Java

C/C++的区别

被 final 的类没有子类

子类不继承超类构造函数，要用 super()

Java 原理

整体运作结构

继承与接口区别

接口是声明一种对象间传递消息的协议，这样可以针对接口编程而不必知道底层实现，好处就是在你底层代码发生变动时，只要接口不变就可以改变底层代码而不必改变调用者的代码

继承的概念不用多说，很好理解。为什么要继承呢?因为你想重用代码?这绝对不是理由，继承的意义也在于抽象，而不是代码重用。如果对象 A 有一个 run()方法，对象 B 也想有这个方法，所以有人就 Class B extends A。这是不经大脑的做法。如果在 B 中实例化一个 A，调用 A 的 Run()方法，是不是可以达到同样的目的?如下：

```
Class B
{
A a=new A();
a.run();
}
```

这就是利用类的聚合来重用代码，是委派模式的雏形，是 GoF 一贯倡导的做法。

继承的意义何在?其实这是历史原因造成的，最开始的 OO 语言只有继承，没有接口，所以只能以继承来实现抽象，请一定注意，继承的本意在于抽象，而非代码重用(虽然继承也有这个作用)，这是很多 Java 烂书最严重的错误之一，它们所造成的阴影，我至今还没有完全摆脱，坏书害人啊，尤其是入门类的，流毒太大。什么时候应该使用继承?只在抽象类中使用，其他情况下尽量不使用。抽象类也是不能实例化的，它仅仅提供一个模版而已，这就很能说明问题。

在 abstract class 方式中，可以有自己的数据成员，也可以有非 abstarct 的成员方法，而在 interface 方式的实现中，只能够有静态的不能被修改的数据成员（也就是必须是 static final 的，不过在 interface 中一般不定义数据成员），所有的成员方法都是 abstract 的。从某种意义上说，interface 是一种特殊形式的 abstract class。

String 对象

1，String s = new String("oops!");解释为新建一个引用 sz 指向 String 对象，

String t = s;t 也指向引用 s;

2，那考虑到==，与 equals 的区别，

String r = new String("oops!");

那么 r==t;r==s;r.equals(s);s.equals(t);的最终结果呢？

看我写的一段源代码；

```
class test
{
        public static void main(String args[])
        {
                String s = new String("oops!");
                String t = s;
                String r = new String("oops!");
```

```
        boolean i,j,k;
        i = (r==t);
        j = (r==s);
        k = (t==s);
        System.out.println("r==t:"+i+" r==s:"+j+" t==s:"+k);
        i = r.equals(s);
        j = t.equals(s);
        System.out.println("r.equals(s):"+i+" t.equals(s):"+j);
    }
}
```

结果：

r==t:false r==s:false t==s:true

r.equals(s):true t.equals(s):true

可以看出“==”是比较引用这个“指针”；而“equals”则是比较 String 的值；

3，String s = new Srting("oops!"); String s = s + "ooh!";

由于 Sting 是不可变类，上面的代码会产生两个类，第一个类会被抛弃！所以建议适当使用 StringBuffer；

s = "Initial Value"; s = new String("Initial Value");

这两个的区别就在于后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，。所以~~这个暂时还不懂~

至于为什么要把 String 类设计成不可变类，是它的用途决定的。其实不只 String，很多 Java 标准类库中的类都是不可变的。在开发一个系统的时候，我们有时候也需要设计不可变类，来传递一组相关的值，这也是面向对象思想的体现。不可变类有一些优点，

比如因为它的对象是只读的，所以多线程并发访问也不会有任何问题。当然也有一些缺点，比如每个不同的状态都要一个对象来代表，可能会造成性能上的问题。

还有扩展知识，很重要的。他讲得比我好，就直接给链接吧！

http://www.cnblogs.com/Mr-Hannibal/archive/2012/05/18/2508094.html

原文：//

Java 六大必须理解的问题

对于这个系列里的问题，每个学 Java 的人都应该搞懂。当然，如果只是学 Java 玩玩就无所谓了。如果你认为自己已经超越初学者了，却不很懂这些问题，请将你自己重归初学者行列。内容均来自于 CSDN 的经典老贴。

问题一：我声明了什么！

String s = "Hello world!";

许多人都做过这样的事情，但是，我们到底声明了什么？回答通常是：一个 String，内容是“Hello world!”。这样模糊的回答通常是概念不清的根源。如果要准确的回答，一半的人大概会回答错误。

这个语句声明的是一个指向对象的引用，名为“s”，可以指向类型为 String 的任何对象，目前指向"Hello world!"这个 String 类型的对象。这就是真正发生的事情。我们并没有声明一个 String 对象，我们只是声明了一个只能指向 String 对象的引用变量。所以，如果在刚才那句语句后面，如果再运行一句：

String string = s;

我们是声明了另外一个只能指向 String 对象的引用，名为 string，并没有第二个对象产生，string 还是指向原来那个对象，也就是，和 s 指向同一个对象。

问题二："=="和 equals 方法究竟有什么区别？

==操作符专门用来比较变量的值是否相等。比较好理解的一点是：

int a=10;

int b=10;

则 a==b 将是 true。

但不好理解的地方是：

String a=new String("foo");

String b=new String("foo");

则 a==b 将返回 false。

根据前一帖说过，对象变量其实是一个引用，它们的值是指向对象所在的内存地址，而不是对象本身。a 和 b 都使用了 new 操作符，意味着将在内存中产生两个内容为"foo"的字符串，既然是“两个”，它们自然位于不同的内存地址。a 和 b 的值其实是两个不同的内存地址的值，所以使用"=="操作符，结果会是 false。诚然，a 和 b 所指的对象，它们的内容都是"foo"，应该是“相等”，但是==操作符并不涉及到对象内容的比较。

对象内容的比较，正是 equals 方法做的事。

看一下 Object 对象的 equals 方法是如何实现的：

boolean equals(Object o){

return this==o;

}

Object 对象默认使用了==操作符。所以如果你自创的类没有覆盖 equals 方法，那你的类使用 equals 和使用==会得到同样的结果。同样也可以看出，Object 的 equals 方法没有达到 equals 方法应该达到的目标：比较两个对象内容是否相等。因为答案应该由类的创建者决定，所以 Object 把这个任务留给了类的创建者。

看一下一个极端的类：

Class Monster{

private String content;

...

boolean equals(Object another){ return true;}

}

我覆盖了 equals 方法。这个实现会导致无论 Monster 实例内容如何，它们之间的比较永远返回 true。

所以当你是用 equals 方法判断对象的内容是否相等，请不要想当然。因为可能你认为相等，而这个类的作者不这样认为，而类的 equals 方法的实现是由他掌握的。如果你需要使用 equals 方法，或者使用任何基于散列码的集合（HashSet,HashMap,HashTable），请察看一下 java doc 以确认这个类的 equals 逻辑是如何实现的。

问题三：String 到底变了没有？

没有。因为 String 被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。请看下列代码：

```
String s = "Hello";

s = s + " world!";
```

s 所指向的对象是否改变了呢？从本系列第一篇的结论很容易导出这个结论。我们来看看发生了什么事情。在这段代码中，s 原先指向一个 String 对象，内容是"Hello"，然后我们对 s 进行了+操作，那么 s 所指向的那个对象是否发生了改变呢？答案是没有。这时，s 不指向原来那个对象了，而指向了另一个 String 对象，内容为"Hello world!"，原来那个对象还存在于内存之中，只是 s 这个引用变量不再指向它了。

通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用 String 来代表字符串的话会引起很大的内存开销。因为 String 对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个 String 对象来表示。这时，应该考虑使用 StringBuffer 类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。并且，这两种类的对象转换十分容易。

同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都 new 一个 String。例如我们要在构造器中对一个名叫 s 的 String 引用变量进行初始化，把它设置为初始值，应当这样做：

```
public class Demo {

private String s;

...

public Demo {

s = "Initial Value";

}

...
```

}

而非

s = new String("Initial Value");

后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为 String 对象不可改变，所以对于内容相同的字符串，只要一个 String 对象来表示就可以了。也就说，多次调用上面的构造器创建多个对象，他们的 String 类型属性 s 都指向同一个对象。

上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java 认为它们代表同一个 String 对象。而用关键字 new 调用构造器，总是会创建一个新的对象，无论内容是否相同。

