（返回Boolean），不能用==。Byte.equals方法结果为true的前提是比较双方均为Byte类型。compare 比较的才是值。

.compareToIgnoreCase(s1)

.startWith(s1)

.endsWith(s1)

.contains(s1) //字串检测

转换和结合：

.concat(s1)

.toUpperCase()

.toLowerCase()

.trim() //返回一个新字符，去掉两边的空白字符。

子串获取： //这里面第二个s真的是小写

.substring(beginIndex)

.substring(beginIndex, endIndex)

搜索：

.indexOf(ch) //没找到返回-1。这些返回值都是int，所以不要是使用short去接受、

.indexOf(ch, fromIndex) //fromIndex之后出现的第一个

.indexOf(str)

.indexOf(str, fromIndex)

.lastIndexOf(ch)

.lastIndexOf(ch, fromIndex) //fromIndex之前的最后一个

.lastIndexOf(str)

.lastIndexOf(str, fromIndex)

替换： 创建一个新的字符串对象

replace()

replaceAll() 正则表达式 替换所有

replaceFirst() 正则表达式 但是只替换一个

划分：

split(string)

正则匹配：

.matches("Java")

.matches(".\*")

\\d表示数字，\\d{3}表示3个连续数字（看来会单独对字符串先进行一次转义）

.匹配除“\r\n”之外的任何单个字符。要匹配包括“\r\n”在内的任何字符，请使用像“[\s\S]”的模式。

"abc def".replaceFirst("(\\w+)\\s+(\\w+)", "$2 $1"); //结果为 def abc

"abc def aaa bbb".replaceAll("(\\w+)\\s+(\\w+)", "$2 $1"); //结果是 def abc bbb aaa

一般的email,形如zhangshan@163.com,abc@sina.com.cn这样一些常用的形式就行了,但是在我们公司的一些客户中邮箱却有一些zhangshna.Mr@163.com,abc\_Wang.dd@sian.com,abc\_Wang.dd.cc@sian.com这 种类似的形式,在@符号之前还有点.,原来是拿来就用,可是现在就不行,得自己研究正则的用法了

原来的正则表达式/^[a-zA-Z0-9\_-]+@[a-zA-Z0-9\_-]+(\.[a-zA-Z0-9\_-]+)+$/; 可以改为

/^(\w)+(\.\w+)\*@(\w)+((\.\w{2,3}){1,3})$/;或者/^(\w)+(\.\w+)\*@(\w)+((\.\w+)+)$/;

\w就是[a-zA-Z0-9\_]，当然在Java里面就是\\w

与数据：

.toCharArray()

静态方法初始化：(使用数组构造String类)

valueOf(new char[] {'J', 'a', 'v', 'a'})

数据类型转化为String：

基本类型使用""+数值

包装类使用toString()

还可以使用更加进一步的格式化数据类型方法：format() 。似乎C++的stringstream有这种，在c里面是sprintf函数。

15.三种print输出函数：都是System类的方法：

.print()

.println() //ln是line，也就是输出以后换行

.printf() //f是format，格式化输出

System.out.printf("输出一个浮点数：%f，一个整数：%d，一个字符串：%s",d,i,s);

System.out.printf("字符串：%2$s，%1$d的十六进制数：%1$#x",i,s);

17.被重载的方法必须具有不同的参数列表。不能基于不同修饰符或返回值类型来重载方法。

18.编程格式：

public class RandomCharacter {

private char getRandomCharacter(char ch1, char ch2) {

if( ch1 > ch2 )

ch1^=ch2^=ch1^=ch2;

return (char)(ch1+Math.random\*(ch2-ch1+1));

}

public char getRandomLowerCaseLetter() {

return getRandomCharacter('a', 'z');

}

public char getRandomUpperCaseLetter() {

return getRandomCharacter('A', 'Z');

}

}

//但是ch1会变成0，ch2能获取ch1的值

估计是因为jvm的优化，使用“java -XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly Main” 查看汇编代码，我查xor指令只查到对自己的置零异或。

14.数组的声明：elementType[] arrayRefer;

二维数组的声明：elementType[][] arrayRefer;不建议使用elementType arrayRefer[][];

使用new elementType[arraySize]来构建数组，然后传过去给arrayRefer。数组的名字在Java里面是种引用，其实我认为应该是智能指针。有一块数组块却已经没有被任何名字引用到，就会被Java回收。虽然java有着不错的内存垃圾回收机制，但是这并不影响内存泄露的发生。

只能int[][] test =new int[2][2]; 不能int[2][2] test;

15.Java中默认数据类型初始化为0，char类型初始化为'\u0000'，boolean初始化为flase。引用类型会被默认初始化为null（String和数组）。在if中都能被转化为false。

如果使用final修饰的变量不会被默认初始化，所以一旦自己没有直接赋值或者在构造函数中赋值就会报错。

16.Java里面的foreach： for(elementType element:xxxx) {...} 同C++

17.复制数组有四种方法：

1.使用循环对元素进行复制

2.System.arraycopy 静态方法 arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length)

这个方法的命名目前唯一违反了Java规则

3..clone方法（类的克隆）

4.序列化与反序列

19.function(new int[]{1,2,3,4,5})这种用new创建的，不同于C++，Java里面的new创建出来，Java还是可以垃圾内存回收的。

20.Java可变参数列表：类型名...变量名

要求：只能表达同类型，而且必须要是最后一个参数（所以一个函数形参里面只能有一个这个）

public static void printMax(double...numbers) {

if(number.length == 0) {

System.out.println("No argument passed");

return;

}

xxxxx;//当做一个数组一样使用

}

21.java.util.Arrays类包含一些对数组进行操作的静态方法：

.sort(arrayRefer)

.sort(arrayRefer, start, one\_after\_end) 后面两个参数是索引号

.equals(arrayRefer1, arrayRefer2)

.fill(arrayRefer, 5)

.fill(arrayRefer, start, one\_after\_end, fill\_element)

.asList(arrayRefer)

.length 就可以知道长度，是一个public成员变量

int[] getClass()输出是[I

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

ArrayList<Integer> list =new ArrayList<>(Arrays.asList(arrayInt) ); //这会类型不兼容的错误，将上面一句改为Integer。int是基本类型，Integer是整数类

注意上面这种通过ArrayList构建“类对象数组”的写法

打印输出一个ArrayList： [Person@16acdd1, Person@ee6681, Person@18bbc5a]

24.java.util.Date类 推荐joda.DataTime

UML

+Date() //给当前时间创建一个对象

+Data(elapseTime: long) //为指定时间创建一个data对象。

+toString() :String

+getTime() : long

+setTime(elaplseTime: long):void //给对象设置一个新的时间

时间的设置都是1970年1月1号开始的毫秒数。

这个类既有setTime又有getTime，已经可以直接给出数据域了。

使用java.util.Date date =new java.util.Date();

日期： Calendar抽象类

Calendar c = Calendar.getInstance(); //创建子类GregorianCalendar对象

c.add(c.DATE,1);

int day = c.get(Calendar.DAY\_OF\_MONTH);

没想到会遇到使用Calendar之后还继续封装成自己想要的时间类，一般名字叫做DateUtils。org.apache.commons.lang.time.DateUtils这个是很不错的一种实现。

GregorianCalendar是 Calendar 的一个具体子类，提供了标准阳历。Calendar的getInstance( )方法返回用默认的地区和时区的当前日期和当前时间所初始化的GregorianCalendar（标准日历）相当于new GregorianCalendar()；

Calendar c = new GregorianCalendar();相当于使用了Calendar.getInstance() System.out.println(c); 将一个类的数据成员名字和值全部显示出来了。

GregorianCalendar firstFlight = new GregorianCalendar(1903, Calendar.DECEMBER, 17);

GregorianCalendar firstFlight = new GregorianCalendar(1903, 11, 17);

25.UML中静态方法使用下划线。如果是-减号开头就表示是private。

26.java.util.Random类

UML

+Random()

+Random(seed: long)

+nextInt(): int

+nextInt(n: int): int //返回0与n之间（不包含n）的随机数 //n为负数，抛出无效参数异常java.lang.IllegalArgumentException

+nextLong():long

+nextFloat(): float

+nextDouble():double

+nextBoolean(): boolean

Math.random()

26.5 最简单的jfx程序

import javafx.application.Application;

import javafx.stage.Stage;

public class Test extends Application {

public void start(Stage primaryStage) {

primaryStage.show();

}

public static void main(String[] args) {

// TODO Auto-generated method stub

Application.launch(args);

}

}

http://www.java2s.com/Tutorials/Java/JavaFX/0340\_\_JavaFX\_GridPane.htm

BurpSuite 使用java开发

26.5 Pane类型可以add Pane类型上：

下面的程序会画四个风扇

import javafx.application.Application;

import javafx.stage.Stage;

import javafx.scene.Scene;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.StackPane;

import javafx.scene.layout.Pane;

import javafx.scene.shape.Circle;

import javafx.scene.shape.Arc;

import javafx.scene.shape.ArcType;

import javafx.scene.paint.Color;

import javafx.geometry.Insets;

public class Test extends Application {

@Override // Override the start method in the Application class

public void start(Stage primaryStage) {

// Create a GridPane and set its properties

GridPane gridPane = new GridPane();

gridPane.setPadding(new Insets(10, 10, 10, 10));

gridPane.setHgap(10);

gridPane.setVgap(10);

// Place nodes in the pane

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

// Create a stack pane

StackPane stackPane = new StackPane();

// Add circle to stack pane

Circle circle = getCircle();

stackPane.getChildren().add(circle);

// Add Arcs to stack pane

getArcs(stackPane);

gridPane.add(stackPane, i, j);

}

}

// Create a scene and place in in the stage

Scene scene = new Scene(gridPane);

primaryStage.setTitle("Exercise\_14\_09"); // Set the stage title

primaryStage.setScene(scene); // Place the scene in the stage

primaryStage.show(); // Display the stage

}

/\*\* Add four arcs to a pane and place them in a stack pane \*/

private void getArcs(StackPane stackPane) {

double angle = 30; // Start angle

for (int i = 0; i < 4; i++) {

Pane pane = new Pane();

