# JAVA

Char数组转换为string：String。valueOf（char【】 char）

知道Java是没有数组/字符串结束符的。与C中的char【】不一样

尾递归：在函数的最后一行调用递归函数。可以利用循环语句来代替。

## Java中数组的常用操作

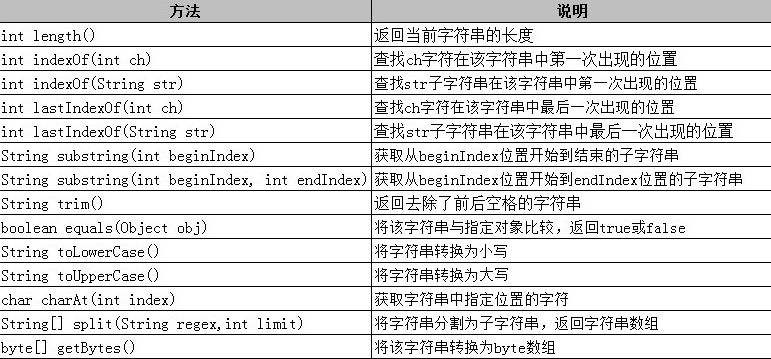
java中有一个类工具，Arrays类，里面包含很多方法可以对数组进行操作：

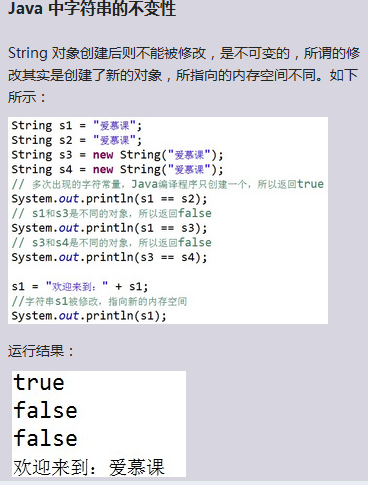
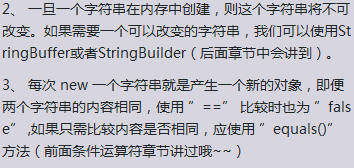
1. fill对数组的值进行填充
2. **Array类**中copyof和copyOfRange以及**System类**中的arraycopy 可以对数组进行复制
3. sort数组排序
4. binarySearch二分法查找数据。

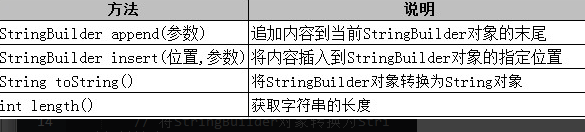
参见<http://www.imooc.com/wiki/view?pid=77>

## java中的String类型

1. **string类的方法**



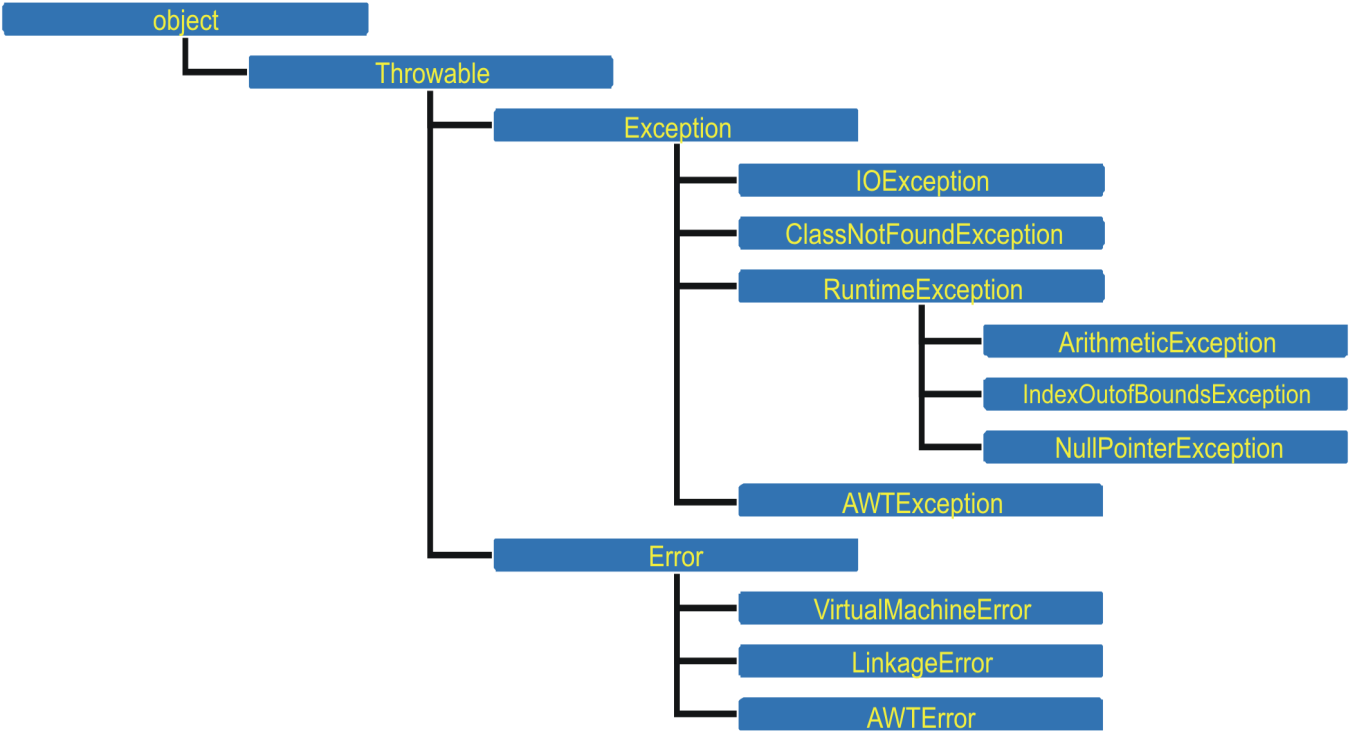
1. ==: 判断两个字符串在内存中首地址是否相同，即判断是否是同一个字符串对象equals(): 比较存储在两个字符串对象中的内容是否一致
2. 字节是计算机存储信息的基本单位，1 个字节等于 8 位， gbk 编码中 1 个汉字字符存储需要 2 个字节，1 个英文字符存储需要 1 个字节（在Unicode编码中，字符是双字节的，java采用Unicode编码）。所以我们看到上面的程序运行结果中，每个汉字对应两个字节值，如“学”对应 “-47 -89” ，而英文字母 “J” 对应 “74” 。同时，我们还发现汉字对应的字节值为负数，原因在于每个字节是 8 位，最大值不能超过127，而汉字转换为字节后超过 127，如果超过就会溢出，以负数的形式显示。
3. 
4. 
5. 在Java中，除了可以使用 String 类来存储字符串，还可以使用 StringBuilder 类或 StringBuffer 类存储字符串，String 类具有是不可变性，当频繁操作字符串时，就会额外产生很多临时变量。使用 StringBuilder 或 StringBuffer 就可以避免这个问题。至于 StringBuilder 和StringBuffer ，它们基本相似，不同之处，StringBuffer 是线程安全的，而 StringBuilder 则没有实现线程安全功能，所以性能略高。因此一般情况下，如果需要创建一个内容可变的字符串对象，应优先考虑使用 StringBuilder 类

