MyJava

目录

[一、 Java简介 3](#_Toc508198507)

[1. Java作用 3](#_Toc508198508)

[2. Java特性 3](#_Toc508198509)

[3. Java开发所需 3](#_Toc508198510)

[4. Java环境安装 3](#_Toc508198511)

[二、 Java基础 5](#_Toc508198512)

[1. Java语法 5](#_Toc508198513)

[2. Java对象 5](#_Toc508198514)

[3. Java注释 5](#_Toc508198515)

[4. Java函数 6](#_Toc508198516)

[5. Java快捷键： 6](#_Toc508198517)

[6. Java关键字 8](#_Toc508198518)

[7. Java控制台 9](#_Toc508198519)

[8. Java数据类型 9](#_Toc508198520)

[9. Java访问修饰符 10](#_Toc508198521)

[10. Java包(Package) 10](#_Toc508198522)

[三、 Java数据结构 12](#_Toc508198523)

[1. Java数组 12](#_Toc508198524)

[2. Java集合 12](#_Toc508198525)

[3. Java字符串 15](#_Toc508198526)

[四、 Java面向对象 16](#_Toc508198527)

[1. 面向对象vs面向过程 16](#_Toc508198528)

[2. Java中面向对象的两个概念 16](#_Toc508198529)

[3. 如何定义一个类 16](#_Toc508198530)

[4. 对象的创建 17](#_Toc508198531)

[5. 内部变量/对象/函数的创建和声明 17](#_Toc508198532)

[6. this关键字 18](#_Toc508198533)

[7. 方法的重载 18](#_Toc508198534)

[8. 变量的作用域 18](#_Toc508198535)

[9. static静态 18](#_Toc508198536)

[10. static作用详解 19](#_Toc508198537)

[11. 内部类 24](#_Toc508198538)

[12. Java包的概念 24](#_Toc508198539)

[五、 Java类（一） 26](#_Toc508198540)

[1. 继承 26](#_Toc508198541)

[2. 多态 26](#_Toc508198542)

[3. 关键字 27](#_Toc508198543)

[4. 方法的重写 27](#_Toc508198544)

[5. 接口interface 27](#_Toc508198545)

[六、 Java类（二） 29](#_Toc508198546)

[1. Object类 29](#_Toc508198547)

[2. 包装类 29](#_Toc508198548)

[3. String类 30](#_Toc508198549)

[4. StringBuilder和StringBuffer类 31](#_Toc508198550)

[七、 Java反射 32](#_Toc508198551)

[1. newInstance和new 32](#_Toc508198552)

[2. 类、实例和Class 33](#_Toc508198553)

[3. 泛型 37](#_Toc508198554)

[八、 Java其它 40](#_Toc508198555)

[1. Java异常 40](#_Toc508198556)

[2. Java I/O 40](#_Toc508198557)

[3. Java序列化 42](#_Toc508198558)

[4. Java杂集 42](#_Toc508198559)

[九、 Java界面 43](#_Toc508198560)

[1. Java界面插件 43](#_Toc508198561)

[2. Java页面布局 43](#_Toc508198562)

[十、 Java程序 44](#_Toc508198563)

[1. Java可执行文件 44](#_Toc508198564)

# Java简介

## Java作用

* + 1. Java电脑程序（J2SE）
    2. Java企业级开发（J2EE）
    3. Java嵌入式程序开发（J2ME）

E - Edition 版本

## Java特性

* + 1. 跨平台：一次编译，到处运行
    2. 面向对象：描述属性、特征更加方便
    3. 安全性：编译型语言
    4. 多线程：同时处理多个任务
    5. 简单易用：框架多、轮子多

## Java开发所需

* + 1. JRE - Java Runtime Enviroument（运行环境）
    2. JDK - Java Development Kit（开发工具）
    3. IDE - Eclipse, Netbeans, InteliJ IDEA（集成环境）

## Java环境安装

* + 1. Java安装设置

1. Java平台——在Orcale官网中下载Java平台（Java SE、Java EE和Java ME JDK）
2. Java IDE——在eclipse官网中下载eclipse-inst-win64程序，安装需要的工具（也可用记事本编辑修改java源程序）

<http://jingyan.baidu.com/article/63acb44a35bd5f61fcc17e9e.html>

<http://jingyan.baidu.com/article/a24b33cd59b58e19fe002bb9.html>（参考）

下载并安装JDK（最新版本1.8，且包含JRE），指定安装目录（推荐默认\Program files\Java\），下一步直到全部安装

NOTE: 确保自己电脑上有且仅有一个JRE&JDK版本

* 环境变量"JAVA\_HOME=JDK的安装目录"
* 环境变量"JRE\_HOME=JRE的安装目录"（JDK路径下）
* (windows) 找到Path并添加 ...;%JAVA\_HOME%\bin;
  + 1. Eclipse安装设置

a）eclipse下载：在eclipse官网中（http://www.eclipse.org/downloads/eclipse-packages/）下载Eclipse IDE for Java EE Developers

b）eclipse语言包下载：http://www.eclipse.org/babel/downloads.php

（参考link：<http://www.jb51.net/softjc/141995.html>）

c）环境语言设置：

（参考：<http://blog.csdn.net/moqiang02/article/details/20567753>）

Eclipse只需要把eclipse.ini里面任意新起一行，增加下面参数即可：  
-Duser.language=[EN](https://www.baidu.com/s?wd=EN&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3m103mHF-ujIBuWcYmWnv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjmYnHTLPHcv)（英文） 或 -Duser.language=ZH（中文）

使用快捷方式修改环境语言步骤如下：

1、复制Eclipse的快捷方式；

2、右键快捷方式-》属性，在“目标”的后边加上 -nl "en"

之前的："D:\Program Files\Java\eclipse\eclipse.exe"

加后的："D:\Program Files\Java\eclipse\eclipse.exe" -nl "en"

* + 1. Tomcat安装设置

1. 安装

—添加环境变量：

变量名：CATALINA\_HOME

路径：D:\Program Files\Java\javaee\apache-tomcat-7.0.72

—运行路径（D:\Program Files\Java\javaee\apache-tomcat-7.0.72）bin下的startup.bat（运行的CMD不能关）(使用shutdown.bat关闭服务器)

—在浏览器中输入（http://localhost:8080/）【显示Tomcat网页则为配置正常】

* + 1. MySql安装设置

（参考：<http://www.2cto.com/database/201506/409821.html>）

* + 1. IDEA安装设置
    2. SVN/Git安装设置

（SVN、Git详见“SVN&Git&Maven配置”文件夹）

# Java基础

## Java语法

对于java语言本身，java是一个纯粹的面向对象的语言，类(class)是作为java程序编写、运行的最基本的单位。条件语句、函数语句等都不可能脱离类独立存在，程序中定义的所有内容都必须在类中。

## Java对象

面向对象，万物皆对象：

* + 1. 封装：隐藏对象的属性和细节，仅对外公开接口，控制程序中属性的读、修改及其他功能，以及它们的访问类别，将抽象出的内容（数据、行为等）进行组合，形成一个实体。
    2. 继承：面向对象的语言中，任何一个实体类都可以理解成一个对象。在这些对象使用或者包含某些共同属性时，可以通过继承的方式来进行获取。被继承的类叫做父类，继承的类叫做派生类。
    3. 多态：多种状态，一个实体类实现的时候具有的多种形式。同样的一个属性或方法，在不同的类对象实现后，可以展现出不同的状态。

## Java注释

* + 1. 单行注释 //
    2. 多行注释 /\* \*/
    3. JavaDoc注释 /\*\* \*/
    4. 代码块（如VS的#region）//region //endregion[可能导致//注释后面没内容会在eclipse中显示不出来]

（<http://blog.csdn.net/huyuyang6688/article/details/50073253>）

## Java函数

* + 1. 普通函数：

修饰符 返回值 函数名(参数类型 参数名) {

...