Arc arc = new Arc(100, 100, 80, 80, angle + 90, 35);

arc.setFill(Color.BLACK);

arc.setType(ArcType.ROUND);

pane.getChildren().add(arc);

stackPane.getChildren().add(pane);

angle += 90;

}

}

/\*\* Return a Circle \*/

private Circle getCircle() {

Circle c = new Circle();

c.setRadius(100);

c.setStroke(Color.BLACK);

c.setFill(Color.WHITE);

return c;

}

}

27.javafx的geometry包中有一个point2D类（这名字取得真好）

UML

+point2D(x:double y:double)

+distance(x:double y:double):double

+distance(p:point2D ):double

+getX(): double

+getY(): double

+toString(): String//返回改点的字符串表示

import javafx.geometry.Point2D

30.

package p1;

class C1 { //不具备公共性

}

package p2;

class C2 {

}

31.java安装了jdk，然后就会把jre的安装文件放在安装目录下叫做jre.msi，安装以后就会将jre安装到和安装目录同一个父目录下。不过jre其实不需要安装了，因为jdk里面其实有jre的。

具备开发功能的jdk自己的jre下才会同时有client性质的jvm和server性质的 jvm，而仅仅作为运行环境的jre下只需要client性质的jvm.dll就够了。

32.不可变类是指没有提供更改方法用于对数据域进行更改的类。

33.javac是将.java-》.class, 然后java class 类名就好。

C:\Users\Administrator>javac main.java

main.java:1: 错误: 类TestInitialize是公共的, 应在名为 TestInitialize.java 的文件

中声明

public class TestInitialize {

^

1 个错误

以上错误说明java的源程序文件名是有要求的，就是类名。类名和文件名大小写无所谓？不行，类名要与文件名完全一样。不过windows的识别是无视大小写的，所以可以继续javac 全小写.java。小写的，cmd会处理了以后正常的文件名传给java。

36.将数值类型转换为包装类对象的过程称为装箱（boxing），相反的转换过程称为开箱（unboxing）。编译器会自动装箱和开箱，也就是自动完成类转换。

//import java.lang.Double; .lang是自动包含

Boolean、Character、Double、Float、Byte、Short、Integer和Long这些都是包装类名字，即使名字的全称。

初始化：

可以采用字符串初始化（例如 使用"5"初始化Interger），也可以使用对应的数值类型初始化

转换：类似数值类型的自动转换，返回数值类型

doubleValue()

floatVlaue()

longValue()

intValue()

shortValue()

boolValue()

toString()

compareTo()

静态方法：

初始化

valueOf(str)//这个函数其实没有什么用

valueOf(str, radix) 比如Double.valueOf("1.1111", 2) 。当然这是错误的，进制转换只有在整数下。因为小数可能是不尽的。

valueOf(1) 等等

以下静态方法适用于将字符串转化为对应的数值类型：；

Interger 是parseInt(str) 或者 parseInt(str, radix) 这个返回int

Java里面不使用静态方法没法构建临时变量，因为构造函数前面需要使用new。

37.大数类（不可变imutable，没有这个关键词）：

import java.math.\*;

BigInteger和BigDecimal

可以使用字符串来进行初始化

方法：

add()

subtract()

multiple()

divide()

remainder() //取余

由于没有对BigDemical类对象精度进行限制，所以如果结果不能终止，除法方法就会抛出ArithmeticException异常。所以使用devide()没住需要传递三个参数使用重载方法，第二个参数指定精度第三个参数指定舍入方式。比如所a.divide(b, 20, BigDemical.ROUND\_UP); a和b都是BigDemical对象。

38.java不提供对“内存”的操作，比如C中的memcmp，因为java没有指针类型。memcmp本质上也会转换为某一种类型的指针，比如char\*然后逐个比较，速度快只是因为这种方式转汇编的时候，转出来比较好。

38.5 email正则

原来的正则表达式/^[a-zA-Z0-9\_-]+@[a-zA-Z0-9\_-]+(\.[a-zA-Z0-9\_-]+)+$/; 可以改为

/^(\w)+(\.\w+)\*@(\w)+((\.\w{2,3}){1,3})$/;或者/^(\w)+(\.\w+)\*@(\w)+((\.\w+)+)$/;

\w就是[a-zA-Z0-9\_]，当然在Java里面就是\\w

39.次类 子类 扩展类 派生类

超类 父类 基类

40.java类只能继承一个父类，但是可以实现多个接口。

41.所有的类都继承自java.lang.Object，包括java.lang.Class。基于Object实现的多态是很强的，Object数组，列表。

Object对象默认提供的8种方法：

public final native Class<?> getClass()

public native int hashCode()

public boolean equals(Object obj)

protected native Object clone() throws CloneNotSupportedException

使用protected，一个类没有显示重写这个方法其他类就不能够调用。如果要通知目标自己的clone可以被调用，应该implements Cloneable（作为一个标识接口不提供任何方法）。

clone的时候只是创建的对象地址不同，但是对象里面的属性引用到的还是同一块内存，所以深度克隆是指在clone函数中通过super.clone创建一个对象之后，还通过new等重新创建内部属性。

除了使用clone函数，还可通过序列化复制对象。

public String toString()

以下方法需要在synchronized 同步关键字所限定的作用域中调用，否则会报错java.lang.IllegalMonitorStateException

public final native void notify()

public final native void notifyAll()

public final native void wait(long timeout) throws InterruptedException

public final void wait(long timeout, int nanos) throws InterruptedException

public final void wait() throws InterruptedException

就是使用某一个对象让线程进行阻塞或者被唤醒，wait函数可以设置一个自动唤醒的时间，如果这个时间之前没有被唤醒的话。

protected void finalize() throws Throwable {}

流程：当对象变成(GC Roots)不可达时，GC会判断该对象是否覆盖了finalize方法，若未覆盖，则直接将其回收。否则，若对象未执行过finalize方法，将其放入F-Queue队列，由一低优先级线程执行该队列中对象的finalize方法。执行finalize方法完毕后，GC会再次判断该对象是否可达，若不可达，则进行回收，否则，对象“复活”

子类可以覆盖该方法以实现资源清理工作，GC在回收对象之前调用该方法。C++中的析构函数调用的时机是确定的（对象离开作用域或delete掉），但Java中的finalize的调用具有不确定性。

finalize方法被低优先级线程执行，于是可能导致性能问题。

对象再生问题：finalize方法中，可将待回收对象赋值给GC Roots可达的对象引用，从而达到对象再生的目的

System.gc()与System.runFinalization()方法增加了finalize方法执行的机会，但不可盲目依赖它们

42. 因为使用的是public class声明，所以类名称应该与文件名称完全一致，即应该使用"HelloDemo.java"表示类的名称。

一个源文件中只能有一个public类。

一个源文件可以有多个非public类。

源文件的名称应该和public类的类名保持一致。例如：源文件中public类的类名是Employee，那么源文件应该命名为Employee.java。

43.

public boolean equals(Object o) {

if(o instanceof Myclass)

return num == ((Myclass)o).num;

}

会被检测出来缺少return，毕竟万一条件不满足呢？！

43.

public class Myclass extends Farther {

Myclass(int a) { super(a); }

public boolean equals(Object o) {

if(o instanceof Myclass) //传入farther对象，这里判断是错误。所以不同于形参是Farther类的对象了

return num == ((Myclass)o).num;

else

return true;

}

}

伪代码实现instanceof:

//尝试类型转换，如果不对jvm会抛出异常，捕获异常，然后将result变量作为结果返回

boolean result;

if (obj == null) {

result = false;

} else {

try {

T temp = (T) obj; // checkcast

result = true;

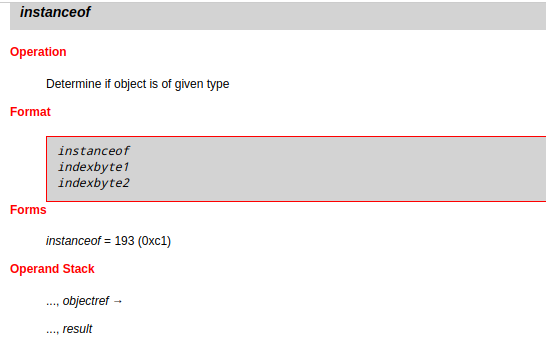
} catch (ClassCastException e) {

result = false;

}

}

JVM有一条名为 instanceof 的指令，而Java源码编译到Class文件时会把Java语言中的 instanceof 运算符映射到JVM的 instanceof 指令上



对indexbyte1和indexbyte2构造的常量池索引进行解析，然后根据java规范判断解析的类是不是objectref的一个实例，最后在栈顶写入结果。

对照C语言中的sizeof，在编译的时候直接变成常量（反汇编可知）。

44.java.util.ArrayList<E>

ArrayList是种泛型类，就是C++中的模板类。

ArrayList<elementType> list =new ArrayList<elementType>(); =>JDK1.7开始 ArrayList<elementType> list =new ArrayList<>();

UML

+ArrayList()

+add(o: E):void

+add(index: int, o: E):void 没想到不是叫做insert

+clear():void

+contain(o: E): boolean

+get(index: int):E

+indexOf(o: Object): int

+isEmpty():boolean

+lastIndexOf(o: object):int

+remove(o:Object):boolean

+remove(index:int):boolean

+set(index:int o:E) :E

可以使用ArrayList<>来当做数据域，来实现堆栈。

45.泛型类

命名类型参数

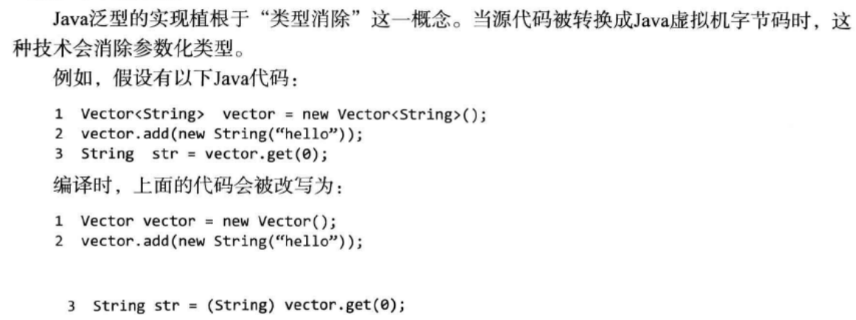
推荐的命名约定是使用大写的单个字母名称作为类型参数。这与 C++ 约定有所不同，并反映了大多数泛型类将具有少量类型参数的假定。对于常见的泛型模式，推荐的名称是：