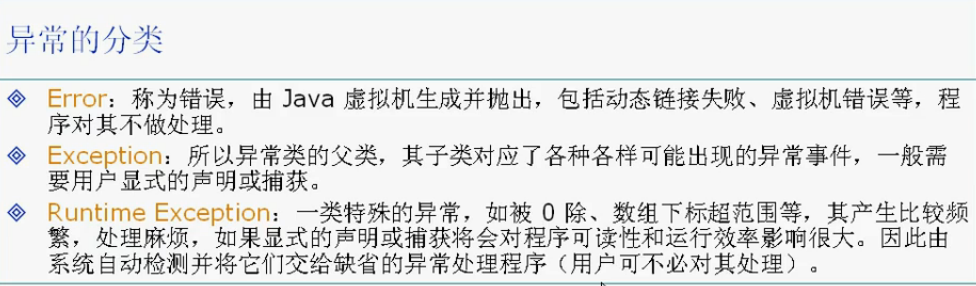


## 3.多种内部类的联系和区别

|  |  |  |  |
| --- | --- | --- | --- |
| **匿名内部类** | **内部类** | **方法（局部）内部类** | **静态内部类** |
| 多用于接口等的一次性实现。 | 可以使用外部类的任何成员。 | 定义在类的方法的内部 | 用static修饰的内部类。 |
| 编程经常使用 | 常用 |  | 不常用 |
| 可以省去定义一个新的类和对象的步骤 | 必须使用外部类对象来创建内部类对象，而不能直接去 new 一个内部类对象 | 方法内部类只在该方法的内部可见，通常在方法内部实现实例化操作和调用等。 | 在实例化的时候只需要用“new 内部类名”不需要像内部类那样通过外部对象来实例化。 |
| 编译文件通常是有外部类$序列号的方式.class。 | 编译上面的程序后，会发现产生了两个 .class 文件，第二个是外部类的 .class 文件，第一个是内部类的 .class 文件，即成员内部类的 .class 文件总是这样：外部类名$内部类名.class |  | 如果外部类的静态成员与内部类的成员名称相同，可通过“类名.静态成员”访问外部类的静态成员；如果外部类的静态成员与内部类的成员名称不相同，则可通过“成员名”直接调用外部类的静态成员 |

## java异常处理



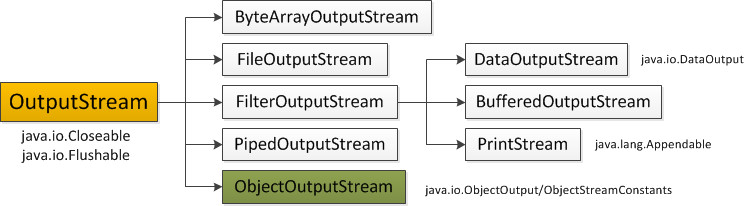
1. try …catch…finally :finally后面的语句无论是否出现异常都会执行。
2. throws声明抛出异常 当目前方法不希望处理程序中可能出现的异常时可以用抛出异常的方法，让调用方法的上层来处理异常或者继续抛出异常到更高的调用层。
3. throw 常被用来主动抛出异常
4. 自定义异常需要继承Exception 并输入一个字符串类型的参数。
5. Runtimeexception异常可以捕获然后处理，也可以不对其做处理。而另外的异常如IOexception异常则必须处理。
6. 