}

e.g.:

public void add(int num1, int num2) {

return num1 + num2;

}

NOTE: void在Java中代表"无"的意思

* + 1. 构造函数

调用：使用"."的方式进行调用

* + 1. 入口函数

public static void main(String[] args) {

}

输出一个Hello world!

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

NOTE: 所有的JAVA文件都必须以.java结尾，编译后会变成.class的字节码文件 - jvm(Java Virtual Machine)真正进行解释执行的

定义Java文件：有且仅有一个public class并与文件名相同。

## Java快捷键：

常用快捷键：

ctrl + D ——删除一行

Alt + UP/Down——上下移动一行

Alt + / ——补全

Ctrl + /——注释/解释

F3（Ctrl + "\*"）——跳到定义

Ctrl+Shift+F(英文模式下)——代码格式化

1、Ctrl+M --切换窗口的大小

2、Ctrl+Q --跳到最后一次的编辑处

3、 F2 --当鼠标放在一个标记处出现Tooltip时候按

F2则把鼠标移开时Tooltip还会显示即Show Tooltip Description。

F3 --跳到声明或定义的地方。

F5 --单步调试进入函数内部。

F6 --单步调试不进入函数内部,如果装了金山词霸2006则要把“取词开关”的快捷键改成其他的。

F7 --由函数内部返回到调用处。

F8 --一直执行到下一个断点。

4、Ctrl+Pg~ --对于XML文件是切换代码和图示窗口

5、Ctrl+Alt+I --看Java文件中变量的相关信息

6、Ctrl+PgUp --对于代码窗口是打开“Show List”下拉框,在此下拉框里显示有最近曾打开的文件

7、Ctrl+/ --在代码窗口中是这种//~注释。

Ctrl+Shift+/ --在代码窗口中是这种/\*~\*/注释,在JSP文件窗口中是〈!--~--〉。

8、Alt+Shift+O(或点击工具栏中的Toggle Mark Occurrences按钮) 当点击某个标记时可使本页面中其他地方的此标记黄色凸显,并且窗口的右边框会出现白色的方块,点击此方块会跳到此标记处。

9、右击窗口的左边框即加断点的地方选Show Line Numbers可以加行号。

10、Ctrl+I格式化激活的元素Format Active Elements。 Ctrl+Shift+F格式化文件Format Document。

11、Ctrl+S保存当前文件。 Ctrl+Shift+S保存所有未保存的文件。

12、Ctrl+Shift+M(先把光标放在需导入包的类名上) 作用是加Import语句。 Ctrl+Shift+O作用是缺少的Import语句被加入,多余的Import语句被删除。

13、Ctrl+Space提示键入内容即Content Assist,此时要将输入法中

Chinese(Simplified)IME-Ime/Nonlme Toggle的快捷键(用于切换英文和其他文字)改成其他的。

Ctrl+Shift+Space提示信息即Context Information。

14、双击窗口的左边框可以加断点。

15、Ctrl+D删除当前行。 其它:

Ctrl+1 快速修复(最经典的快捷键,就不用多说了)

Java数据结构快速入门Java数据结构基...Java数据结构基...

Ctrl+D: 删除当前行

Ctrl+Alt+↓ 复制当前行到下一行(复制增加) Ctrl+Alt+↑ 复制当前行到上一行(复制增加)

Alt+↓ 当前行和下面一行交互位置(特别实用,可以省去先剪切,再粘贴了) Alt+↑ 当前行和上面一行交互位置(同上) Alt+← 前一个编辑的页面

Alt+→ 下一个编辑的页面(当然是针对上面那条来说了) Alt+Enter 显示当前选择资源(工程,or 文件 or文件)的属性

Shift+Enter 在当前行的下一行插入空行(这时鼠标可以在当前行的任一位置,不一定是最后) Shift+Ctrl+Enter 在当前行插入空行(原理同上条) Ctrl+Q 定位到最后编辑的地方

Ctrl+L 定位在某行 (对于程序超过100的人就有福音了) Ctrl+M 最大化当前的Edit或View (再按则反之) Ctrl+/ 注释当前行,再按则取消注释 Ctrl+O 快速显示 OutLine Ctrl+T 快速显示当前类的继承结构 Ctrl+W 关闭当前Editer

Ctrl+K 参照选中的Word快速定位到下一个

Ctrl+E 快速显示当前Editer的下拉列表(如果当前页面没有显示的用黑体表示) Ctrl+/(小键盘) 折叠当前类中的所有代码 Ctrl+×(小键盘) 展开当前类中的所有代码

Ctrl+Space 代码助手完成一些代码的插入(但一般和输入法有冲突,可以修改输入法的热键,也可以暂用Alt+/来代替)

Ctrl+Shift+E 显示管理当前打开的所有的View的管理器(可以选择关闭,激活等操作) Ctrl+J 正向增量查找(按下Ctrl+J后,你所输入的每个字母编辑器都提供快速匹配定位到某个单词,如果没有,则在stutes line中显示没有找到了,查一个单词时,特别实用,这个功能Idea两年前就有了)

Ctrl+Shift+J 反向增量查找(和上条相同,只不过是从后往前查)

Ctrl+Shift+F4 关闭所有打开的Editer Ctrl+Shift+X 把当前选中的文本全部变味小写 Ctrl+Shift+Y 把当前选中的文本全部变为小写 Ctrl+Shift+F 格式化当前代码

Ctrl+Shift+P 定位到对于的匹配符(譬如{}) (从前面定位后面时,光标要在匹配符里面,后面到前面,则反之)

重构里面常用的：

下面的快捷键是重构里面常用的,(注:一般重构的快捷键都是Alt+Shift开头的了)

Alt+Shift+R 重命名 (是我自己最爱用的一个了,尤其是变量和类的Rename,比手工方法能节省很多劳动力)

Alt+Shift+M 抽取方法 (这是重构里面最常用的方法之一了,尤其是对一大堆泥团代码有用) Alt+Shift+C 修改函数结构(比较实用,有N个函数调用了这个方法,修改一次搞定) Alt+Shift+L 抽取本地变量( 可以直接把一些魔法数字和字符串抽取成一个变量,尤其是多处调用的时候)

Alt+Shift+F 把Class中的local变量变为field变量 (比较实用的功能) Alt+Shift+I 合并变量(可能这样说有点不妥Inline) Alt+Shift+V 移动函数和变量(不怎么常用) Alt+Shift+Z 重构的后悔药(Undo)

## Java关键字

* + 1. Java语法符号：

{} [] . <> = ; 等

* + 1. 关键字（命名时可能冲突的）

访问控制：

public, private, protected

类、方法和修饰符：

class, abstract, extends, final, implements, interface, new, synchronized, native, transient, volatile, static

程序控制：

if, else, switch, case, default, break, continue, for, do, while, instanceof

错误处理：

try, catch, throw, throws, assert

基本类型：

byte, int(short, long), float, double, boolean, char, null, true, false

变量引用：

super, this, void

其他：

"@注解", "lambda表达式"(Java8以后才支持的)

保留字：

goto, const

## Java控制台

控制台的输入输出

输出：

System.out.println(); // 输出整行

System.out.print(); // 单个输出

System.out.printf(); // 格式化输出

占位符：

- %c 单个字符

- %d 整数, %o 八进制, %x 十进制

- %f 浮点型

- %s 字符串

NOTE: 占位符默认不会进行类型转换，会抛出异常

输入：

Scanner sc = new Scanner(System.in);

sc.next...(); // 获取不同属性的值

转义字符：某种特定符号属于语法时，需要使用'\'进行转义

- \n 换行

- \t 制表符

- \" 双引号

- \\ 反斜杠

## Java数据类型

* + 1. Java本身是一个强类型的语言，对于每一个表达式的定义和生命，都必须以显示的方式进行。

NOTE: 变量的定义都必须先声明，后使用。

* + 1. Java中提供了8种基本数据类型

- byte: 8b

- short: 16b

- int: 32b

- long: 64b

- char: 16b

- float: 32b (需要在小数后加f/F)