K —— 键，比如映射的键。   
 V —— 值类型，比如 List 和 Set 的内容，或者 Map 中的值。   
 E —— Element。   
 T —— java类型。 ？ —— 不确定类型，所以代码中用Object。

public interface Map<K, V> {  
 public void put(K key, V value);  
 public V get(K key);  
}

public class Lhist<V> {   
 public Lhist(int capacity) { ... }  
 public int size() { ... }  
 public void add(V value) { ... }  
 public void remove(V value) { ... }  
 public V get(int index) { ... }  
}

Lhist<String> stringList = new Lhist<String>(10);



48.异常

同C++如果异常没有被处理就会导致程序退出。它的使用和C++里面的使用一样，抛出未命名异常然后命名。

数组越界会触发异常：ArrayIndexOutOfBoundsException

如果要求用户输入一个整数但是输入double：InputMismatchException C/C++就会读到点号为止

所有的异常都是java.lang.Throwable的超类。Java里面分为Error和Exception，然后Exception又可以分为RuntimeException和非RuntimeException，C++中Exception分为logic\_error和runtime\_error。这种基类，程序一定会用到，所以他们在java.lang里面，而不是在java.util。

Java库里面的每一个异常类都提供两种初始化：无参数初始化与String参数初始化。

异常处理是很慢的事情：涉及新对象的创建，调用栈的返回，寻找异常处理块catch？

如果异常处理器不能处理一个异常，或者只是简单地希望调用者可以注意到异常，可以对异常进行重新的抛出。

50.文本IO涉及到自动的解码和编码（字符流），而二进制IO不需要（二进制流），所以这是最本质的效率问题。块传输和块处理对两者都是可以适用的，这只是一种提速技术。java 的二进制IO文件与主机无关而与JVM有关。

 InputStreamReader 实现从字节流解码成字符流；

 OutputStreamWriter 实现字符流编码成为字节流

51.

JAR（Java archive）是将一系列文件合并到单个压缩文件里，就象Zip那样，多了一个META-INF/MANIFEST.MF 文件。就是别人已经写好的一些类，然后将这些类进行打包，你可以将这些jar包引入你的项目中，然后就可以直接使用这些jar包中的类和属性以及方法。jar包可能也是个可执行文件。

还可以作为一个可被java运行的应用程序，比如bp

涉及因特网应用时，JAR文件显得特别有用。在JAR文件之前，Web浏览器必须重复多次请求Web服务器，以便下载完构成一个“程序片”（Applet）的所有文件。除此以外，每个文件都是未经压缩的。但在将所有这些文件合并到一个JAR文件里以后，只需向远程服务器发出一次请求即可。同时，由于采用了压缩技术，所以可在更短的时间里获得全部数据。

52.为了确保Java技术不会被邪恶目的所利用，SUN公司在设计Java的时候，设计了一套精密的安全模型；即安全管理器（Security Manager）将检查有权使用的所有系统资源。在默认的情况下，只允许那些无害的操作，要想允许执行其他操作，代码需得到数字签名，用户必须得到数字认证。

54.java生成可执行文件：

生成jar：（只需要安装jdk即可）

在同一目录下生成.java对应的.class。然后jar -cvf xxx.jar 目录名字

上述命令就会将目录打包成一个jar

要使用的话：set classpath=.;d:\javapro\my.jar，估计这是设置临时路径

之后可以直接java 类名执行了，不需要再生成.class文件。

生成exe：

对.jar使用exe4j这个工具进行转化为.exe

55.编译的时候：

注: 某些消息已经过简化; 请使用 -Xdiags:verbose 重新编译以获得完整输出

56.

JAR

WAR

EAR

RAR （源文件打包）

英文

Java Archive file

Web Archive file

Enterprise Archive file

包含内容

class、properties文件，是文件封装的最小单元；包含Java类的普通库、资源（resources）、辅助文件（auxiliary files）等

Servlet、JSP页面、JSP标记库、JAR库文件、HTML/XML文档和其他公用资源文件，如图片、音频文件等

除了包含JAR、WAR以外，还包括EJB组件

部署文件

application-client.xml

web.xml

application.xml

容器

应用服务器（application servers）

小型服务程序容器（servlet containers）

EJB容器（EJB containers）

级别

小（库）

中（项目）

大

57.String是非基础变量类型，所以==会出现。这时候是一种引用，存的是地址，但是用的时候就想直接对对象的使用。

Integer a = 1000, b = 1000;

System.out.println(a == b);//false，说明指向的不是同一个内存对象

Integer c = 100, d = 100;

System.out.println(c == d);//true

如果你看去看 Integer.java 类，你会发现有一个内部私有类，IntegerCache.java，它缓存了从-128到127之间的所有的整数对象。如果值的范围在-128到127之间，它就从高速缓存返回实例。

String内部使用一个final char数组实现

不可变对象默认是线程安全的，可以通过pool的思想实现优化：

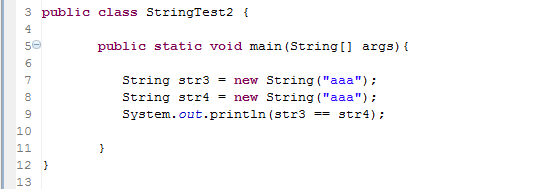
有两种创建字符串对象的方式：1）采用字面值的方式赋值  2）采用new关键字新建一个字符串对象。

字符串池在String类中，且不会被GC回收机制回收，在 Java 7 之前，字符串常量池被放在运行时常量池中，它属于永久代。而在 Java 7，字符串常量池被移到 Native Method 中。这是因为永久代的空间有限，在大量使用字符串的场景下会导致 OutOfMemoryError 错误。

调用String对象的intern()，如果池中已经有字符串了就会返回池中的字符串，不过这时候自己也已经是创建了一个变量了；关键是如果这时候池中没有这个字符串，就会加入字符串的池。

采用字面值的方式创建一个字符串时，JVM首先会去字符串池中查找是否存在"aaa"这个对象，如果不存在，则在字符串池中创建"aaa"这个对象，然后将池中"aaa"这个对象的引用地址返回给字符串常量str，这样str会指向池中"aaa"这个字符串对象；如果存在，则不创建任何对象，直接将池中"aaa"这个对象的地址返回，赋给字符串常量。

采用new关键字新建一个字符串对象时，一定会创建一个单独的变量，而不是使用缓存池。



如果要判断字符串的内容是否相同应该使用equals(Object)

StringBuffer功能上与StringBuilder一样，但是内部使用 synchronized ，即是线程安全的。

运行速度上：StringBuilder > StringBuffer（线程安全） > String

58.class not found这错误遇到的次数太多了...记得的原因一些，比如编译器版本不对，导致源文件没有被编译，还有把自己的class名成了和系统某个class一样的名，还有maven下载包的损坏，损坏的原因可能只是断网了一下，即使能手动找到包也没用，只能重新删除库重来

62.Change your project JSDK to 64-bit version, you are building and running with 32-bit JDK right now which cannot allocate more than 2GB RAM on Windows。应用程序分配内存也不是真的就会占用这么大，除非是vmware这种。

63.java.net

String mytext = java.net.URLEncoder.encode("中国", "utf-8");

String mytext2 = java.net.URLDecoder.decode(mytext, "utf-8");

64.Lamdba表达式：

Java8的特性，但是倒着支持到android2.3。凡是只有一个待实现方法的接口都可以直接使用lamdba表达式。

本质只是一个"语法糖",由编译器推断并帮你转换包装为常规的代码

基本语法:

(parameters) -> expression

或

(parameters) ->{ statements; }

// 1. 不需要参数,返回值为 5

() -> 5

// 2. 接收一个参数(数字类型),返回其2倍的值

x -> 2 \* x

// 3. 接受2个参数(数字),并返回他们的差值

(x, y) -> x – y

// 4. 接收2个int型整数,返回他们的和

(int x, int y) -> x + y

// 5. 接受一个 string 对象,并在控制台打印,不返回任何值(看起来像是返回void)

(String s) -> System.out.print(s)

//配合java8中的forEach使用，或者配合其它函数进行使用

String[] atp = {"Rafael Nadal", "Novak Djokovic",

"Stanislas Wawrinka",

"David Ferrer","Roger Federer",

"Andy Murray","Tomas Berdych",

"Juan Martin Del Potro"};

List<String> players = Arrays.asList(atp);

// 以前的循环方式

for (String player : players) {

System.out.print(player + "; ");

}

// 使用 lambda 表达式以及函数操作(functional operation)

players.forEach((player) -> System.out.print(player + "; ")); //forEach方法会将每一个元素传参到lambda表达式

players.forEach(System.out.println);

类似的函数还有sort、find、filter等

phpProgrammers.stream()

.filter((p) -> (p.getSalary() > 1400))

.forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()));

javaProgrammers.stream()

.filter(genderFilter)

.limit(3) //限制了结果集的大小为3

.forEach((p) -> System.out.printf("%s %s; ", p.getFirstName(), p.getLastName()));

//代替内部匿名类

// 使用匿名内部类

btn.setOnAction(new EventHandler<ActionEvent>() {

@Override

public void handle(ActionEvent event) {

System.out.println("Hello World!");

}

});

// 或者使用 lambda expression

btn.setOnAction(event -> System.out.println("Hello World!"));

//比如实现Runnable接口

// 1.1使用匿名内部类

new Thread(new Runnable() {

@Override

public void run() {

System.out.println("Hello world !");