## 输入输出流：

理解几个概念：字符、字节、文件、文本文件、编码方式、读取方式（以多少个字节进行读取）

抽象类InputStream和OutputStream在输入输出时都是一个一个byte进行的。

Reader和Writer在输出输入是是按字符进行的。在处理字符文本时比InputStream和OutputStream更方便。

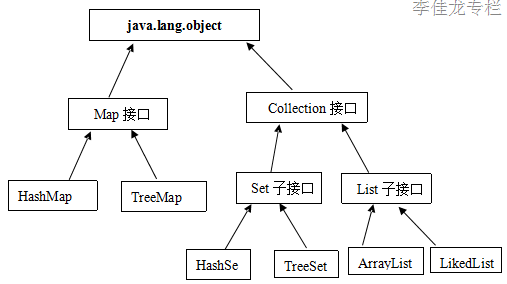


**注意：对输入输出流的“输入和输出”是相对哪个要理解清楚！！（难点）**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 构造函数 | 成员方法 | 备注 |
| InputStream | 抽象类，无 | Read，reset，close，skip等 | 所有字节输入类的父类，skip等方法要在含有buffer的子类中才能使用。 |
| Reader | 抽象类 | 类似于InputStream，每次读入以字符为单位。具体参看API | 字符输入流的父类 |
| FileInputStream | FileOupputStream(File file)  FileOupputStream(String name file) | Read（arg0）：把输入流中的数据读入到arg0里面。 |  |
| Filereader | FileReader(File file) | Read() |  |
| BufferedInputStream |  | Read（） | 比InputStream多了一个缓冲区 |
| **DataInputStream** | **DataInputStream（InputStream in）** | **WriteBytes**  **writeChars**  **writeUTF** | 注意三种不同的写入字符串的方法 |
| BufferedReader |  | Read，readLine，newLine，**flush，**write | 比Reader多了一个缓冲区 |
|  |  |  |  |
|  |  |  |  |

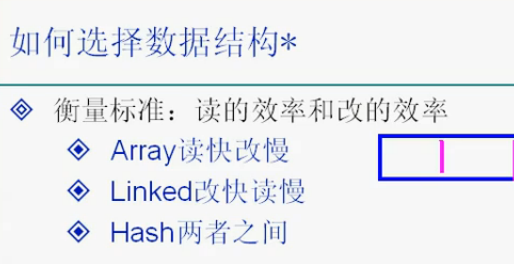
**注意**：bytes=in.read(buff,off,len)函数会返回一个当前读到的字节数。当输入流中的字节被读完时这时bytes的值为最后一次读到的字节数，如果读完后再次运行该函数，此时会返回-1表示读到末尾了**（把内容读完返回的不是-1，要再次运行才返回-1）**

## 关于集合collection的相关知识：

1. 关系图：
2. Set：无序不能重复

List：有序能重复

Map：以键值对的形式存储

1. 要对list排序的话需要重写equals和hashcode方法。索引的时候通常比较hashcode，**对象如果内容一样的话，那么hashcode也应该一样。**
2. 