- double: 64b

- boolean: 1b

默认值：

int(及其他数字类型): 0

boolean: false

特殊：

String: 字符串

字符串的默认值：null

* + 1. Java中的类型转换

XClass a = (XClass) b;

- 自动类型转换

- 强制类型转换（可能会有数据丢失的情况）

## Java访问修饰符

* + 1. public - 共开的，外部可以访问
    2. private - 私有的，只能在内部访问
    3. protected - 受保护的，只能在内部或子类访问
    4. (default) - 默认的，同一包下可以访问

在说明这四个关键字之前，我想就class之间的关系做一个简单的定义，对于继承自己的class，base class可以认为他们都是自己的子女，而对于和自己一个目录下的classes，认为都是自己的朋友。

1、public：public表明该数据成员、成员函数是对所有用户开放的，所有用户都可以直接进行调用

2、private：private表示私有，私有的意思就是除了class自己之外，任何人都不可以直接使用，私有财产神圣不可侵犯嘛，即便是子女，朋友，都不可以使用。

3、protected：protected对于子女、朋友来说，就是public的，可以自由使用，没有任何限制，而对于其他的外部class，protected就变成private。

作用域 当前类 同一package 子孙类 其他package

public √ √ √ √

protected √ √ √ ×

friendly √ √ × ×

private √ × × ×

注：不写时默认为friendly

## Java包(Package)

（java包的机制类似于其他语言的名称空间）原则上提供了相同功能的java类，则放在一个包内。

访问权限：（default同一包内访问）；

同一个文件只能存在一个package声明；

package只能在程序的最上层声明。

导入包

import java.util.Date;

(Note:)

常用包

-java.lang.\*；(\*所有的包)，默认引入，包含java中基本的核心库

-java.util.\*;提供集合类框架、日期、工具类库；

-.net.\*；网络

-.sql.\*；数据库

-.swing/swt.\*；swt基于图形化界面开发，swing基于swt的完善框架；

-.text.\*；提供一些与语言无关的方式处理文本/日期/数字；

（Note：包名的定义，以公司/组织的域名倒置的形式进行命名）

# Java数据结构

## Java数组

* + 1. 类型[] 变量名 = new 类型[长度];

int[] arr = new int[10];

* + 1. 类型[] 变量名 = { 常量（并不是固定值） };
    2. 遍历数组

- 通过"数组.length"进行length次循环遍历

- for-each循环 for(类型 变量 : 类型数组) {}

NOTE: 注意数组越界问题(java.lang.ArrayIndexOutOfBoundsException)

* + 1. 数组的工具类

"Arrays"

- binarySearch() 查找

- copyOf() 拷贝

- sort() 排序

- toString() / deepToString() 多维数组

- equals() 判断数组相同 / deepEquals()

- fill() 以何种形式填充

## Java集合

Java中的集合在J2SE的核心库中是最重要的部分之一。集合存在于java.util包中。对于数组而言，数组保存的个数是在定义数组时就要给定，而在开发中很多时候都需要动态创建并操作，使用数组就会很不方便。集合就是为了解决这些问题，保存了具有相同或同一特性的实例（或引用），从而简便地完成遍历、查找和修改。

使用集合满足的条件：

- 从需求上讲，集合中的元素的功能/意义应该相同或相近

- 从代码上讲，类型应该是一致的

Java中，有三种常见的集合

* + 1. List-ArrayList 不唯一的有序列表

可以保存重复的数据，每个元素都有自己的索引值。添加、删除速度很快，随机读取速度很低。可以理解为动态数组，通过index来定位。

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

- size()

- add()

- remove()

- addAll()

- clear()

- contains()

- isEmpty()

* + 1. Set-HashSet 唯一的无序列表

不保证元素的顺序，只能存储不重复的元素。元素的地址和hashCode有关，随机存储速度很快，遍历速度很低。

- size()

- add()

- remove()

- clear()

- contains()

- isEmpty()

* + 1. Map-HashMap 存放键值对的图

以hashCode形式存放的键值对，键和值都可以自定义。存储映射关系，不维护顺序，也允许重复，但键必须唯一。

- size()

- put()

- remove()

- putAll()

- containsKey()

- isEmpty()

遍历使用：

- map.keySet() 通过key的集合进行遍历

- Map.Entry 通过Entry遍历

其他类型：Vector, Stack, Queue

* + 1. Collection接口

在Collection(List, Set)和Map中都定义了用于实现集合的基本操作（增删改查），在实际开发中用到的ArrayList等都是它们的实现类。根据Java的多态性原则，对当前公共抽象的方法（内容）进行定义，降低以降低程序的耦合度。

- 接口式编程

- 高内聚 vs 低耦合

* + 1. 迭代器Iterator

迭代器在集合中是起迭代作用的(for-each)，会按照正序（从前到后）对集合进行迭代。

- hastNext()

- next()

- remove()

NOTE: ListIterator双向迭代器

* + 1. 集合工具类的封装：

- Arrays

- Collections

- max() min() 求最大值 最小值

- shuffle() 随机排序

- reverse() 反序

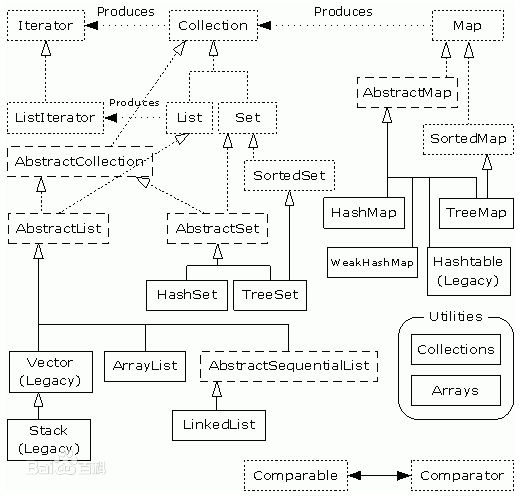
- sort() 排序

java.util.Comparator 接口

通过比较接口，能在某些函数（情况）下定义一个比较规则，在通过迭代器进行遍历的时候，应用这个比较规则，实现求值、排序等功能

<? super T> - 需要的传来的泛型是T的超集（父类）

NOTE: 了解Comparable和Comparator的区别



## Java字符串

* + 1. 字符串为空

以下是java 判断字符串是否为空的四种方法:

方法一: 最多人使用的一个方法, 直观, 方便, 但效率很低:

if(s == null ||"".equals(s));

方法二: 比较字符串长度, 效率高, 是我知道的最好一个方法:

if(s == null || s.length() <= 0);

方法三: Java SE 6.0 才开始提供的方法, 效率和方法二几乎相等, 但出于兼容性考虑, 推荐使用方法二.

if(s == null || s.isEmpty());

方法四: 这是一种比较直观,简便的方法,而且效率也非常的高,与方法二、三的效率差不多:

if (s == null || s == "");

注意:s == null 是有必要存在的.

　　如果 String 类型为null, 而去进行 equals(String) 或 length() 等操作会抛出java.lang.NullPointerException.

　　并且s==null 的顺序必须出现在前面，不然同样会抛出java.lang.NullPointerException.