}

}).start();

// 1.2使用 lambda expression

new Thread(() -> System.out.println("Hello world !")).start();

使用数组或者集合来改变外部变量的值

65.使用迭代器或 for-each 循环是遍历 ArrayList 最有效的方式，性能比采用索引值的传统 for 循环方式好两倍。

66.在java8中并行流的速度是最快的，其次是lambda，然后就是流的速度。

我们先创建了一个 ArrayList，并插入一个 100000 个随机整数，并通过5 种不同的方式遍历所有的值来查找最大值

让我们快速看一下每个方法，按照运行速度由快到慢（尤其是foreach和迭代器）：

命令式风格

iteratorMaxInteger()——使用迭代器遍历列表：

public int iteratorMaxInteger() {

int max = Integer.MIN\_VALUE;

for (Iterator it = integers.iterator(); it.hasNext(); ) {

max = Integer.max(max, it.next());

}

return max;

}

forEachLoopMaxInteger()——不使用迭代器，使用 For-Each 循环遍历列表（不要误用 Java 8 的 forEach）

public int forEachLoopMaxInteger() {

int max = Integer.MIN\_VALUE;

for (Integer n : integers) {

max = Integer.max(max, n);

}

return max;

}

forMaxInteger()——使用简单的 for 循环和索引遍历列表：

public int forMaxInteger() {

int max = Integer.MIN\_VALUE;

for (int i = 0; i < size; i++) {

max = Integer.max(max, integers.get(i));

}

return max;

}

然后就是优雅但是很慢的函数式风格

parallelStreamMaxInteger()——使用 Java 8 并行流遍历列表：

public int parallelStreamMaxInteger() {

Optional max = integers.parallelStream().reduce(Integer::max);

return max.get();

}

lambdaMaxInteger()——使用 lambda 表达式及流遍历列表。优雅的一行代码：

public int lambdaMaxInteger() {

return integers.stream().reduce(Integer.MIN\_VALUE, (a, b) -> Integer.max(a, b));

}

68.Java8中的Stream特性

Stream 不是集合元素，它不是数据结构并不保存数据，它是有关算法和计算的，它更像一个高级版本的 Iterator。原始版本的 Iterator，用户只能显式地一个一个遍历元素并对其执行某些操作；高级版本的 Stream，用户只要给出需要对其包含的元素执行什么操作，比如 “过滤掉长度大于 10 的字符串”、“获取每个字符串的首字母”等，Stream 会隐式地在内部进行遍历，做出相应的数据转换。Stream 就如同一个迭代器（Iterator），单向，不可往复，数据只能遍历一次，遍历过一次后即用尽了。

集合类持有的所有元素都是存储在内存中的，非常巨大的集合类会占用大量的内存，而Stream的元素却是在访问的时候才被计算出来，这种“延迟计算”的特性有点类似Clojure的lazy-seq，占用内存很少。所以stream可以表示无限的集合，比如说自然数。

sortedJavaProgrammers = javaProgrammers

.stream()

.sorted( (p, p2) -> (p.getSalary() - p2.getSalary()) )

.collect( toList() );

平方数

List<Integer> nums = Arrays.asList(1, 2, 3, 4);

List<Integer> squareNums = nums.stream().

map(n -> n \* n).

collect(Collectors.toList());

如果是偶数就会平方输出

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

Stream<Integer> stream = numbers.stream();

stream.filter((x) -> {

return x % 2 == 0;

}).map((x) -> {

return x \* x;

}).forEach(System.out::println);

}

自然数集合的规则非常简单，每个元素都是前一个元素的值+1，因此，自然数发生器用代码实现如下：

class NaturalSupplier implements Supplier<Long> {

long value = 0;

public Long get() {

this.value = this.value + 1;

return this.value;

}

}

Stream<Long> natural = Stream.generate(new NaturalSupplier()); //反复调用get()，将得到一个无穷数列，利用这个Supplier，可以创建一个无穷的代表自然数的Stream

natural.limit(10).forEach(System.out::println);

Java8中增加了流(stream)的概念，为数据的处理带来了很大的方便。并行流将数据分割成不同的块，并且用不同的线程处理不同的块。

现在要根据输入`n`，求从1到n的和，这里我们不使用`n\*(1+n)/2`的方式（可能溢出），而是使用累加。我们分别写出迭代、顺序流、并行流的写法。

public class ParallelStreams {

public static long sequentialSum(long n) {

return Stream.iterate(1L, i->i+1)

.limit(n)

.reduce(0L, Long::sum);

}

public static long iterativeSum(long n) {

long result = 0;

for (long i = 1L; i <= n; i++) {

result += i;

}

return result;

}

public static long parallelSum(long n) {

return Stream.iterate(1L, i->i+1)

.limit(n)

.parallel()//我们可以看到，将顺序流转换为并行流只需要`parallel()`方法即可。

.reduce(0L, Long::sum);

}

}

耗时如下：

顺序流:173 毫秒

循环:4 毫秒

并行流:697 毫秒

为什么会这样？

`iterate`生成的是`Stream<Long>`对象，需要拆箱才能求和；

`iterate`很难分割成独立的小块，因为每次应用这个函数都需要前一次应用的结果，也就是说它其实是顺序执行的。这样反而在并行时增加了分配线程的开销。

这里我们使用`LongStream.rangeClosed`方法来优化，它的特点有：

`LongStream.rangeClosed`直接产生原始类型的`long`数字，没有拆箱与装箱的开销。

`LongStream.rangeClosed`产生一个数字范围，很容易拆分成多个小块。

我们再用`LongStream.rangeClosed`写出顺序和并行计算的版本：

public static long parallelRangedSum(long n) {

return LongStream.rangeClosed(1, n)

.parallel()

.reduce(0, Long::sum);

}

使用了LongStream以后耗时如下：

顺序Range流:7 毫秒

并行Range流:3 毫秒

69.类就是对接口的实现，java因为多重继承太复杂了就只允许进行单一继承，只能有一个父类，不过可以有多个继承接口。

70.org.apache.http.HttpResponse这种包，那么在环境变量中就要设置能找到org文件夹的路径。

72.idea可以直接打开eclipse的项目，我打开的时候（打开项目中的.propject文件）遇到了项目使用的是Java SE1.6，然后就提醒我没有。这不就是我已经有了的jdk1.8的旧版么。idea可以选择一个文件来进行run，或者选择项目来进行run。我一开始打算直接用javac和java来解决问题，但是这个项目我虽然引入了需要的基本类文件，却不能将依赖的jar引入进去（其实需要指定jar包classPath）。无奈之下只好使用IDE来直接打开了。

73.产生4位随机验证码：

public static void main(String[] args) {

String codeTable ="1234567890abcdefghijkmnpqrstuvwxyz";

String messageCode ="";

Random random =new Random();

for(int i1=0; i1 < 4; ++i1)

messageCode +=codeTable.charAt( random.nextInt(codeTable.length()) );

System.out.println(messageCode);

}

reverse word <=>keyword

Unix epoch时间戳

pre increment decrement

post

vild card 通配符

escape 转义

nested 嵌套的

imutable 不可变的

74.反射 ：

主要是指程序可以访问，检测和修改它本身状态或行为的一种能力，并能根据自身行为的状态和结果，调整或修改应用所描述行为的状态和相关的语义。

其作用是使用一定性能来换取灵活，但是性能牺牲还是很大的，后来几种框架ButterKnife、Dagger2通过apt代码自动生成技术，其注解是停留在编译时处理，完全不影响性能

有以下几个作用：

用于写一个配置文件，通过配置文件中的类名，通过对配置文件中的类名的修改，然后在某一个函数中使用配置文件的行数中的字符串来实例化

一个对象

使用别人的库的时候，有时候会不满意功能，想要访问一个私有变量的时候只能使用反射

通过反射还能够修改final关键字修饰的变量，只要不会被编译器内联优化的 final 属性就可以通过反射有效的进行修改

class Person {

public final String name = "Mike"; //简单的字符串类型会被直接优化，直接认为是一个常量，所有用到的地方都被直接替换为字符串常量

public final String name2 = new String("Mike"); //

public final String city; //可被有效修改

}

java.lang.reflect.Constructor;

java.lang.reflect.Field;

java.lang.reflect.Method;

java.lang.reflect.Modifier;

获取类：

//第一种方式：

java.lang.Class;

Classc1 = Class.forName("Employee");

//第二种方式：

//java中每个类型都有class 属性.

Classc2 = Employee.class;

//第三种方式：

//java语言中任何一个java对象都有getClass 方法

Employee e = new Employee();

Classc3 = e.getClass(); //c3是运行时类 (e的运行时类是Employee)

创建对象：

Object =Classc1.newInstance(); 就会调用无参数构造函数

获取类的属性：

//获取整个类 ，像定义一样输出类

Class c = Class.forName("java.lang.Integer");

//获取所有的属性

Field[] fs = c.getDeclaredFields();

//定义可变长的字符串，用来存储属性

StringBuffer sb = new StringBuffer();

//通过追加的方法，将每个属性拼接到此字符串中

//最外边的public定义

sb.append(Modifier.toString(c.getModifiers()) + " class " + c.getSimpleName() +"{\n");

//里边的每一个属性

for(Field field:fs){

sb.append("\t");//空格

sb.append(Modifier.toString(field.getModifiers())+" ");//获得属性的修饰符，例如public，static等等

sb.append(field.getType().getSimpleName() + " ");//属性的类型的名字

sb.append(field.getName()+";\n");//属性的名字+回车

}

sb.append("}");

System.out.println(sb);

//获取指定属性

Field idF = c.getDeclaredField("id");

idF.setAccessible(true); //使用反射机制打破了封装性

Object o = c.newInstance();

idF.set(o, "110");

通过类名来实例化一个对象： object\_factory\_map内容来自于配置文件

Any GetInstanceByName(const string& name) {

return Class.forName(name).newInstance(); //使用newInstance函数来实例化一个对象只能没有参数

}//Any 是java idl 里面的一个类，继承自Object实现了IDLEntity

不带参数：

A a=(A)Class.forName("reflect.A").newInstance();