HashMap类的实现基础：可以是用数组来实现的

## 泛型的知识

1. 泛型类：就是具有一个或多个**类型变量**（T就是一个类型变量）的类。

Public class Pair<T>{ T var0; public void T func(){};}

泛型类像一个普通类的工厂。

1. 泛型方法：
2. 类型变量的限定：

## java的线程（thread）难点！！！！！！！！！！！

有助于理解的文章：http://wang09si.blog.163.com/blog/static/170171804201373194145416/

注意：wait()和notify()一系列的方法，是属于对象的，不是属于线程的。

**wait方法**导致当前线程（称之为 T）将其自身放置在**对象的等待集中**，然后放弃此对象上的所有同步要求。

调用wait方法时，当前线程必须拥有此对象监视器。

通过以下三种方法之一，线程可以成为此对象监视器的所有者：

通过执行此对象的同步实例方法。

通过执行在此对象上进行同步的 synchronized 语句的正文。

对于 Class 类型的对象，可以通过执行该类的同步静态方法。

**notify()唤醒**在此对象监视器上等待的单个线程。如果所有线程都在此对象上等待，则会选择唤醒其中一个线程。选择是任意性的，并在对实现做出决定时发生。

**notify()**并不会让当前线程休眠，但会唤醒休眠的线程。

如果没有线程在wait，调用notify是不会有什么问题的。

尽量少锁定对象，锁定的力度尽量大一些，对访问某个资源的所有代码或方法都要仔细考虑是否应该加同步锁。

1. 当程序中有：

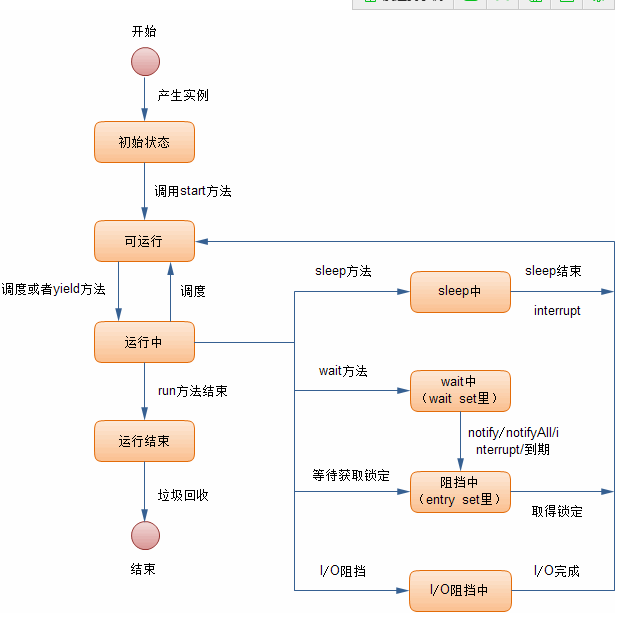
Thread1.start（）

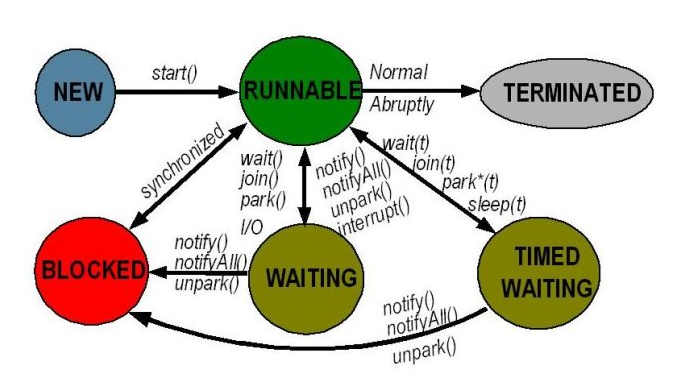
Thread2.start（）

Thread3.start（）

执行时，表示这三个线程同步执行，分段时间控制cpu，通常会中每个线程中设置一个睡眠或等待时间，以让出cpu等系统资源。

1. 线程的几种运行状态：



1. 

**1.新建状态，**和其他java对象一样，仅仅在堆区中被分配了内存。

**2.就绪状态，**当一个线程对象创建后，其他线程调用它的start()方法，该线程就进入就绪状态，java虚拟机会为它创建方法调用栈和程序计数器，处于这个状态的线程位于可运行池中，等待获得cpu的使用权。

**3.运行状态**，处于这个状态的线程占用cpu，执行程序代码，在并发运行环境中如果计算机只有一个cpu，那么任何时刻只会有一个线程处于这个状态，如果有多个可以让几个线程占用不同的cpu，只有处于就绪状态的线程才有机会转到运行状态。

**4.阻塞状态**，指线程因为某些原因放弃cpu，暂时停止运行，当线程处于阻塞状态java虚拟机不会给线程分配cpu，直到线程重新进入就绪状态，它才有机会转到运行状态

**阻塞状态可以分为一下3种**

(**1).**位于对象等待池中的阻塞状态

**(2).**位于对象锁池中的阻塞状态

**(3).**其他阻塞状态 当一个线程执行System.out.println();或者System.in.read()方法时，就会发出一个io请求，该线程放弃cpu，进入阻塞状态，直到io请求处理完毕，该线程才恢复运行。

**5.死亡状态**，当线程退出run()方法时，就进入了死亡状态，该线程结束生命周期，不管线程正常结束还是异常结束，都不会对其他线程造成影响。

1. **suspend() / resume() 方法和wait() / notify() 方法的对比**

l suspend()/resume() 方法

两个方法配套使用，suspend()使得线程进入阻塞状态，并且不会自动恢复，必须其对应的resume() 被调用，才能使得线程重新进入可执行状态。典型地，suspend() 和 resume() 被用在等待另一个线程产生的结果的情形：测试发现结果还没有产生后，让线程阻塞，另一个线程产生了结果后，调用 resume() 使其恢复。

l wait()/notify() 方法

两个方法配套使用，wait() 使得线程进入阻塞状态，它有两种形式，一种允许 指定以毫秒为单位的一段时间作为参数，另一种没有参数，前者当对应的 notify() 被调用或者超出指定时间时线程重新进入可执行状态，后者则必须对应的 notify() 被调用。

l 两者的对比

区别的核心在于，suspend()/resume() 方法，阻塞时都不会释放占用的锁（如果占用了的话），而wait()/notify() 方法则相反。

首先，suspend()/resume() 方法都隶属于 Thread 类，而wait()/notify() 方法却直接隶属于 Object 类，也就是说，所有对象都拥有这一对方法。

其次，suspend()/resume() 方法都可在任何位置调用，但是wait()/notify() 方法却必须在 synchronized 方法或块中调用，理由也很简单，只有在synchronized 方法或块中当前线程才占有锁，才有锁可以释放。若不满足这一条件，则程序虽然仍能编译，但在运行时会出现IllegalMonitorStateException 异常。

关于 wait()/notify() 方法的两点说明：

第一：调用 notify() 方法导致解除阻塞的线程是从因调用该对象的 wait() 方法而阻塞的线程中随机选取的，我们无法预料哪一个线程将会被选择，所以编程时要特别小心，避免因这种不确定性而产生问题。

第二：除了 notify()，还有一个方法 notifyAll() 也可起到类似作用，唯一的区别在于，调用 notifyAll() 方法将把因调用该对象的 wait() 方法而阻塞的所有线程一次性全部解除阻塞。当然，只有获得锁的那一个线程才能进入可执行状态。

1. java的**sychronized关键字**的用法

Java语言的关键字，当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码。

一、当两个并发线程访问同一个对象object中的这个synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。

二、然而，当一个线程访问object的一个synchronized(this)同步代码块时，另一个线程仍然可以访问该object中的非synchronized(this)同步代码块。

三、尤其关键的是，当一个线程访问object的一个synchronized(this)同步代码块时，其他线程对object中所有其它synchronized(this)同步代码块的访问将被阻塞。

四、第三个例子同样适用其它同步代码块。也就是说，当一个线程访问object的一个synchronized(this)同步代码块时，它就获得了这个object的对象锁。结果，其它线程对该object对象所有同步代码部分的访问都被暂时阻塞。

五、以上规则对其它对象锁同样适用.