　　如下Java代码:

　　String str = null;

　　if(str.equals("") || str= == null){//会抛出异常

　　 System.out.println("success");

　　}

　　// "".equals(str);后置确保不会遇null报错。

# Java面向对象

## 面向对象vs面向过程

面向过程：是分析问题解决所需要的步骤，然后通过函数将这些步骤进行一步步实现。使用的时候调用逐个相应功能的函数

面向对象：把问题分析成各个独享的部分，建立对象，并不是单纯为了完成某一个步骤，而是为了描述某个事务或系统在整个解决问题方案中的步骤和行为

## Java中面向对象的两个概念

对象和类

* + 1. 对象(object 或 instance)

在现实世界中是一个具体的事务或事件

- 有机/无机个体：鹦鹉，雪佛兰汽车，氯化钠

- 概念：行为艺术，后现代主义拍摄风格

* + 1. 类(class):

可以描述一组具有相同行为特征的对象的抽象概念或概括

- 有机/无机个体：鸟，机械，材料

- 概念：艺术，电影

* + 1. 实例化

在java里面，把类(class)当成一个数据模型，用户可以通过变量来实现或创造出这个模型的实体，将这一操作或过程称之为实例化

## 如何定义一个类

* + 1. 通过"class"关键字，"修饰符 class 类名"

class Student {

// 定义一些属性

private int age; // Student的属性 - age

private String name; // 属性 - name

public void study() { // Student的study方法

}

public Student() { // 构造函数（构造器）

// “可以用何种形式来构建该类的实例”

// 构造函数本身没有返回值，因为返回的就是这个类的实例

// 构造函数的定义（函数名）必须和类相同

// 调用时只能通过new来调用

// 默认情况下有一个无参的构造函数

}

// 可以定义一个空类

class Address {

// 类中也可以嵌套其他类

}

}

NOTE: 没有提供构造函数的时候，系统有一个默认的无参构造函数进行创建实例，指定了构造函数后，这个无参构造将会被覆盖

* + 1. 类定义的几个特性

- Java命名方式遵循驼峰式命名

- Java类首字母一般都是大写

- 变量、方法、静态区等定义的时候不存在顺序关系，一般情况遵循 静态区->变量->构造器->方法 的从上到下定义

- 对于代码中的缩进、空格、换行，原则上编译器是忽略的

## 对象的创建

* 通过Object obj = new Object(); 进行创建
* 通过String str = "abc"; 进行创建
* 匿名类 new Object();

匿名类（匿名内部类）：是一个在很多高级面向对象语言，并且非常常见，它的产生主要是为了解决某些类在特定环境下仅有一个实例需要调用的时候，可以用“不对该类进行声明就能使用”，对于底层来说，实际还是给它创建一个实例化对象，只不过开发人员不用关心。

## 内部变量/对象/函数的创建和声明

- 变量的声明： 修饰符 类型 变量名;( = XXX)

- 对象的声明： Object xxx = new Object();

- 函数的生命： 修饰符 返回值 函数名(参数...) {}

修饰符：

private protected public // 访问修饰符

- static 静态的

- abstract 抽象的

- final 常量，最终的

- synchronized 同步锁

NOTE: 修饰符中，有些修饰符是互斥的，所以使用时需要注意

调用： 使用"."的形式进行调用

## this关键字

java中"this"关键字指的是“当前对象的实例”。本身是相对于对象内部而言的，所以外部以任何形式的调用都跟this没有关系。通过this关键字，可以得到当前类中的变量/方法。

## 方法的重载

在java中，确定方法（调用）的唯一形式是通过函数名和参数。而在java中，相同名字的方法可以多次定义，这个特性叫做方法的重载。方法的重载必须满足以下条件：

- 方法名相同

- 参数不同

- 参数类型不同

- 参数个数不同

- 参数类型个数均不相同

- 与方法的返回值无关，和参数名无关

NOTE: 构造函数作为特殊的函数类型，也可以重载

## 变量的作用域

* + 1. 局部变量

存在于某个方法/代码块内部，在块中定义的变量，从被赋值开始，到块执行完成，变量销毁

* + 1. 全局变量

存在于某个类中的变量，从任意地方被复制开始，在类被销毁之前一直存在

## static静态

关键字"static"，被static修饰的变量或方法被称为是静态的。一般生命的变量属性，都是“动态生成”的，即什么时候用（赋值等操作），什么时候生成。静态的正好相反，定义了静态事物的话，静态事物会一直存在。

* + 1. Java中，对源代码里面的变量、方法的内存区域作了以下的划分：

- 栈区：存函数的参数值，局部变量值等

- 堆区：程序员进行分配，可能会由OS进行回收

- 全局（静态）区：

- 未初始化区：仅声明的（全局和静态）变量

- 初始化区：被初始化了的变量

- 文字常量区：存一些字符串常量

- 程序代码区：存放程序的二进制代码

Java里面上述所有的操作区域都不是Java程序员控制的。由JVM自行调度(Garbage Collection - GC)。

* + 1. Java中静态变量/方法

本身是存在于静态空间里的，持有着类的引用，类可以创建多个实例，而这些实例会共享同一片静态空间。这个“类的静态空间”，是类在第一次被加载时就产生的（NOTE: 不是指被实例化new，而是ClassLoader加载的时候）。

* + 1. 静态代码块

static {

...

}

* + 1. 静态属性一些特点

- 静态代码块处理的所有内容都必须是静态的

- 调用静态方法/属性可以用实例调用，但是不推荐。应使用类的方式进行调用

- 静态方法里面只能引用/调用静态事物

- 静态方法由于和具体实例不相关，所以不能在方法中调用this/super

## static作用详解

static表示“全局”或者“静态”的意思，用来修饰成员变量和成员方法，也可以形成静态static代码块，但是Java语言中没有全局变量的概念。

被static修饰的成员变量和成员方法独立于该类的任何对象。也就是说，它不依赖类特定的实例，被类的所有实例共享。

只要这个类被加载，Java虚拟机就能根据类名在运行时数据区的方法区内定找到他们。因此，static对象可以在它的任何对象创建之前访问，无需引用任何对象。

用public修饰的static成员变量和成员方法本质是全局变量和全局方法，当声明它类的对象市，不生成static变量的副本，而是类的所有实例共享同一个static变量。

static变量前可以有private修饰，表示这个变量可以在类的静态代码块中，或者类的其他静态成员方法中使用（当然也可以在非静态成员方法中使用--废话），但是不能在其他类中通过类名来直接引用，这一点很重要。实际上你需要搞明白，private是访问权限限定，static表示不要实例化就可以使用，这样就容易理解多了。static前面加上其它访问权限关键字的效果也以此类推。

static修饰的成员变量和成员方法习惯上称为静态变量和静态方法，可以直接通过类名来访问，访问语法为：

类名.静态方法名(参数列表...)

类名.静态变量名

用static修饰的代码块表示静态代码块，当Java虚拟机（JVM）加载类时，就会执行该代码块（用处非常大，呵呵）。

* + 1. static变量

　按照是否静态的对类成员变量进行分类可分两种：一种是被static修饰的变量，叫静态变量或类变量；另一种是没有被static修饰的变量，叫实例变量。

两者的区别是：

　对于静态变量在内存中只有一个拷贝（节省内存），JVM只为静态分配一次内存，在加载类的过程中完成静态变量的内存分配，可用类名直接访问（方便），当然也可以通过对象来访问（但是这是不推荐的）。

　对于实例变量，没创建一个实例，就会为实例变量分配一次内存，实例变量可以在内存中有多个拷贝，互不影响（灵活）。

所以一般在需要实现以下两个功能时使用静态变量：

在对象之间共享值时

方便访问变量时

* + 1. 静态方法

静态方法可以直接通过类名调用，任何的实例也都可以调用，

因此静态方法中不能用this和super关键字，不能直接访问所属类的实例变量和实例方法(就是不带static的成员变量和成员成员方法)，只能访问所属类的静态成员变量和成员方法。