带参数：

Class c=Class.forName("A");

Constructor c1=c.getDeclaredConstructor(new Class[]{int.class,int.class});

A a1=(A)c1.newInstance(new Object[]{5,6});

通过反射机制能够修改一个对象的私有成员

import java.lang.reflect.Field;

public class Demo1 {

public static void main(String[] args) throws ClassNotFoundException, IllegalArgumentException, Exception {

Class c=Class.forName("com.jr.reflect.Dep");

Object c1=c.newInstance();//创建对象后应赋值给另一个对象

Field [] fields=c.getDeclaredFields();

fields[0].setAccessible(true); //修改访问限制

System.out.println(fields[0].getName());

fields[0].set(c1, 1);//第一个参数为:被创建的对象，二：要修改的值

System.out.println(fields[0].get(c1));

}

}

通过修改访问限制来实现python中“删除”成员的效果

75.下面的描述中使用POJO不是很好，因为POJO不包含JavaBean，实际上应该是“Java类”

Annotation Processing （后面简称 AP ）发生在编译时，可以利用 AP 在编译时生成一些代码来解决一些通用的问题，从某种程度上讲，可以替代一些完全依赖反射实现的库从而取得更好的性能。annotation processing tools(APT)，这也是Android ButterKnife依赖的东西。我们在运行时用反射的场景一般是为了通用，比如一些解析 JSON 的库，作者在写库的时候不可能知道你所使用的 JSON 格式是什么样子的，不知道你的 POJO 是什么样子，所以用反射在运行时获取 POJO 有哪些字段，从而动态的完成 JSON 与 POJO 的映射。假如我们在运行之前就知道了 JSON 的格式，POJO 有哪些字段，知道了 JSON 与 POJO 的映射关系，那我们就可以不用反射。我们在写完代码，编译运行的时候，我们的代码就固定了，有哪些 POJO 就固定了，这时候我们可以不用三方的 JSON 解析库，我们自己手写代码完成 JSON 与 POJO 转换的代码，这样避免了运行时反射，性能更好，当然这样就太麻烦了，所以才会有这些三方库的存在。有没有不用自己写又没有运行时反射的方法呢？那就可以利用 AP， AP 可以在编译的时候获取所有 POJO 对象，然后通过反射（这里的反射发生在编译时，不在运行时）获取 POJO 的字段，生成 JSON 与 POJO 映射的相关代码，最后这些代码会和你写的代码一起编译打包运行。相当于你自己手写的转换代码现在是 AP 帮你写了，在这种场景下，AP 生成代码的方式肯定是优于完全基于运行时反射的 JSON 解析库，因为就和你自己手写的代码一样。

这里有一个小问题，那就是 AP 生成的代码怎么引用到，像 butterknife、 dagger 在运行时也是有反射的，因为他们要反射去寻找 AP 生成的那些 class，但是他们的注入过程都不是反射，也就是大部分不是反射，只有少量反射，性能优于完全基于反射的库。而 dagger2 去掉了所有的运行时反射，因为它不需要运行时去寻找 AP 生成的 class，你要在代码中直接引用生成的 class，这时候就能感觉到是 AP 再帮你写代码，因为你引用了一个不是自己写的 class。

最后，反射在很多场景下都会使用，反射具有动态性，解决一些通用问题时都离不开反射，而用不用 AP 是另外一回事，就算用 AP 之后在运行时没有反射，但是在编译时也会有反射。

75. JavaBean and JavaBeans

JavaBeans 从狭义来说，指的是 JavaBeans 规范也就是位于 java.beans 包中的一组 API。从广义上来说，JavaBeans 指的是 API 集合，比如 Enterprise JavaBeans。

JavaBean 指的是按照某种规范实现的Java类。

76.POJO是Plain Ordinary Java Object的缩写不错，但是它通指没有使用Entity Beans的普通java对象，可以把POJO作为支持业务逻辑的协助类。POJO实际上指的即普通Java类，不具有任何特殊角色和不继承或不实现任何其它Java框架的类或接口。

77.邮件的处理有Java Mail API

厂商所提供的Java Mail服务程序可以有选择地实现某些邮件协议，常见的邮件协议包括：

（1）SMTP：简单邮件传输协议，用于发送电子邮件的传输协议；

（2）POP3：用于接收电子邮件的标准协议；

（3）IMAP：互联网消息协议，是POP3的替代协议。

这三种协议都有对应SSL加密传输的协议，分别是SMTPS，POP3S和IMAPS。除Java Mail服务提供程序之外，Java Mail还需要JAF(JavaBeans Activation Framework)来处理不是纯文本的邮件内容，这包括MIME（多用途互联网邮件扩展）、URL页面和文件附件等内容。对附件的处理和HTTP发送文件一样：如果一封邮件中含有附件，那邮件的Content-Type域中必须定义multipart/mixed类型，邮件通过multipart/mixed类型中定义的boundary标识将附件内容同邮件其它内容分成不同的段。基本格式：Content-Type： multipart/mixed;boundary="{分段标识}"

mail. jar：此JAR文件包含Java Mail API和Sun提供的SMTP、IMAP和POP3服务提供程序；

activation. jar：此JAR文件包含JAF API和Sun的实现。

Java Mail包含两部分内容，一部分是Java Mail API，定义了一组平台无关、独立于通讯协议的邮件程序框架，该部分称为应用级接口，也就是供我们调用的部分，另一部分是service provider，该部分使用特定的协议来实现第一部分定义的抽象类和接口，这些协议包括：SMTP、NNTP、POP3、IMAP，如果让Java Mail与邮件服务器通信，就需要相应的协议支持，该部分称为服务提供者接口，也就是Java Mail自身需要的协议支持。在使用Java Mail时，通常我们只需将mail. jar放在class path下使用，它包含了Java Mail API部分和SUN自己实现的service provider部分。可能也有特殊的时候，我们应用程序中需要自己实现service provider部分，那我们只需要mailapi. jar。

79.生成javadoc文件使用的注释

@link字符是可以单独使用如：@link sth 或者@link sth1#sth2

/\*\*注释用到的参数

\*@param context

\*@param attrs

\*/

Public DotView(Context context, Attribute attrs) {

}

/\*\* 提供用户相关的类或函数信息

\* @see MyBase#myFunc()

\*/

79.实际上，Java中存在四种引用，它们由强到弱依次是：强引用、软引用、弱引用、虚引用。下面我们简单介绍下除弱引用外的其他三种引用：

强引用（Strong Reference）：通常我们通过new来创建一个新对象时返回的引用就是一个强引用，若一个对象通过一系列强引用可到达，它就是强可达的(strongly reachable)，那么它就不被回收

软引用（Soft Reference）：软引用和弱引用的区别在于，若一个对象是弱引用可达，无论当前内存是否充足它都会被回收，而软引用可达的对象在内存不充足时才会被回收，因此软引用要比弱引用“强”一些

弱引用对象的存在不会阻止它所指向的对象被垃圾回收器回收。弱引用最常见的用途是实现规范映射(canonicalizing mappings，比如哈希表）。弱可达的(weakly reachable)（也就是说当前指向它的全都是弱引用）。WeakReference类

虚引用（Phantom Reference）：虚引用是Java中最弱的引用，那么它弱到什么程度呢？它是如此脆弱以至于我们通过虚引用甚至无法获取到被引用的对象，虚引用存在的唯一作用就是当它指向的对象被回收后，虚引用本身会被加入到引用队列中，用作记录它指向的对象已被回收。

80.调用exe并获取其标准输出作为返回值

最好不要使用Process pro =Runtime.getRuntime().exec("xxxx.exe") 而是使用一个回调，exe每输出一行字符就进行一次回调。不过实时显示最好的办法是类似于linux中popen函数。

84.类的编写： 通过设置interface实现CallBack

public class CalendarItem extends Fragment implements NoteCalendar2.ChangeDateListener{

CallBack mCallBack;

public void setCalendarCallBack(CallBack cb){

mCallBack = cb;

}

public interface CallBack{

abstract void onChangeSelectedDate(int year,int month ,int day);

}

}在其他方法中通过mCallBack.onChangeSelectedDate(year, month, day); 调用。

85.

void关键字表示函数没有返回结果，是java中的一个关键字。

java.lang.Void是一种类型（也就有Void.TYPE），Void作为函数返回值表示函数返回null。

86.String类型对象加上一个boolean表达式，从而转为一个大String，需要使用()将boolean表达式括起来。

87.

// 错误: 无法从静态上下文中引用非静态 变量 this

public class Test {

public static void main(String args[]) {

Test test=new Test();

Test2 test2=new Test2();

test2.print();

}

public int function() {

int a=1,b=4;

int c=5;

return (a+b)\*c;

}

public class Test2 {

private int a=function();

public void print() {

System.out.println(a);

}

}

}

解决办法：使用静态类，或者将这个类从这里提取出来

88.Java中闭包的实现是不完整的：

Java8编译器实现的只是capture-by-value，暂时没有实现capture-by-reference.但Java又不肯明说，只能粗暴地一刀切，就说既然内外不能同步，那就不许大家改外围的局部变量。这也就导致了,匿名类里面使用外部的变量的话需要使用final修饰.