参考：<http://blog.csdn.net/kyfg27_niujin/article/details/7942006>

1. 主线程和各个线程之间的运行顺序和通信顺序。

## java数据库开发

在用java编写的应用程序和数据库两者的连接中需要两个因素：一个是JDBC的API接口（开发商自己编写好的相应产品的驱动，以满足统一标准的API接口功能），另一个是DBMS数据库管理系统。

**基本开发步骤：**

1. 根据各种不同数据库加载他们对应的驱动程序。
2. 使用java.sql的DriverManager来连接数据库
3. 使用java.sql的Connection接口来创建Statement 对象来执行sql语句
4. 用Resultset接口接受返回的查询数据
5. 关闭各种程序资源

## Java的Swing设计

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 文本组件 | 常用的窗体 | 常用布局器 | 常用面板 | 标签与图标 | 按钮组件 | 复选框组件和列表组件 |
| JTextField | JFrame | 绝对布局 | JPanel | JLabel | JButton | JCheckBox |
| JPassswordField | JDialog | FlowLayout | JScrollPane | Icon | JRadioButton | JComboBox类和JComboBox模型 |
| JTextArea |  | BorderLyaout |  | ImageIcon | ButtonGroup | JList |
|  |  | GridLayout |  |  |  |  |
|  |  |  |  |  |  |  |

## Java中的事件及监听

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 键盘事件 | 鼠标事件 | 选项事件 | 窗体事件 |
| 监听器 | KeyListener | MouseListener | ItemListener | WindowFocusListener;  WindowStateListener;  WindowListener |
| 要实现的方法 | keyTyped()  keyPressed()  keyReleased() | mouseEntered;  mousePressed;  mouseReleased;  mouseClicked;  mouseExited | itemStateChanged(ItemEvent e) | windowGainedFocus,  windowLostFocus;  windowStateChanged;关于WindowListener有七个方法需要实现（见API手册） |
| 常用的事件方法 | getSource()  getKeyChar()  getKeyCode()  getKeyText()  isActionKey()  isControlDown()  isAltDown()  isShaiftDown() | getSource()  getButton()  getClickCount() | getItem();  getStateChange() | getOldState;  getNewState; |

## 项目设计大体流程范例

以设计快递打印系统为例：

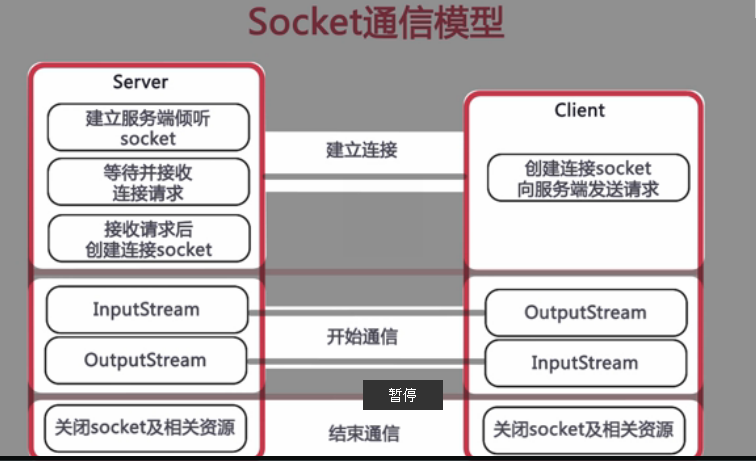
1. 需求分析
2. 总体设计
   1. 系统目标
   2. 构建开发环境
   3. 系统功能结构图
3. 数据库设计
   1. 数据库概要说明
   2. 数据库E-R图
   3. 数据表结构
4. 公共类设计
5. 程序主要功能版块设计
   1. 功能版块1
      1. 技术分析
      2. 具体实现
   2. 功能版块2
   3. 功能版块3
6. 程序打包与安装