因为实例成员与特定的对象关联！这个需要去理解，想明白其中的道理，不是记忆！！！

因为static方法独立于任何实例，因此static方法必须被实现，而不能是抽象的abstract。

例如为了方便方法的调用，Java API中的Math类中所有的方法都是静态的，而一般类内部的static方法也是方便其它类对该方法的调用。

静态方法是类内部的一类特殊方法，只有在需要时才将对应的方法声明成静态的，一个类内部的方法一般都是非静态的

* + 1. static代码块

　static代码块也叫静态代码块，是在类中独立于类成员的static语句块，可以有多个，位置可以随便放，它不在任何的方法体内，JVM加载类时会执行这些静态的代码块，如果static代码块有多个，JVM将按照它们在类中出现的先后顺序依次执行它们，每个代码块只会被执行一次。例如：

public class Test5 {

private static int a;

private int b;

static{

Test5.a=3;

System.out.println(a);

Test5 t=new Test5();

t.f();

t.b=1000;

System.out.println(t.b);

}

static{

Test5.a=4;

System.out.println(a);

}

public static void main(String[] args) {

// TODO 自动生成方法存根

}

static{

Test5.a=5;

System.out.println(a);

}

public void f(){

System.out.println("hhahhahah");

}

}

运行结果：

3

hhahhahah

1000

4

5

　利用静态代码块可以对一些static变量进行赋值，最后再看一眼这些例子，都一个static的main方法，这样JVM在运行main方法的时候可以直接调用而不用创建实例。

* + 1. static和final一块用表示什么

static final用来修饰成员变量和成员方法，可简单理解为“全局常量”！

对于变量，表示一旦给值就不可修改，并且通过类名可以访问。

对于方法，表示不可覆盖，并且可以通过类名直接访问。

有时你希望定义一个类成员，使它的使用完全独立于该类的任何对象。通常情况下，类成员必须通过它的类的对象访问，但是可以创建这样一个成员，它能够被它自己使用，而不必引用特定的实例。在成员的声明前面加上关键字static(静态的)就能创建这样的成员。如果一个成员被声明为static，它就能够在它的类的任何对象创建之前被访问，而不必引用任何对象。你可以将方法和变量都声明为static。static 成员的最常见的例子是main( ) 。因为在程序开始执行时必须调用main() ，所以它被声明为static。

声明为static的变量实质上就是全局变量。当声明一个对象时，并不产生static变量的拷贝，而是该类所有的实例变量共用同一个static变量。声明为static的方法有以下几条限制：

它们仅能调用其他的static 方法。

它们只能访问static数据。

它们不能以任何方式引用this 或super（关键字super 与继承有关，在下一章中描述）。

如果你需要通过计算来初始化你的static变量，你可以声明一个static块，Static 块仅在该类被加载时执行一次。下面的例子显示的类有一个static方法，一些static变量，以及一个static 初始化块：

// Demonstrate static variables，methods，and blocks.

class UseStatic {

static int a = 3;

static int b;

static void meth(int x) {

System.out.println("x = " + x);

System.out.println("a = " + a);

System.out.println("b = " + b);

}

static {

System.out.println("Static block initialized.");

b = a \* 4;

}

public static void main(String args[]) {

meth(42);

}

}

一旦UseStatic 类被装载，所有的static语句被运行。首先，a被设置为3，接着static 块执行(打印一条消息)，最后，b被初始化为a\*4 或12。然后调用main()，main() 调用meth() ，把值42传递给x。3个println ( ) 语句引用两个static变量a和b，以及局部变量x 。

注意：在一个static 方法中引用任何实例变量都是非法的。

下面是该程序的输出：

Static block initialized.

x = 42

a = 3

b = 12

在定义它们的类的外面，static 方法和变量能独立于任何对象而被使用。这样，你只要在类的名字后面加点号运算符即可。例如，如果你希望从类外面调用一个static方法，你可以使用下面通用的格式：

classname.method( )

这里，classname 是类的名字，在该类中定义static方法。可以看到，这种格式与通过对象引用变量调用非static方法的格式类似。一个static变量可以以同样的格式来访问——类名加点号运算符。这就是Java 如何实现全局功能和全局变量的一个控制版本。

下面是一个例子。在main() 中，static方法callme() 和static 变量b在它们的类之外被访问。

class StaticDemo {

static int a = 42;

static int b = 99;

static void callme() {

System.out.println("a = " + a);

}

}

class StaticByName {

public static void main(String args[]) {

StaticDemo.callme();

System.out.println("b = " + StaticDemo.b);

}

}

下面是该程序的输出：

a = 42

b = 99

static成员是不能被其所在class创建的实例访问的。

如果不加static修饰的成员是对象成员，也就是归每个对象所有的。

加static修饰的成员是类成员，就是可以由一个类直接调用，为所有对象共有的

## 内部类

类和类之间可以有嵌套关系，称之为内部类

class A {

class B {

...

}

static class C {

...

}

}

* + 1. 原则：

- 内部类和外部类不能重名

- 内部类中的资源可以互相访问（无视访问修饰符）

- 如果存在多层嵌套关系，则需要使用"类名.this.资源"的形式具体指定访问哪个类中的资源

* + 1. 形式上有两种：

- 成员式：内部类作为成员的形式存在

- 在类中可以直接new它的实例

B b = new B();

- 在外部需要使用类的实例去实例化内部类

A a = new A();

A.B b = a.new B();

- 静态式：内部类作为静态存在

A.C c = new A.C();

## Java包的概念

关键字package

Java的包机制类似于其他语言的“名字空间”，原则上提供了相同功能的java类，应该放到一个包中去

- 包：文件系统中的体现就是文件夹

- 访问权限：(default)

- 同一个文件只能存在一个package声明

- package 必须在程序的最上方定义

* + 1. 导入包

关键字import

NOTE: 自动导入的时候要注意同名不同的包的类，不要

导错

* + 1. 常用java包

- java.lang.\*

默认引入，包含Java中基本的核心库

- java.util.\*

提供集合类框架、日期、工具类库

- java.net.\*

提供了一些网络开发相关的类库

- java.sql.\*

提供了一些数据库相关的类库

- java.swing/swt.\*

swt基于图形化界面开发，swing基于swt的完善框架

- java.text.\*

提供了一些与语言无关的方式处理文本、日期和数字

NOTE: 包名的定义，以公司/组织的域名倒置的形式进行命名。com.infosys.javaweb.xxx

# Java类（一）

## 继承

* + 1. 继承的几个特性
* Java是不支持多重继承的
* 祖先可以嵌套
* 继承可以传递，子类也可以使用祖先的开放资源
* 继承会继承所有祖先的（开放）属性
* 子类可以根据自己的特性重写或拓展父类的功能
* 父类包含非空构造函数时，子类必须在构造器中显式调用父类的构造函数
  + 1. 继承在java中的关键字是"extends"

- 父类（基类）：被继承的类 - super

- 子类（派生类）：继承而产生的类 - sub

* + 1. 父类和子类关系

父类和子类之间并不像现实生活中的“继承”，更多的是类似于“包含”的关系

如：人类-男人-约翰 植物-水果-苹果

访问修饰：

public - 共有的，外部任意访问

private - 私有的，只有自己能访问

protected - 受保护的，自己、子类和同包可以访问

(default) - 默认，同包可以访问

Java继承：类B继承了类A，则B可以访问到A中所有符合访问修饰符的属性、方法和构造器等。同样也包括祖先的资源。

NOTE:

- 资源的享用是指派生类可以共享（继承）父类所有可访问的属性，反之是不允许的

- static类是可以被继承的，而static类本身不能继承自非static类

## 多态

多态值同一个事物可以出现的多种形态。封装和继承从某种角度说都是为了多态服务的。

对编程语言来说：多态是允许不同类的对象对同一个消息作出不同相应，同一消息根据发送对象的不同采取不同的形式。

多态 = 动态绑定

实现多态的条件：

- 继承（Java中也可以实现接口）

- 重写

- 父类指向子类的对象

## 关键字

* + 1. super关键字

使用"super"可以调用父类（直接父类，且不能嵌套）关键字的资源，可以理解成与"this"相对，访问的是父类的实例

- 属性 super.name

- 方法 super.method();

- 构造器 super(); 必须写在子类构造器的最上面

NOTE: 当父类构造器不是默认时（无参构造函数），子类需要在自己的构造函数中显式调用父类的构造器

* + 1. final关键字

"final"关键字本身是指“最终的”，在java里面充当了类似const的角色。final本身可以用来修饰类、变量和方法。

- final int a = 5; // 变量不可以被修改

- final class A {} // 不能被继承

- final void show() {} // 子类不能重写该函数

* + 1. abstract关键字

抽象的"abstract"，可以修饰方法和类，即当前方法或类为“抽象定义”的，需要子类去实现。

NOTE: 抽象类和抽象方法肯定是同时存在的，即一个类中有抽象方法，这个类必定是抽象类

- final 和 abstract是矛盾的

- abstract 在实现时，可以由任意子类实现，同时若子类也是abstract的，则仍可以交给它的子类实现

## 方法的重写

当父类的方法满足不了子类的需求，子类可以针对这个方法进行重写

- 方法重写使用"@Override"关键字

- 重写的方法必须跟父类完全相同

NOTE: 方法的重写 vs 方法的重载

## 接口interface

Java本身是不支持多重继承，为了避免在C/C++中多重继承引发的混乱关系。其他的特性使用接口定义的形式进行实现。通过接口实现所有的方法定义，完成Java语言的多态性

修饰符 interface 接口名 {

...

}

实现接口：class 类 implements 接口 {}

* + 1. 接口的特性

- 接口本身是抽象的，默认就带有abstract

- 接口中的方法也是抽象的

- 接口中所有的方法都是public的

- 接口可以在类中定义

* + 1. 接口和类的区别

- 接口本身不能用来实例化

- 接口没有构造方法

- 接口本身所有的方法都是abstract

- 接口中定义的所有变量都是static final的

- 接口不能被实现，但可以被继承

- 接口是可以多重继承的

* + 1. 抽象类和接口的区别

- 接口不能实例化对象

- 接口没有构造方法

- 接口全部都是abstract，抽象类可以写具体的方法

- 接口是被实现的，而抽象类是继承的

- 接口是可以多重继承的

- 接口的变量全部都是static final的

# Java类（二）

## Object类

java.lang.Object

在Java中，所有的类都继承自Object，Object是所有类的祖先，默认是不需要进行继承的

- getClass(); 获取当前类的类对象

- hashCode(); 获取当前Object实例的hashcode

- equals(); 判断两个对象的值是否相等

- toString(); 以String的形式进行输出

## 包装类

对于基础类型来说，java早期的版本就是普通的基础类型(int, double, float - jvm)，之后为了遵循面向对象语言的设计模式，将基本类型以类的形式做了封装。

Number:

byte - Byte

short - Short

long - Long

int - Integer

float - Float

double - Double

Object:

boolean - Boolean

char - Character

特殊类：

void - Void

* + 1. 包装类的作用：

- 作为和基本数据类型对应的类型存在

- 包含每种类型的相关属性，最大值、最小值及一些方法

* + 1. 包装类的用法：

定义：

- 可以和基本类型相同，直接定义

- 可以通过构造器以自身的值为参数进行构造

- 除了Character之外，其他的类型都可以通过String(字符串)进行构造

常用方法：

- parseXxx() 从其他类型转换到当前基本类型

- valueOf() 转换并返回当前类的对象

- xxValue() 转换成对应的基本类型

- toString() 转换成字符串对象

* + 1. 包装类的包装和拆装：

包装：Integer i = 5;

拆装：int i = new Integer(5);

int sum = new Integer(5) + 10;

NOTE: 在做类型转换的时候，要用包装类提供的转换方法，或先进行运算（拆装），再类型转换（要根据需求使用）。

## String类

- Java字符串

* + 1. Java中，定义字符串的方法有三种：

- String str = "123";

- String str = new String("123");

- String str = "" + 123;

Object类中：toString(); equals();

* + 1. 字符串的不可变性

在JVM中，动态开辟的内存空间（一般都在堆内存中），创建字符串对象后，该字符串常量会一直存在，而新创建的String都会指向这个地址而不会新建该字符串。

注：使用new String()的方式会重新创建一个对象，所以对象地址不同

NOTE: String对hashCode()进行了重写，所以使用时要注

意

* + 1. String常用方法

- length(); 当前字符串的长度

- indexOf(); 字符串出现的位置

- subString(); 获取指定位置的字符串[x,x)

- charAt(); 获取指定位置的字符

- trim(); 去除空格

- replace(); 以字符串进行替换

- split(); 以字符串进行分割

- getBytes(); 获取当前字符串的byte数组

- startWith() endWith(); 判断是否以...开头/结尾

- valueOf(); 把指定类型转换为字符串

NOTE: java中使用正则表达式的类是java.util.regex.Pattern和java.util.regex.Pattern

Eg：public class Test {

static String[] strings = { "aaa", "bbb", "ccc" };

public static void main(String[] args) {

// 以(aaa,bbb,ccc,ddd,eee)的形式输出数组

// 再把转出的字符串转回成一个数组

String str = "(";

for (int i = 0; i < strings.length; i++) {

str += (strings[i] + ",");

}

str = str.substring(0, str.length() - 1);

str += ")";

System.out.println(str);

// 转回去

String newStr = str.substring(1, str.length() - 1);

String[] newArray = newStr.split(",");

for (int i = 0; i < newArray.length; i++) {

System.out.println(newArray[i]);

}

}

}

## StringBuilder和StringBuffer类

由于String具有不可变性，对String的操作会重新创建对象，大量的操作String是不可取的，所以java中设计了这两个类来实现字符串的拼接操作。

区别：StringBuffer是线程安全的，相对效率就会变低；而StringBuilder是不安全的，但是效率高。

常用方法：

- length()

- append()

- delete()

- toString()

# Java反射

## newInstance和new

（参考：http://blog.csdn.net/wantianwen/article/details/9532373）

在Java开发特别是数据库开发中，经常会用到Class.forName( )这个方法。通过查询Java Documentation我们会发现使用Class.forName( )静态方法的目的是为了动态加载类。在加载完成后，一般还要调用Class下的newInstance( )静态方法来实例化对象以便操作。因此，单单使用Class.forName( )是动态加载类是没有用的，其最终目的是为了实例化对象。

这里有必要提一下就是Class下的newInstance()和new有什么区别？，首先，newInstance( )是一个方法，而new是一个关键字，其次，Class下的newInstance()的使用有局限，因为它生成对象只能调用无参的构造函数，而使用new关键字生成对象没有这个限制。

好，到此为止，我们总结如下：

Class.forName("")返回的是类

Class.forName("").newInstance()返回的是object

有数据库开发经验朋友会发现，为什么在我们加载数据库驱动包的时候有的却没有调用newInstance( )方法呢？即有的jdbc连接数据库的写法里是Class.forName(xxx.xx.xx);而有一些：Class.forName(xxx.xx.xx).newInstance()，为什么会有这两种写法呢？

刚才提到，Class.forName("");的作用是要求JVM查找并加载指定的类，如果在类中有静态初始化器的话，JVM必然会执行该类的静态代码段。而在JDBC规范中明确要求这个Driver类必须向DriverManager注册自己，即任何一个JDBC Driver的Driver类的代码都必须类似如下：