同时匿名类已经不是自己类中了，所以不能在其中new自己类中定义的private对象。

但是Java只是不允许改变被lambda表达式捕获的变量，并没有限制这些变量所指向的对象的状态能不能变。要从lambda表达式向外传值的常见workaround之一就是用长度为1的数组：

String[] a = new String[1];

... ( () -> a[0] = "a" );

return a[0];

90.CodeCache

关于启动后负载高的原因，网上很多文章都说是由于启动后随着代码的执行，jvm的jit编译器将部分热点代码编译为目标机器代码，由于编译线程占用了大量的cpu导致系统负载高。jit编译器需要代码执行超过一定频率才会将其编译，系统刚启动的时候大部分代码都是出于解释执行阶段

Jvm提供了一个参数-Xcomp，可以使jvm运行在纯编译模式下，所有方法在第一次被调用的时候就会被编译成机器代码。如果因为解释导致代码的效率一直不是很好，可以开启这个从而提升效率。

编译后的代码被缓存起来就是CodeCache，如果这个缓冲区满了就满了，没有清理机制。很自然，会导致不能后面的代码摆脱解释执行的命运。

92.没有1E，E后面必须要加上数字：

int i1=2;

System.out.println(""+1E2+i1);

93.文件流操作：Java中的基本输入输出流 以及 文件相关输入输出流

基本上函数的参数都是byte[]，而不是char[]

需要处理的异常：IOException（java.io.IOException）

try{

System.out.println(test.getInputMessage());

}catch(IOException ex) {

ex.printStackTrace();

}

都有close函数、flush()函数，有点似乎flush函数需要传如一个true值

1.通过int count=System.in.read(buffer) 去读到缓冲区，读取到回车为止

byte buffer[]=new byte[1024];

int count=System.in.read(buffer);

缺点：下面的函数有一个不支持多字符文字的bug，如果直接将return new String(buffer);就能够显示出来，但是这时候后面会跟着一堆不可见字符。

public String getInputMessage() throws IOException{

System.out.println("请输入您的命令∶");

byte buffer[]=new byte[1024];

int count=System.in.read(buffer);

char[] ch=new char[count-2];//最后两位为结束符\r\n，删去不要

for(int i=0;i<count-2;i++)

ch[i]=(char)buffer[i];

String str=new String(ch);

return str;

}

通过上面的函数可以知道通过char转为String的时候,每一个字节都作为单独的字符显示。但是将byte[] 转换为String的时候能够正常进行多字符编码的解析。最终修复的函数如下：

public String getInputMessage() throws IOException{

System.out.println("请输入您的命令∶");

byte buffer[]=new byte[1024];

int count=System.in.read(buffer);

byte[] ch=new byte[count-2]; //最后两位为结束符，删去不要

for(int i=0;i<count-2;i++)

ch[i]=buffer[i]; // 不能直接转，必须选择有数据的部分，不然会带上乱码

String str=new String(ch);

return str;

}

byte[] 、 char[] 以及 String 之间的区别与转换：

java中的char是Unicode字符，也就是一个字符会占有两个字节的整数位置这样。但是byte这种类型的存储单位就是一个字节。两者之间本质还是一个编码与解码的关系。所以转换函数就是通过编码以及解码。

private byte[] getBytes (char[] chars) {

Charset cs = Charset.forName ("UTF-8");

CharBuffer cb = CharBuffer.allocate (chars.length); //缓冲区分配大小

cb.put (chars);

cb.flip ();

ByteBuffer bb = cs.encode (cb); //java.net.URLEncoder.encode

return bb.array();

}

private char[] getChars (byte[] bytes) {

Charset cs = Charset.forName ("UTF-8");

ByteBuffer bb = ByteBuffer.allocate (bytes.length);

bb.put (bytes);

bb.flip ();

CharBuffer cb = cs.decode (bb);

return cb.array();

}

或者使用类InputStreamReader OutputStreamWriter

InputStreamReader 将字节流转换为字符流。是字节流通向字符流的桥梁。 如果不指定字符集编码，该解码过程将使用平台默认的字符编码。

int read();//读取单个字符。

int read(char []cbuf);//将读取到的字符存到数组中。返回读取的字符数。

OutputStreamWriter 将字符流转换为字节流，如果不指定解码放啊发就会说使用操作系统默认的解码

void write(int c);//将单个字符写入。

viod write(String str,int off,int len);//将字符串某部分写入。

void flush();//将该流中的缓冲数据刷到目的地中去。

//String 与它们之间的转换

String str ="123";

byte[] bt=str.getBytes();

2.FileInputStream 、FileOutputStream 支持操作何形式的文件，FileDescriptor 文件描述符指定的流，如果是操作文本文件据说FileInputReader和FileOutputWriter效率更高

学习网址：http://blog.csdn.net/caixiexin/article/details/6719407 关注read以及write函数。读入支持skip函数

很明显是从　　java.io.InputStream、java.io.OutputStream继承过来，所以也会自带缓冲区，因为也有flush函数

缺点：不支持对文件位置的偏移，类似乎没有提供对文件指针的操作

public void copyFile(String src,String dest) throws IOException{

FileInputStream in=new FileInputStream(src);//构造函数接受文件路径、File对象

File file=new File(dest);

if(!file.exists())

file.createNewFile();

FileOutputStream out=new FileOutputStream(file);

int c;

byte buffer[]=new byte[1024];

while((c=in.read(buffer))!=-1)...{

for(int i=0;i<c;i++)

out.write(buffer[i]);

}

in.close();

out.close();

}

3.FileChannel类 Java NIO FileChannel是连接文件的通道。Java NIO FileChannel类是NIO用于替代使用标准Java IO API读取文件的方法，在速度上较传统的文件读写API快。使用FileChannel，您可以从文件中读取数据和将数据写入文件，以及操作（比如说对文件的裁剪）。缺点就是FileChannel无法设置为非阻塞模式。它总是以阻塞模式运行。

可以通过FileInputStream 、FileOutputStream对象中的getChannel()返回一个FileChannel对象。这个类的对象需要通过InputStream，OutputStream或RandomAccessFile来获取。同样是最后要close的。如果要说的话，获取与linux中的管道有点接近。

try {

fileInputStream = new FileInputStream(fromFile);

fileOutputStream = new FileOutputStream(toFile);

//得到fileInputStream的文件通道

fileChannelInput = fileInputStream.getChannel();

//得到fileOutputStream的文件通道

fileChannelOutput = fileOutputStream.getChannel();

//将fileChannelInput通道的数据，写入到fileChannelOutput通道

fileChannelInput.transferTo(0, fileChannelInput.size(), fileChannelOutput);

// 否则应该是会抛出异常

} catch (IOException e) {

e.printStackTrace();

} finally {

try {

if (fileInputStream != null)

fileInputStream.close();

if (fileChannelInput != null)

fileChannelInput.close();

if (fileOutputStream != null)

fileOutputStream.close();

if (fileChannelOutput != null)

fileChannelOutput.close();

} catch (IOException e) {

e.printStackTrace();

}

}

返回通道连接的文件的大小： 不如从文件属性中获取文件大小来得快

channel.size（）;

读写数据：通过read以及write函数

ByteBuffer buf = ByteBuffer.allocate（48）;

FileInputStream is = new FileInputStream(in);

FileOutputStream os = new FileOutputStream(out);

FileChannel fis = is.getChannel();

FileChannel fos = os.getChannel();

while(fis.read(bytedata)!= -1){

bytedata.flip();

fos.write(bytedata);

bytedata.clear();

}

截断文件：

channel.truncate（1024）;//从当前位置开始的1024字节截断文件，相当于删除之后的内容

设置文件指针位置： 从而进行特定位置上的读写

long pos channel.position（）;

channel.position（pos +123）;

强制写入磁盘：

channel.force(true) 将通道里尚未写入磁盘的数据强制写到磁盘上。出于性能方面的考虑，操作系统会将数据缓存在内存中。操作系统的行为会影响部分函数API的具体执行效果。尤其是这种调用操作系统API并且对性能有不小影响的。

4.FileReader FileWriter 对文本文件的读写

FileReader：

通过这种方式读到的是\r\n回车，如果仅仅消除了\n，那么\r的效果就会有点爽

int read(); // 读取单个字符。返回作为整数读取的字符，如果已达到流末尾，则返回 -1。

int read(char []cbuf);//将字符读入数组。返回读取的字符数。

void close();//关闭此流对象。释放与之关联的所有资源。

FileWriter:

void write(String str) //写入字符串。当执行完此方法后，字符数据还并没有写入到目的文件中去。此时字符数据会保存在缓冲区中。

viod flush() //刷新该流中的缓冲。尝试将缓冲区中的字符数据保存到目的文件中去。

viod close() //关闭此流。在关闭前会先刷新此流的缓冲区。在关闭后，再写入或者刷新的话，会抛IOException异常。

write函数的参数类型与read函数的参数类型 不是一样的诶

5.Scanner PrintWriter Java书本上的标配

优点是能够实现格式化的输入以及输出。

Scanner：

Scanner s = new Scanner(System.in);

PrintWriter：

学习网址：http://www.cnblogs.com/xiaotiaosi/p/6394147.html

通过想要输出的流目标来初始化一个这个类的对象。

比较奇特的事情是提供了一个append函数允许直接向其缓冲区中提供字符序列（比如说字符串）

6.BufferedReader BufferedWriter

int read();//读取单个字符。

int read(char[] cbuf,int off,int len);//将字符读入到数组的某一部分。返回读取的字符数。达到尾部 ，返回-1。

String readLine(); //读取一个文本行。

void close(); //关闭该流。并释放与该流相关的所有资源。

BufferedReader inFromClient = new BufferedReader(new InputStreamReader(socket.getInputStream())); 调用了close方法以后socket也会被关闭

使用了上面这行命令以后就会通过getInputStream读取到自己设定的缓冲区的时候就就会出现一些错误的字符，建议使用DataOutpuStream，这种流支持的接口比较多。

BufferedWriter：

void write(char ch);//写入单个字符。

void write(char []cbuf,int off,int len)//写入字符数据的某一部分。

void write(String s,int off,int len)//写入字符串的某一部分。

void newLine()//写入一个行分隔符。

void flush();//刷新该流中的缓冲。将缓冲数据写到目的文件中去。

void close();//关闭此流，再关闭前会先刷新他。

1. JDK8 提供的特性： 两个readAllxxx函数

最开头介绍的getResourceStream也是获取所有内容

System.out.println(new String(Files.readAllBytes(Paths.get("D:\\jd.txt"))));