## JAVA的网络通信

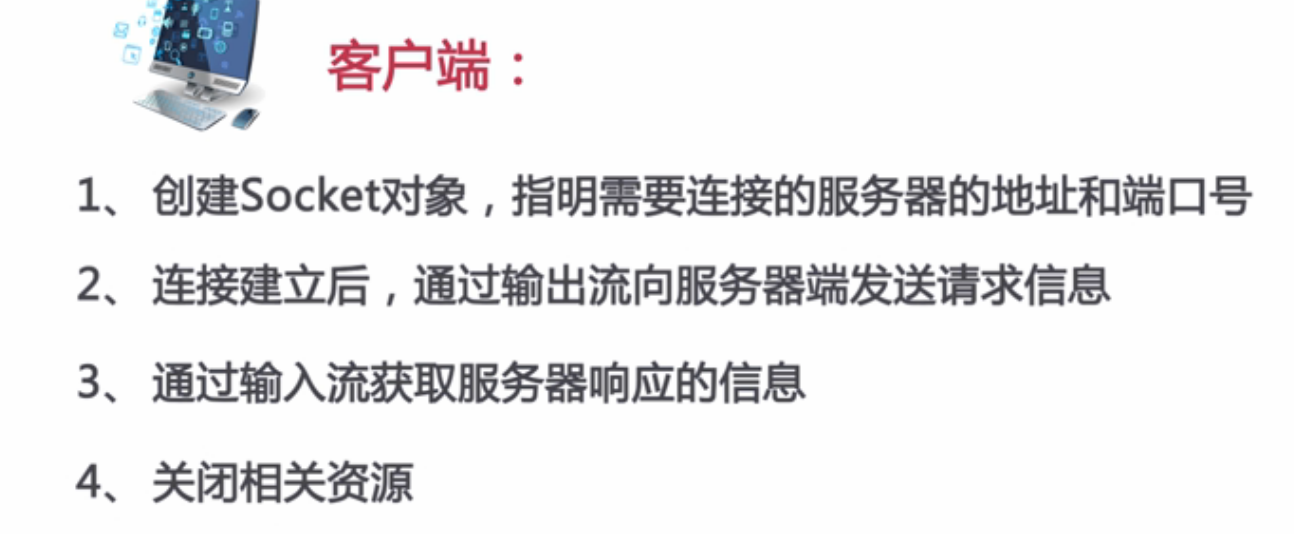
Socket：TCP通信

Datagram：UDP通信

多线程通信程序Socket







## Java Design Pattern

**单例模式：**只需要一个实例的情况下，如 线程池，配置文件等。构造函数private化，使用static关键字，返回一个在类中创建好的实例。

**模版模式：**对于很多工程的实现时有一个固定的大的框架，而在部分地方有不同。这种情况可以使用模版设计。对于模版的流程可以用一个final关键字修饰，使得子类不能做修改。对于子类都拥有的方法或者成员可以用private修饰。对于不同的地方，应该用abstract关键字来定义一个抽象的方法，让子类自己单独实现。同时可以结合**钩子函数**，来决定某个步骤是否对某些子类是没有必要的。

**策略模式：**策略模式的使用最大好处在于提高代码的复用率。在父类中定义一个接口成员，并调用他规定的方法。然后在子类中调用实现了该接口的类的构造函数。

## 反射

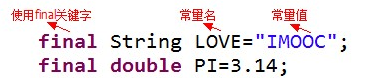
## 序列化

## 其它知识点总结

1. return在没有返回值的时候的作用是中断程序。用在返回值为Void的程序中。

return语句通常用在一个方法体的最后，以退出该方法并返回一个值。在Java语言中，单独的**return语句用在一个方法体的中间时，会产生编译错 误，因这会使得一些语句执行不到。但可以通过把return语句嵌入某些语句**（如 if…else）来使程序在未执行完方法中的所有语句时退出。

1. java中获取时间常用的两个类是Date类和Calendar类。
2. Enumerable和Enumerator是C#中的
3. 类型推断是从java 7开始的。ArrayList<int> array=new ArrayList<>();
4. Java中Math类中的各种计算的方法。
5. 产生随机数有两种方法：一种：用Math类里面的random方法来实现，另一种：实例化一个java的Random类，由其对象来调用自己的相应方法，产生随机数。
6. 数据的格式化主要可以用：
   1. 时间的格式化可以用，String类的format方法
   2. 数字的格式化可以用，DecimalFormat类提供的方法
7. switch case 语句后面必须是整数或者字符型。
8. 数组的初试化和定义与C不同e.g. java：**String[]** yuchang=**new String**[5];C:int a []
9. foreach语句：for（数据类型 数据变量：遍历对象）
10. 判断方法重载的依据：
    1. 必须是在同一个类中
    2. 方法名相同
    3. 方法参数的个数、顺序或类型不同
    4. 与方法的修饰符或返回值没有关系
11. java可以通过返回数组的数组名来实现对整个数组的返回。C和c++不可以
12. 数组名.Length可以看数组的长度。
13. Java语言中调用方法（函数）跟c语言不同，前者由于是封装起来的类，每次都需要通过调用对象的方法来实现，后者是通过直接调用方法名来实现。
14. break：跳出当前**循环** ；continue：跳过循环剩下的语句，进入**下一次循环**。
15. java中的uri包括url和urn两个子集，相当于是存储网址的一个数据类型引用。具体解析见<http://blog.163.com/haizai219@126/blog/static/444125552010811250106/>。
16. 类对象和类的对象的区别。
17. java中的成员变量可以被其它类调用，且自动被赋初值为0，而局部变量,即方法中的变量，不会被赋予初值，且不能被除本方法外的其它方法调用。局部变量可以和成员变量可以重名，但是会优先就近输出局部变量的值，若要区分两者可通过this关键字来实现。
18. static定义的静态的成员变量或者静态方法可以直接通过“类.方法”来调用，也可以通过创建一个对象然后调用对象方法的方式来调用。而普通方法或者成员变量必须通过第二种方法调用。静态方法只能调用同类中的静态变量而不能调用非静态变量且不能用super或者this关键字。Static和final static都相当于全局变量

. 