至于为什么要把 String 类设计成不可变类，是它的用途决定的。其实不只 String，很多 Java 标准类库中的类都是不可变的。在开发一个系统的时候，我们有时候也需要设计不可变类，来传递一组相关的值，这也是面向对象思想的体现。不可变类有一些优点，比如因为它的对象是只读的，所以多线程并发访问也不会有任何问题。当然也有一些缺点，比如每个不同的状态都要一个对象来代表，可能会造成性能上的问题。所以 Java 标准类库还提供了一个可变版本，即 StringBuffer。

问题四：final 关键字到底修饰了什么？

final 使得被修饰的变量"不变"，但是由于对象型变量的本质是“引用”，使得“不变”也有了两种含义：引用本身的不变，和引用指向的对象不变。

引用本身的不变：

final StringBuffer a=new StringBuffer("immutable");

final StringBuffer b=new StringBuffer("not immutable");

a=b;//编译期错误

引用指向的对象不变：

final StringBuffer a=new StringBuffer("immutable");

a.append(" broken!"); //编译通过

可见，final 只对引用的“值”(也即它所指向的那个对象的内存地址)有效，它迫使引用只能指向初始指向的那个对象，改变它的指向会导致编译期错误。至于它所指向的对象的变化，final 是不负责的。这很类似==操作符：==操作符只负责引用的“值”相等，至于这个地址所指向的对象内容是否相等，==操作符是不管的。

理解 final 问题有很重要的含义。许多程序漏洞都基于此----final 只能保证引用永远指向固定对象，不能保证那个对象的状态不变。在多线程的操作中,一个对象会被多个线程共享或修改，一个线程对对象无意识的修改可能会导致另一个使用此对象的线程崩溃。一个错误的解决方法就是在此对象新建的时候把它声明为 final，意图使得它“永远不变”。其实那是徒劳的。

问题五：到底要怎么样初始化！

本问题讨论变量的初始化，所以先来看一下 Java 中有哪些种类的变量。

1. 类的属性，或者叫值域

2. 方法里的局部变量

3. 方法的参数

对于第一种变量，Java 虚拟机会自动进行初始化。如果给出了初始值，则初始化为该初始值。如果没有给出，则把它初始化为该类型变量的默认初始值。

int 类型变量默认初始值为 0

float 类型变量默认初始值为 0.0f

double 类型变量默认初始值为 0.0

boolean 类型变量默认初始值为 false

char 类型变量默认初始值为 0(ASCII 码)

long 类型变量默认初始值为 0

所有对象引用类型变量默认初始值为 null，即不指向任何对象。注意数组本身也是对象，所以没有初始化的数组引用在自动初始化后其值也是 null。

对于两种不同的类属性，static 属性与 instance 属性，初始化的时机是不同的。instance 属性在创建实例的时候初始化，static 属性在类加载，也就是第一次用到这个类的时候初始化，对于后来的实例的创建，不再次进行初始化。这个问题会在以后的系列中进行详细讨论。

对于第二种变量，必须明确地进行初始化。如果再没有初始化之前就试图使用它，编译器会抗议。如果初始化的语句在 try 块中或 if 块中，也必须要让它在第一次使用前一定能够得到赋值。也就是说，把初始化语句放在只有 if 块的条件判断语句中编译器也会抗议，因为执行的时候可能不符合 if 后面的判断条件，如此一来初始化语句就不会被执行了，这就违反了局部变量使用前必须初始化的规定。但如果在 else 块中也有初始化语句，就可以通过编译，因为无论如何，总有至少一条初始化语句会被执行，不

会发生使用前未被初始化的事情。对于 try-catch 也是一样，如果只有在 try 块里才有初始化语句，编译部通过。如果在 catch 或 finally 里也有，则可以通过编译。总之，要保证局部变量在使用之前一定被初始化了。所以，一个好的做法是在声明他们的时候就初始化他们，如果不知道要出事化成什么值好，就用上面的默认值吧！

其实第三种变量和第二种本质上是一样的，都是方法中的局部变量。只不过作为参数，肯定是被初始化过的，传入的值就是初始值，所以不需要初始化。

问题六：instanceof 是什么东东？

instanceof 是 Java 的一个二元操作符，和==，>，<是同一类东东。由于它是由字母组成的，所以也是 Java 的保留关键字。它的作用是测试它左边的对象是否是它右边的类的实例，返回 boolean 类型的数据。举个例子：

String s = "I AM an Object!";

boolean isObject = s instanceof Object;

我们声明了一个 String 对象引用，指向一个 String 对象，然后用 instancof 来测试它所指向的对象是否是 Object 类的一个实例，显然，这是真的，所以返回 true，也就是 isObject 的值为 True。

instanceof 有一些用处。比如我们写了一个处理账单的系统，其中有这样三个类：

public class Bill {//省略细节}

public class PhoneBill extends Bill {//省略细节}

public class GasBill extends Bill {//省略细节}

在处理程序里有一个方法，接受一个 Bill 类型的对象，计算金额。假设两种账单计算方法不同，而传入的 Bill 对象可能是两种中的任何一种，所以要用 instanceof 来判断：

public double calculate(Bill bill) {

if (bill instanceof PhoneBill) {

//计算电话账单

}

if (bill instanceof GasBill) {

//计算燃气账单

}

…

}

这样就可以用一个方法处理两种子类。

然而，这种做法通常被认为是没有好好利用面向对象中的多态性。其实上面的功能要求用方法重载完全可以实现，这是面向对象变成应有的做法，避免回到结构化编程模式。只要提供两个名字和返回值都相同，接受参数类型不同的方法就可以了：

public double calculate(PhoneBill bill) {

//计算电话账单

}

public double calculate(GasBill bill) {

//计算燃气账单

}

所以，使用 instanceof 在绝大多数情况下并不是推荐的做法，应当好好利用多态。

java 方向及学习方法

java 分成 J2ME（移动应用开发），J2SE（桌面应用开发），J2EE(Web 企业级应用)，所以 java 并不是单机版的，只是面向对象语言。建议如果学习 java 体系的话可以这样去学习：

*第一阶段：Java 基础，包括 java 语法，面向对象特征，常见 API，集合框架；

*第二阶段：java 界面编程，包括 AWT，事件机制，SWING，这个部分也可以跳过，用的时候再看都能来及；

*第三阶段：java API：输入输出，多线程，网络编程，反射注解等，java 的精华部分；

*第四阶段：数据库 SQL 基础，包括增删改查操作以及多表查询；

*第五阶段：JDBC 编程：包括 JDBC 原理，JDBC 连接库，JDBC API，虽然现在 Hibernate 比 JDBC 要方便许多，但是 JDBC 技术仍然在使用，JDBC 思想尤为重要；

*第六阶段：JDBC 深入理解高级特性：包括数据库连接池，存储过程，触发器，CRM 思想；

*第七阶段：HTML 语言学习，包括 HTML 标签，表单标签以及 CSS，这是 Web 应用开发的基础；

*第八阶段：JavaScript 脚本语言，包括 javaScript 语法和对象，就这两个方面的内容；

*第九阶段：DOM 编程，包括 DOM 原理，常用的 DOM 元素以及比较重要的 DOM 编程思想；

*第十阶段：Servlet 开发，从此开始踏入 java 开发的重要一步，包括 XML，Tomcat 服务器的安装使用操作，HTTP 协议简单理解，Servlet API 等，这个是 java web 开发的基础。

*第十一阶段：JSP 开发：JSP 语法和标签，自定义标签，EL,JSTL 库了解以及 MVC 三层架构的设计模式理念；

*第十二阶段：AJAX 开发：AJAX 原理，请求响应处理，AJAX 开发库；

*第十三阶段：轻量级框架，三大框架之一 Struts 框架的学习，自此踏入 java web 开发的精华部分，包括 Struts 体系架构，各种组件，标签库和扩展性的学习；

*第十四阶段：Hibernate 框架学习，三大框架之一，包括检索映射技术，多表查询技术，缓存技术以及性能方面的优化；

*第十五阶段：Spring 框架的学习，三大框架之一，包括了 IOC,AOP,DataSource，事务，SSH 集成以及 JPA 集成；

*最后呢，还有些 java 的技术，包括 EJB3.0 等，可以选择学习，与三大轻量级框架相比，EJB 就是当之无愧的重量级了。

```
string a="aa"+"bb";

stringbuilder sb=new stringbuilder();

sb.append("aa");

sb.append("bb");
```

这两种在内存操作是不同的,第一种内存中有三个 string(分别为"aa","bb","aabb") ,第二种有三个("aabb")

第一种内存中有三个 string(分别为"aa","bb","aabb") ,第二种也有三个("aa","bb","aabb\0..."(具体长度根据 StringBuilder.Capacity))

区别在于:

string s="aa"; s +="bb"; 则内存中存在了 aa bb aabb 三个

stringbuilder ss="aa"; ss.Append("bb")。则内存中只有两个 "bb" "aabb" 此处少了一个"aa"，其实是因为 stringbuilder 是可变的，"aa"变成了"aabb"，而不是重新分配新空间。

String,Stringbuffer,Stringbuilder 的区别

写法 1:

string a="aa"+"bb";

写法 2：

stringbuilder sb=new stringbuilder();

sb.append("aa");

sb.append("bb");

1.如果这些语句被执行的次数很少，没有在循环里出现，那就用写法 1，简单明了。如果频繁地执行，比如出现在循环里，那可以用写法 2.