List<String> lines = Files.readAllLines(Paths.get("D:\\jd.txt"), StandardCharsets.UTF\_8);

import java.nio.file.Files;

import java.io.IOException;

import java.nio.file.Paths;

public class Test {

public static void main(String[] args) {

try {

//特定将两种斜杠都试了一下

System.out.println(new String(Files.readAllBytes(Paths.get("C:/Users\\tmp\\jd.txt"))));

}catch(IOException e) {

e.printStackTrace();

}

}

}

8.读取特殊文件，比如通过dom4j读取xml文件

装饰流：DataInputStream DataOutputStream Scanner PrintStream 建立在其他流之上，提供额外的功能

DataInputStream DataOutputStream

DataOutputStream数据输出流 将java基本数据类型写入数据输出流中。并可以通过数据输入流DataInputStream将数据读入

DataInputStream:

int read(byte[] b);//从输入流中读取一定的字节，存放到缓冲数组b中。返回缓冲区中的总字节数。

int read(byte[] buf,int off,int len);//从输入流中一次读入len个字节存放在字节数组中的偏移off个字节及后面位置。

使用DataInputStream来读取字节流中的数据：

ByteArrayInputStream bin=new ByteArrayInputStream(byte[] buf);

DataInputStream dis=new DataInputStream(bin);

String name=dis.readUTF();//从字节数组中读取

序列化有很多种协议，Java默认的序列化协议在空间与时间上的开销都不好。https://blog.csdn.net/baiye\_xing/article/details/73249819

Java序列化： 存的不仅仅是数据还有类型

序列化输出与输入： 不需要考虑异常抛出（不需要自己处理）

输出到文件：

ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("student.txt"));

oos.writeObject(new Student(10090, "小明", 20));

oos.close();

ObjectInputStream ois = new ObjectInputStream(new FileInputStream("student.txt"));

Student s = (Student)ois.readObject();

System.out.println(s.id);

产生的文件内容是这样子的“id:10090 name:小明 age:20”

如果要将序列化的内容输入/输出到字节数组，那么应：

ByteArrayOutputStream baos = new ByteArrayOutputStream(); //ByteArrayOutputStream捕捉内存缓冲区的数据转换为字节流。构造函数支持传递一个int值作为内部byte缓冲区的大小。

ObjectOutputStream oos2 = new ObjectOutputStream(baos);

oos2.writeObject("Save another object:\n");

oos2.writeObject(new Person("Phil", "654321")); //将数据写入到内部的byte缓冲区中

oos2.close();

ObjectInputStream ois2 = new ObjectInputStream(new ByteArrayInputStream(baos.toByteArray()));

s = (String)ois2.readObject();

p = (Person)ois2.readObject();

System.out.println(s + p);

在public等域作用关键字后面加上transient，就能指示不希望被序列化的数据成员。一个静态变量不管是否被transient修饰，均不能被序列化，因为不算是对象的一部分了。但是其实还是可以让被transient修饰的成员被序列化：

public class Person implements Serializable {

private String userName;

private transient String password;

public Person(String userName, String password) {

this.userName = userName;

this.password = password;

}

public String toString() {

return "userName:" + userName + " password:" + password;

}

private void writeObject(ObjectOutputStream out) throws IOException {

out.defaultWriteObject(); //序列化所有非transient字段,必须是该方法的第一个操作

out.writeObject(password); //序列化transient字段

}

private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {

in.defaultReadObject(); //反序列化所有非transient字段,必须是该方法的第一个操作

password = (String)in.readObject(); //反序列化transient字段

}

}

ArrayList本身实现了Serializable接口

Scanner类 提供格式化输入功能

PrintStream类 提供进行格式化输出功能

PrintStream 为其他输出流添加了功能，使它们能够方便地打印各种数据值表示形式。与其他输出流不同，PrintStream 永远不会抛出IOException！

同时提供一个写入自动刷新功能，即在write函数之后自动调用flush函数。

PrintStream 打印的所有字符都使用平台的默认字符编码转换为字节。

创建对象提供一个OutStream：

FileOutputStream out=new FileOutputStream("D:/test.txt");

PrintStream p=new PrintStream(out);

读取整个文件的时候可以通过配合文件操作类File，获取File.length 来动态new一个数组。之后直接读取到这个数组中。如果文件过大就分块读取。

Java的输出可以到一个文件，也可以到字节数组（如果视为缓冲区，那么和输出到标准输出流中一样，都是输出到内存上的缓冲区）。

94.文件操作： File 类

学习网址：http://www.runoob.com/java/java-file.html

File file=new File(dest);

if(!file.exists())

file.createNewFile(); //可以是写文件的时候创建

95.缓冲区 ByteBuffer 、CharBuffer、StringBuffer 缓冲区的作用就是将多次的读写通过缓冲变成一次

ByteBuffer buffer=ByteBuffer.allocate(256); //分配指定大小的空间最为缓冲区

ByteBuffer buffer=ByteBuffer.wrap(byteArray); //直接将数组包含进来，从而缓冲区中就会拥有与数组一样的数据

buffer.flip();//人称"反转此缓冲区"。因为就像C语言中的文件指针一样到了整个缓冲区的最后，需要将它挪到最前面。

buffer.position() 得到最后的位置

buffer.clear();

int bytesReaded=socketChannel.read(buffer); //从套接字通道读取数据到缓冲区

socketChannel.write(buffer); //从缓冲区写入数据到套接字缓冲区

提供的这些方法就是ByteBuffer以及byte[]数组之间的区别

StringBuffer 类中的方法主要偏重于对于字符串的变化，例如追加、插入和删除等，这个也是StringBuffer和String类的主要区别。

常用函数：

append、insert、reverse、setCharAt、deleteCharAt、trimToSize

96.

char[] cChar=new char[]{'a','b'};

//char[] cChar=new char[2]{'a','b'}; 是错误的

97.正常来讲越贴近环境越接近底层是越快的。

98.使用get()和set()而不使用public的成员变量，是因为虽然两者看起来是效果上等价，但是在后期升级上面是不同的，比如加上限制判断，或者因为上面原因吧需要加上异常处理。特别是类的编写与使用部门不是同一个的时候。

98.数组复制：

System.arraycopy(a, 0, b, 0, a.length);

99.StringBuilder

因为String是不可修改的类，所以每一次改动都会创建一个新的对象。这时不如使用同等地位的StringBuilder，其中的具体原理与C++中的vector一样， 会按照预定义的内存扩容方式，控制好内存缓冲区。（ArrayList数组同样也是实现了自动扩容）

101.异或运算

a ^= b ^= a ^=b;

不等价于

a ^=b;

b ^= a;

a ^= b;

b ^= a ^=b;

a^=b; 这样子的两条式子却又是对的

这是因为编译器的优化，导致式子里面的两个a是同一个数值。于是经过了a ^= b ^= a ^=b; a一定会变成0。

102. 支持类里的声明时候的初始化：

public class Test {

int[] id =new int[120];

public static void main(String[] args) {

}

}

103. 在一个文件中使用另一个文件中定义的类，如果没有设置包名就可直接使用，如果设置了包名就需啊哟import 或者使用的时候类名签名都加上包名。

104.ATM.java:10: 错误: 不兼容的类型: int无法转换为boolean

while(1) {

105. 字符串常量多行字符：

System.out.println("\nMain menu\n"+

"1:check balance\n"+

"2:withdraw\n"+

"3:deposit\n"+

"4.exit\n"+

"Enter a choice:");

不能使用C、C++中在字符串后面使用一个\来包含下一行为本行扩展。那个功能是编译器特殊支持的。

106.Socket 需要使用io与net包中的相关类

TcpClient.java

import java.io.\*;

import java.net.\*;

public class TcpClient {

public static void main(String[] args) throws Exception{

String sentence;

String modifieddSentence;

BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));

Socket clientSocket = new Socket("127.0.0.1",5678);

DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());

BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

sentence = inFromUser.readLine();

outToServer.writeBytes(sentence + '\n');

modifieddSentence = inFromServer.readLine();

System.out.println("From Server:" + modifieddSentence);

clientSocket.close();

}

}

TcpServer.java

import java.io.\*;

import java.net.\*;

public class TcpServer {

public static void main(String argv[]) throws Exception{

String clientSentence;

String capitalizedSentence;

ServerSocket welcomeSocket = new ServerSocket(5678);

while(true) {

Socket connectionSocket = welcomeSocket.accept();

BufferedReader inFromClient = new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));

DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());

clientSentence = inFromClient.readLine();

capitalizedSentence = clientSentence.toUpperCase() + '\n';

outToClient.writeBytes(capitalizedSentence);

}

}

}

UdpClient.java

import java.io.\*;

import java.net.\*;

public class UdpClient {

public static void main(String args[]) throws Exception{

BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));

DatagramSocket clientSocket = new DatagramSocket();

InetAddress IPAdress = InetAddress.getByName("127.0.0.1");

byte[] sendData = new byte[1024];

byte[] receiveData = new byte[1024];

String sentence = inFromUser.readLine();

sendData = sentence.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData,sendData.length,IPAdress,987);

clientSocket.send(sendPacket);

DatagramPacket receivePacket = new DatagramPacket(receiveData,receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence = new String(receivePacket.getData());

System.out.println("From Server:" + modifiedSentence);

clientSocket.close();

}

}

UdpServer.java

import java.io.\*;

import java.net.\*;

public class UdpServer {

public static void main(String args[]) throws Exception{

DatagramSocket serverSocket = new DatagramSocket(987);

byte[] receiveData = new byte[1024];

byte[] sendData = new byte[1024];

while(true) {

DatagramPacket receivePacket = new DatagramPacket(receiveData,receiveData.length);

serverSocket.receive(receivePacket);

String sentence = new String(receivePacket.getData());

InetAddress IPAddress = receivePacket.getAddress();

int port = receivePacket.getPort();

String capitalizedSentence = sentence.toUpperCase();

sendData = capitalizedSentence.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendData,sendData.length,IPAddress,port);

serverSocket.send(sendPacket);

}

}

}

TcpClient先于TcpServer运行会抛出异常，UdpClient却可以先于UdpServer执行。

NIO技术：

IoAcceptor acceptor = new NioSocketAcceptor();//创建一个非阻塞的server端的Socket

107.java中访问修饰符 默认default

| **权限** | **类内** | **同包** | **不同包子类** | **不同包非子类** |
| --- | --- | --- | --- | --- |
| private | √ | × | × | × |
| default | √ | √ | × | × |
| protected | √ | √ | √ | × |
| public | √ | √ | √ | √ |

108.不同的机子上面class文件不一定可以运行，所以需要java重新javac。

110.Android开发中

Bitmap是一种类

Bitmap.Config也是一个公开类，通过这个类来设置Bitmap，引入构造函数

111.List与ArrayList的关系就像是Calendar与GeometricCalendar之间的关系。前者是抽象类，所以不能创建对象；后者是具体实现的类，所以一般都是通过前者来使用后者类的实例。Calendar的使用中存在一种使用方法：Calendar c =Calendar.getInstance()；实际上就是new GeometricCalendar()，构造了这个类的实例。

List list =new ArrayList<类名>();基本上不能直接在构造List的时候传入参数，目前只见过一种看起来有点像的方式：new ArrayList<类名>(Arrays.asList("123","123"));通过这种方式转为List之后传入进去。

112.