1. 常量名一般使用**大写字符，多个词用下划线隔开。**

final关键字：

修饰类则该类不能被其它类继承个改写。可用于模版设计当中。

修饰方法则该方法不能被重写；

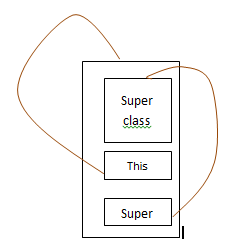
修饰变量则该变量只能被赋值一次 ，相当于常量，如上所定义。

1. 引用类型：是指变量存储的是一个地址。
2. 队列：先进先出

栈：栈限制仅在表的一端进行插入和删除运算的线性表，先进后出FILO

堆：对于堆，我们可以随心所欲的进行增加变量和删除变量，不要遵循什么次序，只要你喜欢

1. **实体类**是就是Java中的O/R Mapping映射，即数据库中的一个表映射成对应的一个Java类,其中还有一个映射文件
2. 多态的用处：可以通过继承（父类引用指向子类对象）和接口来实现。
3. 接口的修饰必须是public，且可以接口再继承接口。接口可以作为成员变量。接口实例可以 调用接口中声明的实现在子类的具体方法。对于实现了多个接口的类，如果将这个类看做某个接口，那么相当于只能看到这个类关于这个接口的这一部分内容。
4. 内存分析：局部变量和形参在执行时会在栈内存中临时分配一块空间，而new出来的对象则会出现在堆内存中。程序中定义的方法则是存在code segment中，方法是动态调用的。引用类型通常占用两块内存空间。Java的引用就相当于指针。**但是这里的引用不是物理地址**。
5. This 表示指向当前对象，super表示指向父对象。内存中的原理图如下。



1. 不写访问权限是默认为default权限，在继承的时候，子类拥有父类的所有成员和方法，但是不能访问private修饰的。



1. **子类的构造过程必须调用父类的构造方法。如果子类没有显示的调用父类的构造方法，则会默认调用父类的无参构造方法。且如果要调用父类的构造方法要在子类的构造方法的第一行调用。**
2. Foreach循环其实效率不高，不建议使用。
3. java中的Iterator和Iterable 区别

为什么一定要实现Iterable接口，为什么不直接实现Iterator接口呢？   
      看一下JDK中的集合类，比如List一族或者Set一族，都是实现了Iterable接口，但并不直接实现Iterator接口。   
仔细想一下这么做是有道理的。   
  
      因为Iterator接口的核心方法next()或者hasNext() 是依赖于迭代器的当前迭代位置的。   
      如果Collection直接实现Iterator接口，势必导致集合对象中包含当前迭代位置的数据(指针)。   
      当集合在不同方法间被传递时，由于当前迭代位置不可预置，那么next()方法的结果会变成不可预知。   
      除非再为Iterator接口添加一个reset()方法，用来重置当前迭代位置。   
      但即时这样，Collection也只能同时存在一个当前迭代位置。   
      而Iterable则不然，每次调用都会返回一个从头开始计数的迭代器。   
      多个迭代器是互不干扰的。

1. 如果要实现某个类的对象的排序，需要实现接口comparable的compare to方法。

Comparable & Comparator 都是用来实现集合中元素的比较、排序的，只是 Comparable 是在集合内部定义的方法实现的排序，Comparator 是在集合外部实现的排序，所以，如想实现排序，就需要在集合外定义 Comparator 接口的方法或在集合内实现 Comparable 接口的方法。

Comparable接口只提供了   int   compareTo(T   o)方法，也就是说假如我定义了一个Person类，这个类实现了   Comparable接口，那么当我实例化Person类的person1后，我想比较person1和一个现有的Person对象person2的大小时，我就可以这样来调用：person1.comparTo(person2),通过返回值就可以判断了；而此时如果你定义了一个   PersonComparator（实现了Comparator接口）的话，那你就可以这样：PersonComparator   comparator=   new   PersonComparator();  
comparator.compare(person1,person2);

1. 父类引用子类对象有助于代码维护。
2. 泛型的好处：有助于编译时发现错误，有助于自动打包，不需要强制转换类型。
3. Eclipse常用快捷键:

【Alt+↑/↓】 上移/下移当前行(或选中的多行)

【ALT+/】 提供内容的辅助

【Ctrl+/】 快速添加注释

【Ctrl+D】 删除当前行(或选中的多行)

【Ctrl+Shift+O】 快速生成import

【Ctrl+Shift+F】 格式化代码

查找

【F3】快速定位光标位置的某个类、方法和属性。

【F4】显示类的继承关系，并打开类继承视图。

【Ctrl+F】在当前文件查找string

【Ctrl+H】在Workspace内查找string

【Ctrl+O】快速定位类的方法和属性

【Ctrl+L】定位到当前编辑器的某一行

【Ctrl+K】将光标停留在变量上，按Ctrl+K键可以查找到下一个同样的变量

【Ctrl+Shift+T】打开.java

【Ctrl+Shift+R】打开资源

Debug

【F5】进入方法执行。

【F6】单步执行。

【F7】退出当前方法执行。

【F8】执行到断点或程序结束。

## 读书记录

### 一、Think in JAVA

#### 第一章

设计程序的目标是去创建能够提供理想服务的来解决问题的一系列对象。

在建新的类的时候，应该首先考虑组合，组合是指**has-a关系**，如果一个类含有其他成员对象。因为他更加简单灵活。在有了一些经验之后就会明白哪些场合该使用继承。