2.StringBuilder 的效率源于事先分配了内存，追加字符串时内存不够了，那还是需要再次分配内存，同样会降低效率，所以给 StringBuilder 指定的出示容量要合适，太小了，分配内存会频繁，太大了，浪费空间。StringBuilder 重新分配内存时是按照上次的容量加倍进行分配的。

String 字符串常量

StringBuffer 字符串变量（线程安全）

StringBuilder 字符串变量（非线程安全）

StringBuilder 被设计为与 StringBuffer 具有相同的操作接口。在单机非多线程(Multithread)的情况下使用 StringBuilder 会有较好的效率，因为 StringBuilder 没有处理同步(Synchronized)问题。StringBuffer 则会处理同步问题，如果 StringBuilder 会在多线程下被操作，则要改用 StringBuffer，让对象自行管理同步问题。

System.out.println(8+8+"88"+8+8);结果是 168888 而不是 168816，体会一下，不解释。

什么是 JVM

首先这里澄清两个概念：JVM 实例和 JVM 执行引擎实例，JVM 实例对应了一个独立运行的 Java 程序，而 JVM 执行引擎实例则对应了属于用户运行程序的线程；也就是 JVM 实例是进程级别，而执行引擎是线程级别的。

JVM 是什么？—JVM 的生命周期

JVM 实例的诞生：当启动一个 Java 程序时，一个 JVM 实例就产生了，任何一个拥有 publicstaticvoidmain(String[]args)函数的 class 都可以作为 JVM 实例运行的起点，既然如此，那么 JVM 如何知道是运行 classA 的 main 而不是运行 classB 的 main 呢？这就需要显式的告诉 JVM 类名，也就是我们平时运行 Java 程序命令的由来，如 JavaclassAhelloworld,这里 Java 是告诉 os 运行 SunJava2SDK 的 Java 虚拟机，而 classA 则指出了运行 JVM 所需要的类名。

JVM 实例的运行：main（）作为该程序初始线程的起点，任何其他线程均由该线程启动。JVM 内部有两种线程：守护线程和非守护线程，main（）属于非守护线程，守护线程通常由 JVM 自己使用，Java 程序也可以标明自己创建的线程是守护线程。JVM 实例的消亡：当程序中的所有非守护线程都终止时，JVM 才退出；若安全管理器允许，程序也可以使用 Runtime 类或者 System.exit()来退出。

JVM 是什么？—JVM 的体系结构

粗略分来，JVM 的内部体系结构分为三部分，分别是：类装载器（ClassLoader）子系统，运行时数据区，和执行引擎。下面将先介绍类装载器，然后是执行引擎，最后是运行时数据区

1，类装载器，顾名思义，就是用来装载.class 文件的。JVM 的两种类装载器包括：启动类装载器和用户自定义类装载器，启动类装载器是 JVM 实现的一部分，用户自定义类装载器则是 Java 程序的一部分，必须是 ClassLoader 类的子类。（下面所述情况是针对 SunJDK1.2）

动类装载器：只在系统类(JavaAPI 的类文件)的安装路径查找要装入的类

用户自定义类装载器：

系统类装载器：在 JVM 启动时创建，用来在 CLASSPATH 目录下查找要装入的类其他用户自定义类装载器：这里有必要先说一下 ClassLoader 类的几个方法，了解它们对于了解自定义类装载器如何装载.class 文件至关重要。

protectedfinalClassdefineClass(Stringname,bytedata[],intoffset,intlength)

protectedfinalClassdefineClass(Stringname,bytedata[],intoffset,intlength,ProtectionDomainprotectionDomain);protectedfinalClassfindSystemClass(Stringname)

protectedfinalvoidresolveClass(Classc)

defineClass 用来将二进制 class 文件（新类型）导入到方法区,也就是这里指的类是用户自定义的类（也就是负责装载类）

findSystemClass 通过类型的全限定名，先通过系统类装载器或者启动类装载器来装载，并返回 Class 对象。

ResolveClass:让类装载器进行连接动作（包括验证，分配内存初始化，将类型中的符号引用解析为直接引用），这里涉及到 Java 命名空间的问题，JVM 保证被一个类装载器

装载的类所引用的所有类都被这个类装载器装载，同一个类装载器装载的类之间可以相互访问，但是不同类装载器装载的类看不见对方，从而实现了有效的屏蔽。

2，执行引擎：它或者在执行字节码，或者执行本地方法

要说执行引擎，就不得不的指令集，每一条指令包含一个单字节的操作码，后面跟 0 个或者多个操作数。

（一）指令集以栈为设计中心，而非以寄存器为中心这种指令集设计如何满足 Java 体系的要求：

平台无关性：以栈为中心使得在只有很少 register 的机器上实现 Java 更便利 compiler 一般采用 stack 向连接优化器传递编译的中间结果，若指令集以 stack 为基础，则有利于运行时进行的优化工作与执行即时编译或者自适应优化的执行引擎结合，通俗的说就是使编译和运行用的数据结构统一，更有利于优化的开展。

网络移动性：class 文件的紧凑性。

安全性：指令集中绝大部分操作码都指明了操作的类型。（在装载的时候使用数据流分析期进行一次性验证，而非在执行每条指令的时候进行验证，有利于提高执行速度）。

（二）执行技术

主要的执行技术有:解释，即时编译，自适应优化、芯片级直接执行其中解释属于第一代 JVM，即时编译 JIT 属于第二代 JVM，自适应优化（目前 Sun 的 HotspotJVM 采用这种技术）则吸取第一代 JVM 和第二代 JVM 的经验，采用两者结合的方式

自适应优化：开始对所有的代码都采取解释执行的方式，并监视代码执行情况，然后对那些经常调用的方法启动一个后台线程，将其编译为本地代码，并进行仔细优化。若方法不再频繁使用，则取消编译过的代码，仍对其进行解释执行。

3，运行时数据区：主要包括：方法区，堆，Java 栈，PC 寄存器，本地方法栈

（1）方法区和堆由所有线程共享

堆：存放所有程序在运行时创建的对象

方法区：当 JVM 的类装载器加载.class 文件，并进行解析，把解析的类型信息放入方法区。

（2）Java 栈和 PC 寄存器由线程独享，在新线程创建时间里

（3）本地方法栈：存储本地方法调用的状态

上边总体介绍了运行时数据区的主要内容，下边进行详细介绍，要介绍数据区，就不得不说明 JVM 中的数据类型。

JVM 中的数据类型：JVM 中基本的数据单元是 word,而 word 的长度由 JVM 具体的实现者来决定

数据类型包括基本类型和引用类型，

（1）基本类型包括：数值类型(包括除 boolean 外的所有的 Java 基本数据类型)，boolean（在 JVM 中使用 int 来表示，0 表示 false，其他 int 值均表示 true）和 returnAddress（JVM 的内部类型，用来实现 finally 子句）。

（2）引用类型包括：数组类型，类类型，接口类型

前边讲述了 JVM 中数据的表示，下面让我们输入到 JVM 的数据区

首先来看方法区：

上边已经提到，方法区主要用来存储 JVM 从 class 文件中提取的类型信息，那么类型信息是如何存储的呢？众所周知，Java 使用的是大端序（big?endian:即低字节的数据存储在高位内存上，如对于 1234，12 是高位数据，34 为低位数据，则 Java 中的存储格式应该为 12 存在内存的低地址，34 存在内存的高地址，x86 中的存储格式与之相反）来存储数据，这实际上是在 class 文件中数据的存储格式，但是当数据倒入到方法区中时，JVM 可以以任何方式来存储它。