Arrange t1=new Arrange("1","1","1");

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

list.add(t1);

System.out.println(list.contains(t1)); 重新new出来的t2只是值一样，但是变量不同

113.接口：

[修饰符] interface 接口名 [extends 父接口名列表]{

[public] [static] [final] 常量;

[public] [abstract] 方法;

...比如static方法块

}

114.

在java语言中，注释有三种:

1) 单行注释： 用 "//" 开头，后跟注释内容。

2) 多行注释： 用 "/\*" 开头，"\*/" 结尾，其中包含多行注释内容。

3) 文档注释：　用 "/\*\*"开头，"\*/" 结尾，其中包含多行注释内容。

115.Java只能继承自一个类，要想多重继承，可以通过内部类来实现。https://www.cnblogs.com/haodawang/p/5967249.html

116.assert：https://www.cnblogs.com/wardensky/p/4307848.html

117.在Java虚拟机规范中试图定义一种Java内存模型（Java Memory Model，JMM）来屏蔽各个硬件平台和操作系统的内存访问差异,它定义了程序中变量的访问规则，往大一点说是定义了程序执行的次序。注意，为了获得较好的执行性能，Java内存模型并没有限制执行引擎使用处理器的寄存器或者高速缓存来提升指令执行速度，也没有限制编译器对指令进行重排序。也就是说，在java内存模型中，也会存在缓存一致性问题和指令重排序(编译器想要优化程序而按照数据依赖性对指令重排，可是多线程代码似乎会出错)的问题。

（原子性）在Java中，对基本数据类型的变量的读取和赋值操作是原子性操作，即这些操作是不可被中断的，要么执行，要么不执行：

x = 10; //语句1

y = x; //语句2

x++; //语句3

x = x + 1; //语句4

上面四条语句中只有语句1是原子操作。语句2实际上包含2个操作，它先要去读取x的值，再将x的值写入工作内存，虽然读取x的值以及 将x的值写入工作内存 这2个操作都是原子性操作，但是合起来就不是原子性操作了。同样的，x++和 x = x+1包括3个操作：读取x的值，进行加1操作，写入新的值。由这些语句可以知道volatile不能保证原子性。

（可见性）Java提供了volatile关键字来保证可见性。当一个共享变量被volatile修饰时，它会保证修改的值会立即被更新到主存，当有其他线程需要读取时，它会去内存中读取新值。而普通的共享变量不能保证可见性，因为普通共享变量被修改之后，什么时候被写入主存是不确定的，当其他线程去读取时，此时内存中可能还是原来的旧值，因此无法保证可见性。另外，通过synchronized和Lock也能够保证可见性，synchronized和Lock能保证同一时刻只有一个线程获取锁然后执行同步代码，并且在释放锁之前会将对变量的修改刷新到主存当中。因此可以保证可见性。

（指令有序性）在Java里面，可以通过volatile关键字来保证一定的“有序性”（具体原理在下一节讲述）。另外可以通过synchronized和Lock来保证有序性，很显然，synchronized和Lock保证每个时刻是有一个线程执行同步代码，相当于是让线程顺序执行同步代码，自然就保证了有序性。（其实就是可见性）。

volatile关键字禁止指令重排序有两层意思：

1）当程序执行到volatile变量的读操作或者写操作时，在其前面的操作的更改肯定全部已经进行，且结果已经对后面的操作可见；在其后面的操作肯定还没有进行；

2）在进行指令优化时，不能将在对volatile变量访问的语句放在其后面执行，也不能把volatile变量后面的语句放到其前面执行。

//x、y为非volatile变量

//flag为volatile变量

x = 2; //语句1

y = 0; //语句2

flag = true; //语句3

x = 4; //语句4

y = -1; //语句5

由于flag变量为volatile变量，那么在进行指令重排序的过程的时候，不会将语句3放到语句1、语句2前面，也不会讲语句3放到语句4、语句5后面。但是要注意语句1和语句2的顺序、语句4和语句5的顺序是不作任何保证的。volatile关键字能保证，执行到语句3时，语句1和语句2必定是执行完毕了的，且语句1和语句2的执行结果对语句3、语句4、语句5是可见的。

“观察加入volatile关键字和没有加入volatile关键字时所生成的汇编代码发现，加入volatile关键字时，会多出一个lock前缀指令”，volatile 的读性能消耗与普通变量几乎相同，但是写操作稍慢，因为它需要在本地代码中插入许多内存屏障指令来保证处理器不发生乱序执行。

synchronized 加到 static 方法前面是给class 加锁，即类锁；而synchronized 加到非静态方法前面是给对象上锁。synchronized(this)是对象锁，如果有多个对象就有相对应的多个锁，synchronized(类的名.class)是类锁，不管有几个对象就公用一把锁。

119. .class 这个类的默认成员返回的类型是java.lang.Class

120. 在try块中有System.exit(0);这样的语句，System.exit(0);是终止Java虚拟机JVM的，连JVM都停止了，所有都结束了，当然finally语句也不会被执行到。不过如果try里面有return， 那么finally会被执行，return的时候是复制了一个变量然后返回，所以之后finally操作的变量如果是基本类型的话不会影响返回值。 但是如果返回值是引用类型的话，因为指向同一个对象所以还是有影响的。

121.类中类可以调用外边类的“类名.this”。

122.虽然号称Java的&&也是有结合性，但是 前面已经判断出false了，它还是会继续去判断后面， 不同C语言。可以用一个对象去测试：

if(false && obj.xxx)

xx

else

obj =new Classxxx();

123.多线程中的中断函数：

首先，一个线程不应该由其他线程来强制中断或停止，而是应该由线程自己自行停止。所以，Thread.stop, Thread.suspend, Thread.resume 都已经被废弃了。而 Thread.interrupt 的作用其实也不是中断线程，而是「通知线程应该中断了」，具体到底中断还是继续运行，应该由被通知的线程自己处理。具体来说，当对一个线程，调用 interrupt() 时，① 如果线程处于被阻塞状态（例如处于sleep, wait, join 等状态），那么线程将立即退出被阻塞状态，并抛出一个InterruptedException异常。仅此而已。② 如果线程处于正常活动状态，那么会将该线程的中断标志设置为 true，仅此而已。被设置中断标志的线程将继续正常运行，不受影响。interrupt() 并不能真正的中断线程，需要被调用的线程自己进行配合才行。也就是说，一个线程如果有被中断的需求，那么就可以这样做。① 在正常运行任务时，经常检查本线程的中断标志位，如果被设置了中断标志就自行停止线程。② 在调用阻塞方法时正确处理InterruptedException异常。（例如，catch异常后就结束线程。）Thread thread = new Thread(() -> {

while (!Thread.interrupted()) {

// do more work.

}

});

thread.start();

// 一段时间以后

thread.interrupt();

Thread.interrupted()清除标志位是为了下次继续检测标志位。

124.DecimalFormat 可以限制float、double等的位数。

DecimalFormat fnum = new DecimalFormat("##0.00");

fnum.format(Float.parseFloat("12.3456"))

125.类型符:

类型参数“<T>”主要用于声明泛型类或泛型方法。

无界通配符“<?>”主要用于使用泛型类或泛型方法。

ArrayList<E>中的“E”也是类型参数。只是表示容器中元素Element的时候，习惯用“E”。

126.Java 远程调用RMI：https://blog.csdn.net/a19881029/article/details/9465663

能够让在某个java虚拟机上的对象像调用本地对象一样调用另一个java 虚拟机中的对象上的方法，使用socket实现

129. Java单例:https://blog.csdn.net/goodlixueyong/article/details/51935526 推荐使用文章中介绍的方法3、5,尤其是最后一种通过枚举来实现。使用枚举除了线程安全和防止反射调用构造器之外，还提供了自动序列化机制，防止反序列化的时候创建新的对象。因此，《Effective Java》作者推荐使用的方法。

Java scope 分为单例与多例：

singleton作用域：当把一个Bean定义设置为singleton作用域是，Spring IoC容器中只会存在一个共享的Bean实例，并且所有对Bean的

请求，只要id与该Bean定义相匹配，则只会返回该Bean的同一实例。值得强调的是singleton作用域是Spring中的缺省作用域。

prototype作用域：prototype作用域的Bean会导致在每次对该Bean请求（将其注入到另一个Bean中，或者以程序的方式调用容器的getBean()方法）时都会创建一个新的Bean实例。根据经验，对有状态的Bean应使用prototype作用域，而对无状态的Bean则应该使用singleton作用域。

Dagger2 Scope分为：

Dagger 2默认并不提供@ActivityScope 、 @ApplicationScope 这些注解。这些只是最常用的自定义Scope。只有@Singleton scope是默认提供的（由Java自己提供）

@Singleton可以保持类的单例。

@ApplicationScope注解的Component类与Applicaiton对象的生命周期一致。

@ActivityScope注解的Component类与Activity的生命周期一致

130.