采用方法的多态性，调用方法采用后期绑定模式。

继承：is-a。尽管private成员被继承了，但是是不能访问的。

区别is-like-a关系。扩展了基类的

**Java都异常处理就像是与程序正常执行路径并行的**，在错误发生时的执行的另一条路径，因为他是另一条路径所以不会干扰到正常的代码。不处理异常且不抛出时，产生异常处后面的程序无法执行，如果处理了或者抛出了，则之后的程序也会执行。

#### 第二章

垃圾回收器在堆内存中，用于决定一个对象的生命周期是否结束。（其工作原理是什么？）

**局部变量要赋予初试值**（但是成员变量会自动赋予默认初始值）。系统不会自动赋值。

如果方法返回值为void，可以用return语句作为退出方法的一种方式，如果不是void，那么无论在什么地方使用return都会编译器强制返回一个正确类型的值。

Java.lang包是自动导入到程序中的。

# ANDROID

## SQLite 应用

主要涉及两个重要的类：**SQLiteDataBase** 和**SQLiteOpenHelper**。

**注意：**一个应用只能使用一个数据库，当新建第二个的时候，能成功创建但是没有任何数据，而且对其操作会报错。

**步骤1：**创建一个数据库，可以使用SQLiteDataBase的creatOrOpenSQLiteDataBase方法或者使用SQLiteOpenHelper的getwritable方法。

**步骤2：**执行对数据进行操作（创建表格，删除表格，插入数据，删除数据，查询数据）：通常可以用insert方法或者exec方法，但是前者更好，因为后者的参数是跟sql语句，在不熟悉的情况下可能会出现拼写错误的情况。如果是查询方法的话可以用rawquery或者query，这两个方法都会返回一个Cursor类的对象，用来读取查出来的结果。（需要配合游标对象的movetoNext等方法使用。

**步骤3：**关闭游标对象，关闭数据库对象。

注意：在使用SQLiteOpenHelper时，1.重新onCreat方法其中主要是初试化数据库，包括新建表格等。2.重写upgrade方法。

**问题：**刚开始的时候对sql语言不是很熟悉。对SQLiteDataBase的增加删除也不是很熟悉

## SQL语句

## 开发经验&易出错使用方法总结

### 开发经验

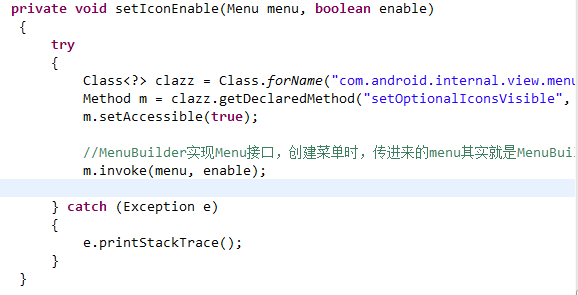
开发项目的经验总结：

1. 整个项目的**功能模块**有哪些，具体的**数据结构**设计，模块通信接口（即需**要哪些输入参数**），需要设计哪些**field 和local variable**以及他们在什么时候**初始化**（个人觉得尤其重要），哪些工具类（比如toast或者log或者xml的inflater或者intent跳转）
2. 调试的重要性和逻辑性。最好遵循**一边coding一边debu**g，这就要求能清楚的明白这个系统的框架结构，先写**那些基础性**的，然后一**步一步搭建**。
3. 一些高级用法如：用接**口实现多态，状态机，模块设计**模式等。
4. 关于BaseAdapter的用法：首先是通过getCout（）返回要绘制的item的数量，然后调用getView（）方法返回的VIew绘制。对于getVIew的三个参数：conterView的理解比较复杂，参看：<http://blog.csdn.net/harvic880925/article/details/25335957>。

注意几点：conterVIew先是被创建值为null，然后被复用（其中涉及Recycler的回收机制）。通常会写判断：conterView是否为null，不是的时候就可以直接调用，效率更高。

可以通过写holder来帮助存储内容，然后setTag方法保存在conterView中以便下次调用。

1. 从A activity跳转到B activity 时可以在Bactivity中通过getIntent获得这个跳转intent进而获得它里面的数据。
2. 在使用adapter的时候如果，对数据源做了增删操作时应该调用adapter。Notifydatasetchange方法来更新数据。
3. 在对activity的oncreateoptionsmenu方法中增加菜单选项时，如果想使用SetIcon方法需要用下面的方法先反射然后在设置



1. Edittext控件里面的内容可以通过.getText().toString().trim()来获得，返回一个String值。

.getEditableText().clear()来清空内容。

1. import static（静态导入）是JDK1.5中的新特性，一般我们导入一个类都用 import com.....ClassName;而静态导入是这样：import static com.....ClassName.\*;这里多了个static，还有就是类名ClassName后面多了个 .\* ，意思是导入这个类里的静态方法。当然，也可以只导入某个静态方法，只要把 .\* 换成静态方法名就行了。然后在这个类中，就可以直接用方法名调用静态方法，而不必用ClassName.方法名的方式来调用。
2. 绘制自制的View。需要三个元素：a一个继承了View的myVIew，这个类的重点是重写onDraw方法，实现自己的界面。传入的canvas对象的draw系列方法有利于绘制图像。

b一个继承了thread的mythread类，这个类用来实现界面的刷新功能，更新各项参数值，其中通常会用到postinvalidate方法来更新参数。C 一个新的activity来执行myView和myThread。

### 易出错点

1. listView不要放到ScrollView中，会导致listView中的内容不能完全显示。只会显示position=0那个item
2. “android:layout\_gravity”是相对与它的父元素说的，说明元素在父元素的位置。

android:gravity这样前面没有“layout”的属性，是对元素本身说的！

## 读书记录

### android核心技术和实例详解

#### 第二章