类型信息：包括 class 的全限定名，class 的直接父类，类类型还是接口类型，类的修饰符（public,等），所有直接父接口的列表，Class 对象提供了访问这些信息的窗口（可通过 Class.forName(“”)或 instance.getClass()获得），下面是 Class 的方法，相信大家看了会恍然大悟，getName(),getSuperClass(),isInterface(),getInterfaces(),getClassLoader();

static 变量作为类型信息的一部分保存

指向 ClassLoader 类的引用：在动态连接时装载该类中引用的其他类

指向 Class 类的引用：必然的，上边已述

该类型的常量池：包括直接常量（String，integer 和 floatpoint 常量）以及对其他类型、字段和方法的符号引用（注意：这里的常量池并不是普通意义上的存储常量的地方，这些符号引用可能是我们在编程中所接触到的变量），由于这些符号引用，使得常量池成为 Java 程序动态连接中至关重要的部分

字段信息：普通意义上的类型中声明的字段

方法信息：类型中各个方法的信息

编译期常量：指用 final 声明或者用编译时已知的值初始化的类变量

class 将所有的常量复制至其常量池或者其字节码流中。

方法表：一个数组，包括所有它的实例可能调用的实例方法的直接引用（包括从父类中继承来的）

除此之外，若某个类不是抽象和本地的，还要保存方法的字节码，操作数栈和该方法的栈帧，异常表。

举例：

```
classLava{
privateintspeed=5;
voidflow(){}
classVolcano{
publicstaticvoidmain(String[]args){
Lavalava=newLava();
```

```
lava.flow();

}

}
```

运行命令 JavaVolcano;

（1）JVM 找到 Volcano.class 倒入，并提取相应的类型信息到方法区。通过执行方法区中的字节码，JVM 执行 main（）方法，（执行时会一直保存指向 Vocano 类的常量池的指针）

（2）Main（）中第一条指令告诉 JVM 需为列在常量池第一项的类分配内存（此处再次说明了常量池并非只存储常量信息），然后 JVM 找到常量池的第一项，发现是对 Lava 类的符号引用，则检查方法区，看 Lava 类是否装载，结果是还未装载，则查找“Lava.class”,将类型信息写入方法区，并将方法区 Lava 类信息的指针来替换 Volcano 原常量池中的符号引用，即用直接引用来替换符号引用。

（3）JVM 看到 new 关键字，准备为 Lava 分配内存，根据 Volcano 的常量池的第一项找到 Lava 在方法区的位置，并分析需要多少对空间，确定后，在堆上分配空间，并将 speed 变量初始为 0，并将 lava 对象的引用压到栈中

（4）调用 lava 的 flow（）方法

好了，大致了解了方法区的内容后，让我们来看看堆

Java 对象的堆实现：

Java 对象主要由实例变量（包括自己所属的类和其父类声明的）以及指向方法区中类数据的指针，指向方法表的指针，对象锁（非必需），等待集合（非必需），GC 相关的数据（非必需）（主要视 GC 算法而定，如对于标记并清除算法，需要标记对象是否被引用，以及是否已调用 finalize（）方法）。

那么为什么 Java 对象中要有指向类数据的指针呢？我们从几个方面来考虑

首先：当程序中将一个对象引用转为另一个类型时，如何检查转换是否允许？需用到类数据

其次：动态绑定时，并不是需要引用类型，而是需要运行时类型，

这里的迷惑是：为什么类数据中保存的是实际类型，而非引用类型？这个问题先留下来，我想在后续的读书笔记中应该能明白

指向方法表的指针：这里和 C++的 VTBL 是类似的，有利于提高方法调用的效率

对象锁：用来实现多个线程对共享数据的互斥访问

等待集合：用来让多个线程为完成共同目标而协调功过。（注意 Object 类中的 wait(),notify(),notifyAll()方法）。

Java 数组的堆实现：数组也拥有一个和他们的类相关联的 Class 实例，具有相同 dimension 和 type 的数组是同一个类的实例。数组类名的表示：如[[LJava/lang/Object 表示 Object[][]，[I 表示 int[]，[[[B 表示 byte[][][]

至此，堆已大致介绍完毕，下面来介绍程序计数器和 Java 栈

程序计数器：为每个线程独有，在线程启动时创建，

若 thread 执行 Java 方法，则 PC 保存下一条执行指令的地址。

若 thread 执行 native 方法，则 Pc 的值为 undefined

Java 栈：Java 栈以帧为单位保存线程的运行状态，Java 栈只有两种操作，帧的压栈和出栈。

每个帧代表一个方法，Java 方法有两种返回方式，return 和抛出异常，两种方式都会导致该方法对应的帧出栈和释放内存。

帧的组成：局部变量区（包括方法参数和局部变量，对于 instance 方法，还要首先保存 this 类型，其中方法参数按照声明顺序严格放置，局部变量可以任意放置），操作数栈，帧数据区（用来帮助支持常量池的解析，正常方法返回和异常处理）。

本地方法栈：依赖于本地方法的实现，如某个 JVM 实现的本地方法借口使用 C 连接模型，则本地方法栈就是 C 栈，可以说某线程在调用本地方法时，就进入了一个不受 JVM 限制的领域，也就是 JVM 可以利用本地方法来动态扩展本身。

相信大家都明白 JVM 是什么了吧。

安装 JDK 后 JRE 与 JVM 联系浅谈

监视 JSP 中 JVM 可用内存

JDK、JRE、JVM 之间的关系

Java 之父：我们看中的并非 Java 语言，而是 JVM

Java 虚拟机（JVM）中的内存设置详解

fianl

有一种被称为 inline 的机制，它会使你在调用 final 方法时，直接将方法主体插入到调用处，而不是进行例行的方法调用，例如保存断点，压栈等，这样可能会使你的程序效率有所提高，然而当你的方法主体非常庞大时，或你在多处调用此方法，那么你的调用主体代码便会迅速膨胀，可能反而会影响效率，所以你要慎用 final 进行方法定义。

易记错

位运算

&：代表析；[1&0]=0

|：代表合；[1|0]=1

^：代表 XOR；不同=1

与逻辑运算的区别

位运算：

A	B	A|B	A&B	A^B
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

逻辑运算符只有 3 个，即：与 AND(&&)、或 OR(||)、非 NOT(!)。

位运算符只有 7 个，即：与 AND(&)、或 OR(|)、非 NOT(~)、异或 XOR(^)、左移 ShiftLeft(<<)、右移 ShiftRight(>>)，无符号右移（>>>）

逻辑运算：

exp1 ,exp2 是函数表达式。

exp1	exp2	exp1&&exp2
false	false	false //第一个已经是 false, 所以 exp2 表达式不会执行。
false	true	false //同上
true	false	false
true	true	true

exp1 exp2 exp1||exp2

false false false

false true true

true false true //第一个已经是 true ,所以 exp2 表达式不会执行

true true true //同上

子项目

应用篇

输入输出流

练习 1：通过以下代码，结合“输入输出流.docx”,对 IO 流有一定的概念和理解。

```
import java.io.*;
public class test2 {
	public static void main(String[] args) throws IOException
	{
		try
		{ //创建文件输入流对象
//			FileInputStream rf = new FileInputStream("1.txt");
			BufferedInputStream rf = new BufferedInputStream(System.in);
			int n=512;
			byte buffer[] = new byte[n];
//			while ((rf.read(buffer,0,n)!=-1) && (n>0)) //读取输入流
//			{
			rf.read(buffer,0,n); //如果要上面的 while 循环，就去掉本行
```

```
                        System.out.print(new String(buffer));
//                      }
//                      System.out.println();
                        rf.close(); //关闭输入流
                }
                catch (IOException ioe)
                {
                        System.out.println(ioe);
                }
                catch (Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

练习 2：

```
import java.io.*;
public class test2 {
        /**
         * @param args
         * @throws IOException
         */
        public static void main(String[] args) throws IOException
```

```
    {
        try
        { //创建文件输入流对象
            int n =512;
            int count;
            byte[] buf = new byte[n];
            FileInputStream dq = new FileInputStream("1.txt");
            count = dq.read(buf);
            System.out.print(new String(buf));
            FileOutputStream bc = new FileOutputStream("4.txt");
            bc.write(buf,0,count);
            bc.close();
            dq.close();
        }
        catch (IOException ioe)
        {
            System.out.println(ioe);
        }
        catch (Exception e)
        {
            System.out.println(e);
        }
    }

}
```

通过 Eclipse 的说明文件来理解：

```
import java.io.*;

public class test1 {

	/**

	 * @param args

	 * @throws IOException

	 */

	public static void main(String[] args) throws IOException

	{

		PrintStream console = System.out;