[java] view plain copy

public class MyJDBCDriver implements Driver {

static {

DriverManager.registerDriver(new MyJDBCDriver());

}

}

既然在静态初始化器的中已经进行了注册，所以我们在使用JDBC时只需要Class.forName(XXX.XXX);就可以了。

Java中工厂模式经常使用newInstance()方法来创建对象，因此从为什么要使用工厂模式上可以找到具体答案。 例如：

class c = Class.forName("Example");

factory = (ExampleInterface)c.newInstance();

其中ExampleInterface是Example的接口，可以写成如下形式：

String className = "Example";

class c = Class.forName(className);

factory = (ExampleInterface)c.newInstance();

进一步可以写成如下形式：

String className = readfromXMlConfig;//从xml 配置文件中获得字符串

class c = Class.forName(className);

factory = (ExampleInterface)c.newInstance();

上面代码已经不存在Example的类名称，它的优点是，无论Example类怎么变化，上述代码不变，甚至可以更换Example的兄弟类Example2 , Example3 , Example4……，只要他们继承ExampleInterface就可以。

从JVM的角度看，我们使用关键字new创建一个类的时候，这个类可以没有被加载。但是使用newInstance()方法的时候，就必须保证：1、这个 类已经加载；2、这个类已经连接了。而完成上面两个步骤的正是Class的静态方法forName()所完成的，这个静态方法调用了启动类加载器，即加载 java API的那个加载器。

现在可以看出，newInstance()实际上是把new这个方式分解为两步，即首先调用Class加载方法加载某个类，然后实例化。 这样分步的好处是显而易见的。我们可以在调用class的静态加载方法forName时获得更好的灵活性，提供给了一种降耦的手段。

最后用最简单的描述来区分new关键字和newInstance()方法的区别：

newInstance: 弱类型。低效率。只能调用无参构造。

new: 强类型。相对高效。能调用任何public构造。

## 类、实例和Class

（参考：http://www.cnblogs.com/flyme/p/4571030.html）

类

类是面向对象编程语言的一个重要概念，它是对一项事物的抽象概括，可以包含该事物的一些属性定义，以及操作属性的方法。面向对象编程中，我们都是以类来编码。

实例

简单理解，就是new，就是对类的实例化，创建这个类对应的实际对象，类只是对事物的描述，而实例化就相当于为这个描述新开辟了一块内存，可以改变这块区域里的各种属性（成员变量），当然，也可以实例化多块区域，只是不同的对象而已。

Class（类的类）

注意这里C大写了，与类概念区分开，在java里，Class是一个实实在在的类，在包 java.lang 下，有这样一个Class.java文件，它跟我们自己定义的类一样，是一个实实在在的类，Class对象就是这个Class类的实例了。在Java里，所有的类的根源都是Object类，而Class也不例外，它是继承自Object的一个特殊的类，它内部可以记录类的成员、接口等信息，也就是在Java里，Class是一个用来表示类的类。（o(∩\_∩)o 有点绕啊，抓住关键一点，Class是一个实实在在的类，可以为它创建实例，也就是本文后面提到的Class对象，也看叫做Class实例）。

　　java提供了下面几种获取到类的Class对象的方法：

　　　　1) 利用对象实例调用getClass()方法获取该对象的Class实例；

　　 2) 使用Class类的静态方法forName("包名+类名")，用类的名字获取一个Class实例

　　 3)运用 类名.class 的方式来获取Class实例；

　　我们知道java世界是运行在JVM之上的，我们编写的类代码，在经过编译器编译之后，会为每个类生成对应的.class文件，这个就是JVM可以加载执行的字节码。运行时期间，当我们需要实例化任何一个类时，JVM会首先尝试看看在内存中是否有这个类，如果有，那么会直接创建类实例；如果没有，那么就会根据类名去加载这个类，当加载一个类，或者当加载器(class loader)的defineClass()被JVM调用，便会为这个类产生一个Class对象（一个Class类的实例），用来表达这个类，该类的所有实例都共同拥有着这个Class对象，而且是唯一的。

总结

　　在java里，类只是信息描述的，写明了有哪些内部属性及接口，你可以理解为是定义了一套规则；而Class对象在java里被用来对类的情况进行表述的一个实例，也就是是类的实际表征，可以理解为是对规则的图表化，这样JVM才能直观的看懂，可以看做是一个模版；而类的实例化对象，就是通过模版，开辟出的一块内存进行实际的使用。

例子：

我们通过一个例子来理解Class实例，为了说明方便，我们新建一个包名深点的类。

新建Name.java（当然，该文件要放在com\dxjia\sample的目录下）

复制代码

1 package com.dxjia.sample;

2

3 public class Name {

4 static int count = 0;

5 static {

6 count++;

7 System.out.println("Name Class Loaded! count = [" + count + "]" );

8 }

9

10 public Name() {

11 System.out.println("Name Constructor called!");

12 }

13

14 }

复制代码

再在根目录新建一个Test主类

复制代码

1 import com.dxjia.sample.Name;

2

3 public class Test {

4 static {

5 Name mName;

6 System.out.println("Test Class loaded");

7 }

8

9 public static void main(String[] args) {

10 System.out.println("entern Test main()");

11

12 // Name.class

13 Class mClassPointClass;

14 // Class.forName("完整包名+类名")

15 Class mClassForName;

16 // new 对象后，对象.getClass()

17 Class mClassObjectPointClass1;

18 Class mClassObjectPointClass2;

19

20 try {

21 //测试 类名.class

22 mClassPointClass = Name.class;

23 System.out.println("mClassPointClass = " + mClassPointClass);

24

25 //测试Class.forName()

26 mClassForName = Class.forName("com.dxjia.sample.Name");

27 System.out.println("mClassForName = " + mClassForName);

28

29 //测试Object.getClass()

30 Name name1 = new Name();

31 mClassObjectPointClass1 = name1.getClass();

32 System.out.println("mClassObjectPointClass1 = " + mClassObjectPointClass1);

33 } catch (ClassNotFoundException e) {

34 // TODO Auto-generated catch block

35 e.printStackTrace();

36 return;

37 }

38

39 Name name2;

40 System.out.println("defined one Name object");

41 name2 = new Name();

42 System.out.println("Name object instance done!");

43

44 mClassObjectPointClass2 = name2.getClass();

45

46 if (mClassForName == mClassPointClass

47 && mClassPointClass == mClassObjectPointClass1

48 && mClassObjectPointClass1 == mClassObjectPointClass2) {

49 System.out.println("all the Class object equal...");

50 }

51 }

52 }

复制代码

分别对他们进行编译：

1 javac com\dxjia\sample\Name.java

2 javac Test.java

执行：

1 java Test

代码中使用了static静态代码块来进行实验，一个类的运行，JVM做会以下几件事情 1、类装载 2、链接 3、初始化 4、实例化；而初始化阶段做的事情是初始化静态变量和执行静态方法等的工作，而且永远只执行一次。

输出结果：

Test Class loaded

entern Test main()

mClassPointClass = class com.dxjia.sample.Name

Name Class Loaded! count = [1]

mClassForName = class com.dxjia.sample.Name

Name Constructor called!

mClassObjectPointClass1 = class com.dxjia.sample.Name

defined one Name object

Name Constructor called!

Name object instance done!