		BufferedInputStream in = new BufferedInputStream(new
FileInputStream("1.txt"));

		PrintStream out = new PrintStream(new BufferedOutputStream(new
FileOutputStream("3.txt")));

		System.setIn(in);

		System.setOut(out);

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

		String s;

		while((s=br.readLine())!=null)

			System.out.println(s);

		out.close();

		System.setOut(console);

	}

}
```

练习 3：

```
import java.io.*;
import java.util.Date;
import java.text.SimpleDateFormat;
public class test3 {

    /**
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException
    {
        String fname = "1.txt"; //待复制的文件名
        String childdir = "backup"; //子目录名
        new test3().update(fname,childdir);
    }
    public void update(String fname,String childdir) throws IOException
    {
        File f1,f2,child;
        f1 = new File(fname); //当前目录中创建文件对象 f1
        child = new File(childdir); //当前目录中创建文件对象 child
        if (f1.exists())
        {
```

```
                if (!child.exists()) //child 不存在时创建子目录
                        child.mkdir();
                f2 = new File(child,fname); //在子目录 child 中创建文件 f2
                if (!f2.exists() || //f2 不存在时或存在但日期较早时
                                f2.exists()&&(f1.lastModified() > f2.lastModified()))
                        copy(f1,f2); //复制
                getinfo(f1);
                getinfo(child);
        }
        else
                System.out.println(f1.getName()+" file not found!");
}
public void copy(File f1,File f2) throws IOException
{ //创建文件输入流对象
        FileInputStream rf = new FileInputStream(f1);
        FileOutputStream wf = new FileOutputStream(f2);
        //创建文件输出流对象
        int count,n=512;
        byte buffer[] = new byte[n];
        count = rf.read(buffer,0,n); //读取输入流
        while (count != -1)
        {
                wf.write(buffer,0,count); //写入输出流
                count = rf.read(buffer,0,n);
        }
```

```
                System.out.println("CopyFile "+f2.getName()+" !");
                rf.close(); //关闭输入流
                wf.close(); //关闭输出流
        }
        public static void getinfo(File f1) throws IOException
        {
                SimpleDateFormat sdf;
                sdf= new SimpleDateFormat("yyyy 年 MM 月 dd 日 hh 时 mm 分");
                if (f1.isFile())
                        System.out.println("<FILE>\t"+f1.getAbsolutePath()+"\t"+
                                        f1.length()+"\t"+sdf.format(new
Date(f1.lastModified())));
                else
                {
                        System.out.println("\t"+f1.getAbsolutePath());
                        File[] files = f1.listFiles();
                        for (int i=0;i<files.length;i++)
                                getinfo(files[i]);
                }
        }

}
```

线程

JavaScript

基本的语法

alert:弹出窗口

document.write():在网页上显示文本

String 属性

string.anchor(anchorName)

big(),small(),fixed(),Italics()粗体,fontsize()

fontcolor(color)

string=stringValue.toLowerCase,toUpperCase

string.indexOf("str",fromindex)//搜索字符

string.substring(start,end)

Math 属性

常量：PI,E

abs()绝对值，sin(),cos();asin(),acos();tan(),atan();

round()四舍五入，sqrt()平方根，pow(base,exponent)基于几次放的根

外置对象

浏览器内置对象

History

onClick="history.back()"

onClick="history.forward()"

onClick="history.length()"本窗口已打开了多少网页

Document

窗体对象

document.myform.button.value

document.myform.text.value//网页修改 name 值呢？

select

```
<html>
 <head>
   <script language="javascript">
    function test(a) {
     var mystr = ""
     for ( var i=0; i<a.options.length; i++) {
       if ( a.options[i].selected)
       mystr+=a.options[i].text+"\n"
     }
       alert(mystr)
   }
   </script>
 </head>
 <body>
   <form method="POST" action="" name="myform">
     <select size="4" name="D1" multiple>
     <option>长城</option>
     <option selected>故宫</option>
     <option>北戴河</option>
     <option>漓江</option>
     <option>西湖</option>
```

```
    </select>
    <input type="button" value="你想去的是" name="B1" onclick="test(myform.D1)">
   </form>
 </body>
</html>
```

checkbox

```
<HTML>
 <HEAD>
        <Script Language="JavaScript">
                function test()
                {
                        document.myform.checkbox1.click();
                        document.myform.checkbox2.click();
                        document.myform.checkbox3.click();
                        document.myform.checkbox4.click();
                }
         function checkall()
                {
                        if(document.myform.checkbox6.checked==true)
                        {
                                document.myform.checkbox1.checked=true;
                                document.myform.checkbox2.checked=true;
                                document.myform.checkbox3.checked=true;
                                document.myform.checkbox4.checked=true;
                        }
```

```
                else
                {
                        document.myform.checkbox1.checked=false;
                        document.myform.checkbox2.checked=false;
                        document.myform.checkbox3.checked=false;
                        document.myform.checkbox4.checked=false;
                }
        }
    </Script>
 </HEAD>
 <BODY>
  <form name="myform">
    <INPUT type="checkbox" name="checkbox1">长城<br>
    <INPUT type="checkbox" name="checkbox2">故宫<br>
    <INPUT type="checkbox" name="checkbox3">北戴河<br>
    <INPUT type="checkbox" name="checkbox4">西湖<br>
    <INPUT type="checkbox" value="全部反选" name="checkbox5" onClick="test()">全部反选
    <INPUT type="checkbox" value="全选" name="checkbox6" onClick="checkall()">全选
  </form>
 </BODY>
</HTML>
```

radio

```
<HTML>
 <HEAD>
```

```
        <Script Language="JavaScript">
                function whichRadio()
                {
                        for(var i=0;i<5;i++)
                        {
                                if(document.myform.myradio[i].checked==true)
                                        return document.myform.myradio[i].value;
                        }
                        return "没有想去的地方";
                }
        </Script>
 </HEAD>
 <BODY>
  <form name="myform">
        <INPUT type="radio" name="myradio" value="长城">长城<br>
        <INPUT type="radio" name="myradio" value="故宫">故宫<br>
        <INPUT type="radio" name="myradio" value="北戴河">北戴河<br>
        <INPUT type="radio" name="myradio" value="漓江">漓江<br>
        <INPUT type="radio" name="myradio" value="西湖">西湖<br>
  </form>
  <form name="btnForm">
        <INPUT type="button" value="显示长度" name="displayBtn" onClick="alert
(myform.myradio.length)"><br><br>
        <INPUT type="button" value="谁被选中" name="whichBtn"
onClick="alert(whichRadio ())"><br>
  </form>
```

</BODY>

</HTML>

子项目

Java 大纲

初识

Java 是什么

Java 是什么，Java 的介绍。

一、前言

『Java』从 1995 年的暑假开始在计算机业界就受到了高度注意，特别是在 Internet 和多 媒体(Multimedia)相关产品类方面。Java 为何有如此这么大的魅力？人作如此的比喻： Java 在全球资讯网(World Wide Web, WWW)地位就如同电子表格(Spreadsheet)与个人计 算机(PC)的关系。那 Java 究竟有那些特色呢？

Java 是一种软件技术

是一种由美国 SUN 计算机公司(Sun Microsystems, Inc.)所研究而成的语言

是一种为 Internet 发展的计算机语言

是一种使网页(Web Page)产生生动活泼画面的语言

是一种使网页(Web Page)由静态(Static)转变为动态(Dynamic)的语言

是一种语言，用以产生「小应用程序(Applet(s))」

是一种简化的 C++语言 是一种安全的语言，具有阻绝计算机病毒传输的功能

是一种将安全性(Security)列为第一优先考虑的语言

是一种使用者不需花费很多时间学习的语言

是一种突破用户端机器环境和 CPU 结构的语言

是一种「写一次，即可在任何机器上执行(Write OnceRun Anywhere)」的语言是有史以来，第一套允 使用者将应用程序(Applications)通过 Internet 从远端的服务器(Remote Server)传输到本地端的机器 上(LocalMachine)并执行

是一种应用程序提供者不需要知道使用者的计算机硬件(如：Sun, Intel, 或 MAC 等)与软件(如：SW- UNIX, MAC O/S, Windows, 或 NT 等)环境的语言(Kestenbaum, 1995)。

下面将依序地介绍 Java，首先是 Java 的发展历史与 Java 语言介绍，其次依序是 Java Applet 和 HotJava 的简单介绍。

二、Java FAQ

下面以问答的方式来说明 Java 的发展历史与其背景(下列内容整理自 Java FAQ list and Tutorial 和 The Java Language: A White Paper，读者若欲深 入了解，请自行参阅原文)：

1.Java 何时开始发展？(When)

最早大概可追溯至 1991 年四月份，Sun 的绿色计划(Green Project)开始着手于发展消费性电子产品(Consumer Electronics)，所使用的语言是 C、C++、及 Oak (为 Java 语 言的前身)，后因语言本身和市场的问题， 使得消费性电子产品的发展无法达到当初 预期的目标，再加上网络的兴起， 绿色计划也因此而改变发展的方向，这已是 1994 年了。

为何称之为 Java？(Why) "Java"是美国 SUN 计算机公司 Java 发展小组历经无数次的激烈讨论之后才被选择出。 生动(Liveliness)、动画(Animation)、速度(Speed)、交互性(Interactivity)为当 初选择名字时所欲表达出的特色。"Java"是在无数的建议中脱颖而出的，而"Java" 不是由几个单字的首字所组成， 而是从许多程序设计师钟爱的热腾腾、香浓咖啡中 产生灵感的。

谁开发了 Java？(Who) Java 是美国 SUN 计算机公司 Java 发展小组开发的，早期的成员(绿色工程)是 Patrick Naughton, James Gosling, & Mike Sheridan，而现在大家较为熟悉的成员是 James Gosling。

在那里开发了 Java？(Where)

也就是问 Java 的出生地？答案是美国。

如何可以找到所需的 Java 信息？(How to)

在网路上，您可以连到 Sun 公司的 Java WWW 网站，URL 是 http://java.sun.com/，或是 http://www.javasoft.com/。在那里几乎可以找到您所需要的所有 Java 信息，但是语言多少是一个障碍， 至少对某些人而言；没关系，目前国内已有很多个网站提供中文 Java 信息。在清华和中科院的 FTP 站点上有不少有关资料。想象以后应会有更多的站点提供相关信息。

如何才能看到 Java 的效果？(How Do I)

首先您需要有含有 Java 解释器的浏览器(Browser)，例如：Netscpae 公司的 Netscape Navigator 2.0 以上或是 Sun 公司的 HotJava 浏览器，对个人计算机使用者而言，操作 系统需是 Windows 95 或是 Windows NT。

Java 是因为撰写 C++语言程序时的困难而研制开的，起先，只是一个消费性电子产品大计划中的一部份，C++语言是当初被考虑采用的，但从一开始的编译问题一直到最后的一连串问题迫使得放弃 C++语言，而有 Java 语言的产生。Sun 是要 Java 成为一个简 单(Simple)、面向对象的(Object Oriented)、 分布式的(Distributed)、解释的(Interpreted)、健壮的(Robust)、安全的(Secure)、 结构中立的(Architecture Neutral)、可移植的(Portable)、高效能的(High Performance)、多线程的(Multithreaded)、动态的(Dynamic)的程序语言(摘译自 TheJava Language: A White Paper, 1995)。

在 Sun 的 Java 语言白皮书中明白地说明上述 Java 语言的技巧。若以木工为比喻，一个面 向对象的木工，他(她)最主要的重点是即将要做的木椅子，其次才是所需要的工具； 反之；一个以非面向对象的木工，他(她)所关心的只是工具。最近的即插即用(Plug and Play)亦是面向对象设计的重点。 分布式的(Distributed)：Java 有一个很周全的程薪录 JAVA 介绍 。

Java 为何有如此这么大的魅力？人作如此的比喻： Java 在全球资讯网(World Wide Web, WWW)地位就如同电子表格(Spreadsheet)与个人计 算机 TTP 和 FTP 等 TCP/IP 通讯协定相配合。Java 应用程序(Applications) 能在网路上开启及连结使用物件，就如同透过 URLs 连结使用一个本地文件系统(Local File System)。 健壮的(Robust)：由 Java 所编写出的程序能在多种情况下执行而具有其稳定性。Java 与 C/C++最大不同点是 Java 有一个指针模型(Pointer Model)来排除内存被覆盖(Overwriting Memory)和毁损数据(Corrupting Data)的可能性。

安全的(Secure)：Java 是被设计用于网络及分布式的环境中，安全性自必是一个很 重要的考虑。Java 拥有数个阶层的互锁(Interlocking)保护措施，能有效地防止病 毒的侵入和破坏行为的发生。

结构中立的(Architecture Neutral)：一般而言，网络是由很多不同机型的机器所 组合而成的，CPU 和作业系统体系结构均有所不同；因此，如何使一个应用程序可以 在每一种机器上执行，是一个难题。所幸，Java 的编译器产生一种结构中立的目标 文件格式(Object File Format)；这使得编译码得以在很多种处理器中执行。

可移植的(Portable)：原始资料型式的大小是被指定的，例如"float"一直是表示一 个 32 位元 IEEE 754 浮点运算数字，因绝大多数的 CPU 都具有此共同特征。程序库属于 系统的一部份，它定义了一些可移植的程序接口，Java 本身具备有很好的可移植性。

解释的(Interpreted)：Java 解释器能直接地在任何机器上执行 Java 位元码(Bytecodes)，因此在进行程序连结时，时间的节省，这对于缩短程序的开发过程，有极大的帮助。

高效能的(High Performance)：Java 位元码迅速地能被转换成机器码(Machine Code)，从位元码转换到机器码的效能几乎与 C 与 C++没有分别。

多线程的(Multi threaded)：Java 语言具有多线程的功能，这对于交互回应能力及 即时执行行为是有帮助的。

动态的(Dynamic)：Java 比 C 或 C++语言更具有动态性，更能适应时刻在变的环境，Java 不会因程序库的更新，而必须重新编译程序。

此外，Hank Shiffman (Making Sense of Java)亦针一般对 Java 的错误看法及观 念提出他的说明，特在此摘译如下：

"Java 是一种编写 Web Pages 的一种语言，就如同 HTML 和 VRML 一样" 事实上，Java 并不像是 HTML 此一类的描述语言(Description Language)，而是一种 编程语言(Programming Language)。描述语言标明内容和位置，而编程语言描述一 种产生结果的过程。

2. "Java 语言容易学习和使用，不像 C、C++和其它程序语言"

Java 是一种编程语言。Java 容易学吗？Java 或许是比 C 或 C++容易学，但仍是一种编程语言，而不是一种描述语言。

3. "Java 码是可移植的，但 C 及 C++不是"

Java 原代码(Source Code)是比 C 语言来得可移植一点，差别在于 Java 的目标码。Java 码在一种机器上进行编译，而能在所有的机器上执行， 只要那部机器上有 Java 解释器。

4. "Java 能被拓展而在机器上执行任何事情"

理论上，Java Applet (Java 小应用程序)能做任何事情，如模拟 3D VRML 模型、播放电影、产生音频....等。但事实上，一个小应用程序(Applet)仅能在那一页上被执行，而无法在那一页之外执行。同时，Java 亦受限于程序库的功能。

5. "Java 是适合于建立大型的应用程序"

如果 Java 适合于大型程序，则 Java 就不适合应用于 Web 浏览器了。第一个商业 性的 Java Applets (Applix's Java-Based Spreadsheet) 并不是全然使用 Java， 它只使用 Java 作为用户接口，而所有的处理工作， 是用 CGI 码。

6. "Java 是解释执行的，Basic 是解释执行的，因此 Java=Basic"

虽然 Java 的确是使用解释器，但事实上，Java 则与 C 或 C++等完全编译语言较为相近，但与 Basic 或 APL 等完全解译语言较不相近。

7. "Java 删除了 CGI 命令稿(Scripts)和程序的需求"

Java Applets 将会取代部份 CGI 的用途。在有些情况，Java Applets 能够取代一些服务器端代码(Server-Side Code)，但大多数的情况，基于安全性理由或是效 能的考虑，Java 仍无法全然取代 CGI Scripts。

8. "Netscape's JavaScript 是与 Java 有相关"

除了名称之外，Java 和 JavaScript 是有一点点相关。JavaScript 是一种命令稿语 言，是可以在 HTML 页中使用。Java 码并未出现在 HTML 中，而在 HTML 中通过一个链 结来链结编译码组。Java 和 JavaScript 之间的关系就如同 C 语言和 C Shell 一般。

Java 折行规则

一些企业在招聘程序员的时候，总会特意提出一个要求，即“要求具有良好的编码规范”。确实现在程序开发人员已经不在是单枪匹马的单干，而是讲究团体作战。此时就要求团队内的乘员都可能够恪守代码的编写规范，这对于乘员之间共享代码、排错等作业都具有非常现实的意义。这在 Java 语言中当然也不例外。笔者借这次机会，就跟大家分享一下 Java 源代码的折行规则。虽然这基本不涉及到功能层面的内容，但是对于提高代码的阅读性却有不可替代的作用。

一、代码的最大长度。

虽然在 Java 的编译器中对于代码的最大长度没有硬性的规定。但是如果代码的长度太长，超过了编译器的最当行宽，显然阅读起来比较麻烦。为此根据笔者的经验，通常情况下 Java 源代码的行长度不应该大于 80 个字符。如果超过这个长度的话，在一些开发工具和编辑器上就无法很好的显示。如需要通过滚动条来显示后面部分的代码。当其他项目成员阅读这超长的代码时，就会看得眼花缭乱。当人的温饱问题解决了之后，就需要开始注意美观方面的问题。所以程序开发人员在开发应用程序的时候，要尽量避免书写长的代码。如果代码的每行长度确实需要超过 80 个字符的话(最好将每行代码的长度控制在 70 个字符左右)，那么就需要对代码进行分行。

二、在恰当的地方对代码进行分行。

笔者建议将 Java 源代码每行的长度控制在 70 个字符、最大不超过 80 个字符。当超过这个字符长度的时候，开发人员就需要考虑在恰当的地方对他们进行分行处理。不过这个分行也不是说开发人员想在哪里进行分行就在哪里进行分行。这个分行是有一定技巧的。虽然这些技巧大部分并不是强制性的规定，但是都是一些专家们的经验总结，可以提高代码的阅读性。为此笔者希望各位程序开发能够严格的遵守。

技巧一：高层折行优于低层折行。

这个技巧是说，在考虑对代码进行折行处理的时候，需要注意代码的层次性。如某段代码涉及到混合四则运算，而四则运算又有明显的运算顺序，此时对代码进行折行时就最好能够在四则运算的关键顺序上进行折行处理。如现在有如下的一段代码：

```
Mynum=mynum1*(mynum1+mynue2+mynum3-mynum4)+8*mynum5
```

如果要对这段代码进行折行的话，该在哪个地方加入一个折行符号呢?如果是笔者处理的话，笔者会按如下的格式对代码进行折行处理。

```
Mynum=mynum1*(mynum1+mynue2+mynum3-mynum4)

+8*mynum5
```

这主要是根据四则运算的运算层次来进行折行的。显然，*符号的优先级要比+符号要高。所以在+号前面对其进行折行处理，那么就可以一目了然的反应出代码的运算层次。可以大幅度的提高代码的阅读性。所以代码折行的第一个技巧就是高层折行优于低层折行。如此的话，可以使得应用程序的结构代码更加的清晰，更容易被团队成员所理解。

技巧二：在运算符前面进行折行处理。

其实在如上的折行技巧中，还隐藏着一个规则，即在运算符之前进行折行。如上例所示，笔者就是在+号前面进行折行，而不是在+号后面进行折行处理。这主要也是考虑到代码的可读性。如上面这个例子中，如果在+号后面进行折行处理的话，则下一行就会给人一种凭空多出来的感觉，显得代码很不连贯。跟这个规则类似，如果在折行处理的时候遇到逗号时，那么最好能够在逗号后面进行折行。如在一个方法中，需要传入 5 个参数。此时如果代码行比较长，那么就需要在几个参数之间进行折行处理。此时最理想的折行位置，就是在某个参数的逗号后面。注意，使在逗号后面进行折行，而不是在逗号前面。因为一个参数一个逗号是匹对的。而如果一个逗号加一个参数，则让人看起来很不舒服。

技巧三：这行代码的对起方式。

当不得已对代码进行折行处理时，下一行的代码应该与其同等级的代码行左对齐。如上例所示，在+号前面将某一段四则运算公式进行折行处理的时候，其+号符号已经采取缩进处理。其缩进后的效果就是要与其同等级的代码行左对齐。如此的话，明眼人一看就知道这段代码采取过折行处理;而且跟上一行代码的层次关系。当一段代码被分割成三行甚至跟更多行数的时候，这个规则会非常的有用。如果能够严格遵守这个规则，即使将代码分割成多行，看起来也不会觉得那么混乱。反而给人一种比较有层次的感觉。

另外在采取缩进处理的时候，可以利用 Tab 键来提高缩进处理的效率。因为直接按空格的话，有可能空格字符数量不一致，会让人觉得层次不起，产生比较大的混乱。一般情况下，当代码行两侧距离页边的距离比较大，看其来不怎么舒服时，可以在代码行中通过插入 TAB 键(会在代码行中连续插入 8 个字符)来提高代码的阅读性，让代码的缩进实现统一。

技巧四：为变量寻找一个合适的位置。

在编写应用程序时，尽量将变量声明放置在一个代码块的开始处，也就是说{}花括号的开始位置。虽然说可以在需要使用变量的时候再对其进行声明。但是笔者不同义如此操作。因为如果在的呢感到使用变量的时候再来声明变量时，会降低代码的可读性。同理，也需要避免低层声明与高层声明重复，这样会引起代码混乱并可能引发程序功能性错误。而且这种错误在后续的排错中很难被发现。为此要在应用程序开发的时候就要尽量避免这种错误。

在声明变量的时候，有时候可能代码很简单，生命变量的行总共加起来也不会超过十个字符。此时能否把多个变量的声明写在同一行呢?从就技术上来说，这是可行的。也就是说，Java 编译器允许将多个变量定义在同一行上。但是从阅读性上来说，这并不是很好的做法。笔者的建议是，即使变量定义再简单，或者变量比较多，也最好分行进行变量的声明。也就是说，一行声明一个变量。这可以提高代码的可阅读性。而且有时候往往需要对变量加一个注释说明变量的用途，如果以行定义一个变量，添加行注释也相对简单许多。

总之，以上的这些折行的规则基本上不会影响到代码的运行。但是，对于代码的维护与后续的排错、升级、二次开发等等具有不可忽视的作用。而且现在基本上应用程序开发式团队开发，故大家都遵守同样的代码编写规范是非常重要的。笔者在开发一个应用程序的时候，事先都会花一定的时间，跟项目成员强调这些折行的规则。目的只有一个，就是提高代码的可读性，便于后续代码的共享与维护。毕竟后续面对这些代码的，并不是客户，而是我们自己。我们程序员在编写代码的时候，不能够搬起石头砸自己的脚。笔者认为，现在一个合格的程序人员，不仅技术功底上要过得硬，而且还必须要遵守这些“无形规则”的约束。难怪现在这么多企业在挑选程序开发人员的时候，都会注明“良好的编码规范”。现在对于这些无形中的条条框框，项目经理已经开始重视起来。

java 方向及学习方法

几个主要的概念：

JVM Java Virtual Machine（Java 虚拟机），它是一个虚构出来的计算机,是通过在实际的计算机上仿真模拟各种计算机功能来实现的。 Java 虚拟机有自己完善的硬件架构,如处理器、堆栈、寄存器等,还具有相应的指令系统。JVM 屏蔽了与具体操作系统平台相关

的信息,使得 Java 程序只 需生成在 Java 虚拟机上运行的目标代码(字节码),就可以在多种平台上不加修改地运行。Java 虚拟机在执行字节码时,实际上最终还是把字节码解释成具 体平台上的机器指令执行。

JRE Java Runtime Environment（Java 运行环境），运行 JAVA 程序所必须的环境的集合，包含 JVM 标准实现及 Java 核心类库。

JSDK Java Software Development Kit，和 JDK 以及 J2SE 等同。

JDK Java Development Kit(Java 开发工具包):包括运行环境、编译工具及其它工具、源代码等，基本上和 J2SE 等同。

J2ME Java 2 Micro Edition（JAVA2 精简版）API 规格基于 J2SE，但是被修改为可以适合某种产品的单一要求。J2ME 使 JAVA 程序可以很方便的应用于电话卡、寻呼机等小型设备，它包括两种类型的组件，即配置（configuration）和描述（profile）。

J2EE Java 2 Enterprise Edition（JAVA2 企业版），使用 Java 进行企业开发的一套扩展标准，必须基于 J2SE，提供一个基于组件设计、开发、集合、展开企业应用的途 径。J2EE 平台提供了多层、分布式的应用模型，重新利用组件的能力，统一安全的模式以及灵活的处理控制能力。 J2EE 包括 EJB, JTA, JDBC, JCA, JMX, JNDI, JMS, JavaMail, Servlet, JSP 等规范。

J2SE Java 2 Standard Edition（JAVA2 标准版），用来开发 Java 程序的基础，包括编译器、小工具、运行环境，SUN 发布的标准版本中还包括核心类库的所有源代码。

java 分成 J2ME（移动应用开发），J2SE（桌面应用开发），J2EE(Web 企业级应用)，所以 java 并不是单机版的，只是面向对象语言。建议如果学习 java 体系的话可以这样去学习：

*第一阶段：Java 基础，包括 java 语法，面向对象特征，常见 API，集合框架；

*第二阶段：java 界面编程，包括 AWT，事件机制，SWING，这个部分也可以跳过，用的时候再看都能来及；

*第三阶段：java API：输入输出，多线程，网络编程，反射注解等，java 的精华部分；

*第四阶段：数据库 SQL 基础，包括增删改查操作以及多表查询；

*第五阶段：JDBC 编程：包括 JDBC 原理，JDBC 连接库，JDBC API，虽然现在 Hibernate 比 JDBC 要方便许多，但是 JDBC 技术仍然在使用，JDBC 思想尤为重要；

*第六阶段：JDBC 深入理解高级特性：包括数据库连接池，存储过程，触发器，CRM 思想；

*第七阶段：HTML 语言学习，包括 HTML 标签，表单标签以及 CSS，这是 Web 应用开发的基础；

*第八阶段：JavaScript 脚本语言，包括 javaScript 语法和对象，就这两个方面的内容；

*第九阶段：DOM 编程，包括 DOM 原理，常用的 DOM 元素以及比较重要的 DOM 编程思想；

*第十阶段：Servlet 开发，从此开始踏入 java 开发的重要一步，包括 XML，Tomcat 服务器的安装使用操作，HTTP 协议简单理解，Servlet API 等，这个是 java web 开发的基础。

*第十一阶段：JSP 开发：JSP 语法和标签，自定义标签，EL,JSTL 库了解以及 MVC 三层架构的设计模式理念；

*第十二阶段：AJAX 开发：AJAX 原理，请求响应处理，AJAX 开发库；

*第十三阶段：轻量级框架，三大框架之一 Struts 框架的学习，自此踏入 java web 开发的精华部分，包括 Struts 体系架构，各种组件，标签库和扩展性的学习；

*第十四阶段：Hibernate 框架学习，三大框架之一，包括检索映射技术，多表查询技术，缓存技术以及性能方面的优化；

*第十五阶段：Spring 框架的学习，三大框架之一，包括了 IOC,AOP,DataSource，事务，SSH 集成以及 JPA 集成；

*最后呢，还有些 java 的技术，包括 EJB3.0 等，可以选择学习，与三大轻量级框架相比，EJB 就是当之无愧的重量级了。

学习的内容

前两天公司一兄弟说要学习 Java，特意针对公司实际情况整理了这样一份要点，现在发到网上，以后有机会慢慢修改更新。

一、Java 基础知识

1.常用开发工具，Eclipse、NetBeans 以及命令行工具，现在使用的 WSAD 即在 Eclipse 基础上装插件而得。

2.Java 程序的最简单格式，即在一个类中定义 main 方法。

3.数据类型 Java 的数据类型分为简单类型和复杂类型两种，简单类型有 8 个，byte(8b，b 即 bit，位的意思)、short(16b)、int(32b)、long(64b)、char(16b)、boolean(16b)、float(32b)、double(64b)，其他均为复杂类型，例如 Object、String，其中 Object 是所有复杂类型的基类，而且每个简单类型都有对应的复杂类型。关于自定义的复杂类型，可区分为 class 和 interface 两种。

4.变量与参数的概念，定义变量并对变量赋值，如使用 null、true、false 赋值，并注意 long（使用 L 标识）和 float（使用 F 标识）的赋值。需要理解引用与指针的区别和联系。此外需要解释 Java 内存的清理机制，以及装箱和拆箱的概念。

5.编码规范 包、类、接口、字段、方法以及局部变量、参数可以由字母、数字、下划线组成，并以字母开头。需要初步了解以上对象的概念。注释有三种://单行注释、/* 多行注释 */、/** 文档注释 */(注意，Java 是能够根据注释生成文档的)

6.运算符 包括算术、自增/自减、关系（需要强调==与=的区别）、逻辑、按位运算、移位、三元、字符串操作、类型转换、instanceof 等

7.方法的概念 需要了解返回值及 return 语句

8.分支语句 if-else（包括 if、if-else、if-else if-else 等形式，并了解语句的嵌套与代码块的概念）、switch-case（涉及 break 语句）

9.for、while、do-while（涉及 break、continue 语句），同时需要介绍 Java 的数组，数组的维数，并注意数组的 length 与 String 类型的 length 的区别。了解 Collection 框架，包括 Map、Set、List 等接口，以及相应的 HashMap、Hashtable、Vector、ArrayList 等。

10.try-catch-finally、throw、throws 语句，了解 Exception 和 RuntimeException，注意区分必检异常与可规避异常，并了解一些常用的 Exception

11.class 基础知识

1)构造、方法与字段的概念

2)作用域的概念 public、private、protected 与友元（考虑如何不允许类被构造）

3)静态成员与实例成员（解释为何存在不允许类被构造的情况）

4)class 的继承，要说明那些类成员可以继承，那些不可以

12.class 高级知识

1)构造与方法的重载与重写

2)不允许重写的类与类成员 使用 final，借助 static 与 final 定义常量

3)使用父类操作子类的实例

13.abstract class 与 interface

1)多态的用处，发生在使用父类操作子类的实例的情况下，父类的作用体现为契约

2)abstract class 的定义 不允许定义实例，只能单继承

3)interface 的定义 不允许定义实例，允许多继承

4)abstract class 与 interface 的区别 主要是区分使用情况

14.线程的概念（JSP 中用不到，自己去了解下吧）

15.XML 操作 了解 Xerces，该库可以从 Apache 上下载到

16.反射（其实属于高级知识了） 包括 Class、Method、Field、Constructor 等类型的使用，尝试在 xml 文件中定义类的名称及方法，然后操作类，即实现一个简单的 IoC 容器

二、Java 高级知识

1.数据库连接技术（JDBC） 包括 Connection、Statement(包括 PreparedStatement 和 CallableStatement，需要了解 SQL 语句与存储过程哦)、ResultSet，同时需要了解 JDBC 反射的概念

2.Servlet 简介 主要了解 Web.xml 配置文件的修改（主要是区分出/a/b/c.do 与*.do 的区别）、服务器的启动与停止以及 Request、Response、Session、Application、out 等对象

3.JSP 简介 除了解 Servlet 所有的 5 个对象，还应了解 Page 对象（相当于 Servlet 的 this 对象），明白数据在页面内及页面间的传递。此外，还要了解 Servlet 调用 JSP，包括重定位、forward、include 等

4.Web 高级应用 借助 Web.xml，结合 XML 与反射技术，创建一个简单的 MVC 框架

5.JSP 扩展标记 了解 javax.servlet.jsp.tagext.TagSupport 和 javax.servlet.jsp.tagext.TagBodySupport 接口，与自定义的 MVC 框架结合

6.Ajax 其实应该算是前台技术，应该更多的了解 JavaScript（包括 js 基础，例如变量、函数什么的，不过真正的重点应该是 eval 动弹执行函数、通过 div/span 对象的 id 操作其显示或隐藏以及界面动弹修改、用 js 实现面向对象操作等）

7.WebService（如果对 SOA 有兴趣，就了解一下，未来发展趋势之一）

三、Java 工具箱

1.JUnit 需要了解单元测试与测试驱动开发（关键不是 JUnit 本身）

2.Ant 需要了解自动构建（如果对服务器管理没兴趣，那么知道有这个东西就可以了）

3.Log4J 日志工具的作用主要在于开发阶段，不要过多使用 System.out.println

4.AspectJ 即 AOP，也是未来发展趋势之一

四、Java 常用框架（都知道，不废话了）

Struts、Spring、Hibernate、iBatis、JDO2、EJB3 等

面向对象性

Java 基本知识

Java 重点

工作职务的不同会有不同的面试主题。然而，下面几个领域是很常见的：

数据结构和算法

多线程

字节操作

内存分配

对象，继承，设计模式

递归

汇编知识和程序运行原理

后缀类型

.Java java 的源代码

.class java 的字节码