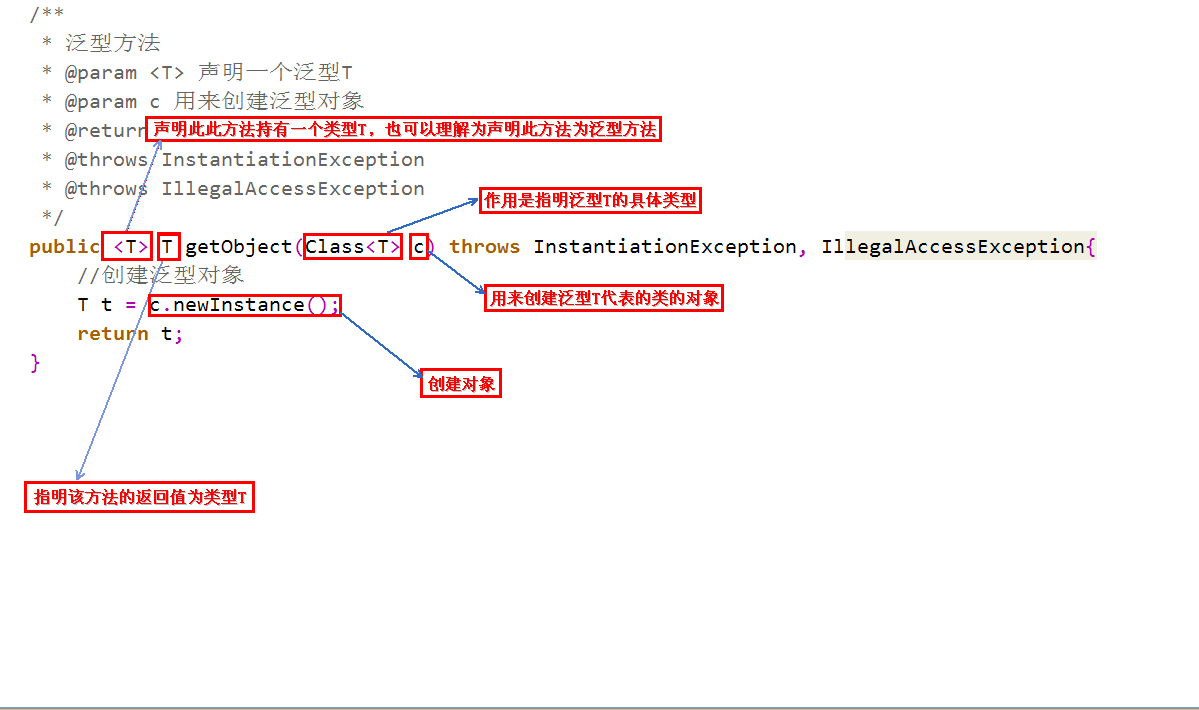
all the Class object equal...

通过结果可以看出在使用 类名.class获得Class实例时，并不会触发类的初始化，而 Class.forName方法就会触发，当然实例化对象肯定也是会触发的，但因为static代码块只执行一次，所以不会再有打印，最后的打印，说明一个类的Class实例只有唯一的一个。

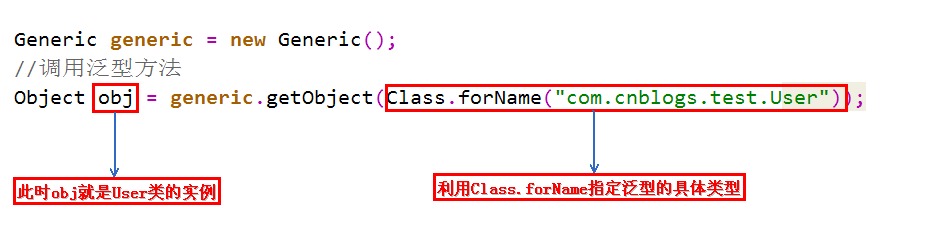
## 泛型

（参考：<http://www.cnblogs.com/iyangyuan/archive/2013/04/09/3011274.html>）

泛型方法：泛型是什么意思在这就不多说了，而Java中泛型类的定义也比较简单，例如：public class Test<T>{}。这样就定义了一个泛型类Test，在实例化该类时，必须指明泛型T的具体类型，例如：Test<Object> t = new Test<Object>();，指明泛型T的类型为Object。但是Java中的泛型方法就比较复杂了。

泛型类，是在实例化类的时候指明泛型的具体类型；泛型方法，是在调用方法的时候指明泛型的具体类型。**定义泛型方法**语法格式如下：

**调用泛型方法**语法格式如下：

说明一下，定义泛型方法时，必须在返回值前边加一个<T>，来声明这是一个泛型方法，持有一个泛型T，然后才可以用泛型T作为方法的返回值。

Class<T>的作用就是指明泛型的具体类型，而Class<T>类型的变量c，可以用来创建泛型类的对象。

为什么要用变量c来创建对象呢？既然是泛型方法，就代表着我们不知道具体的类型是什么，也不知道构造方法如何，因此没有办法去new一个对象，但可以利用变量c的newInstance方法去创建对象，也就是利用反射创建对象。

       泛型方法要求的参数是Class<T>类型，而Class.forName()方法的返回值也是Class<T>，因此可以用Class.forName()作为参数。其中，forName()方法中的参数是何种类型，返回的Class<T>就是何种类型。在本例中，forName()方法中传入的是User类的完整路径，因此返回的是Class<User>类型的对象，因此调用泛型方法时，变量c的类型就是Class<User>，因此泛型方法中的泛型T就被指明为User，因此变量obj的类型为User。

       当然，泛型方法不是仅仅可以有一个参数Class<T>，可以根据需要添加其他参数。

       为什么要使用泛型方法呢？因为泛型类要在实例化的时候就指明类型，如果想换一种类型，不得不重新new一次，可能不够灵活；而泛型方法可以在调用的时候指明类型，更加灵活。

Java反射

-Class类

-静态和动态关系

-反射d方法获取类的信息（方法、成员变量、构造器等）

-反射调用方法和赋值

反射：Java提供的一种动态地访问或获取类的方法或变量（属性）的一种机制。

静态和动态的关系：

静态加载：程序在编译的时候就需要加载的对象

动态加载：程序在运行的时候才需要加载的对象

# Java其它

## Java异常

在Java中，允许抛出和捕获异常。

异常：有异于当前行为状态的、非正常的、有错误出现的，阻止当前方法或作用域正常进行的问题。

1.1 Java中异常有两种：

- Error 一般指JVM的系统异常，虚拟机错误、内存不足等，这种error是api无法捕获的

- Exception 程序中手动或自动（api）抛出的异常

- 运行时异常(RuntimeException) 在运行的时候才可能会产生的异常

e.g.: NullPointerException, ClassCastException等

- 检查异常 编译的时候（编写源程序时）就需要捕获的异常

e.g.: IOException, InterruptedExption等

1.2 try-catch, throw, throws, finally

- try-catch 尝试捕获异常的代码块，在catch中执行捕获到异常后的操作

- throw 抛出一个异常（会阻塞当前程序的运行）

- throws 告知函数可能会抛出的异常

- finally 无论程序try-catch执行成功失败，都会执行finally中的操作

1.3 自定义异常

可以选择性继承一个异常，实现其相关构造、说明(Message)的属性赋值。

1.4 异常链

在执行（捕获）多个异常的时候，由一个try，可以有多个catch进行处理。如果出现嵌套（继承）的关系的时候，按照从下到上的顺序依次抛出，谁能捕获到即停止。父类可以捕获子类的异常。

NOTE: Exception类是所有异常的基类，所以它可以捕获所有的异常。

## Java I/O

I: input, O: output

2.1 文件字符编码

由于自然语言的不同，在通用的编程领域中定义了许多字符集的编码格式。它们是类似于“字典”的形式对文字进行规定，再由不同的API进行获取，转码，从而放入内存。

byte[] bts = 字符串.getBytes("编码格式");

NOTE: Java是双字节编码的形式，使用的是UTF-16be的编码

NOTE: 在windows上创建文本文件的时候，文件的格式ANSI的，而文本的内容格式是不固定的。

2.2 File类

File类在java.io.File，本身是作为文件的定义，包括文件的路径、类型、结构等，File并不能直接读写文件内容。

常用方法：

- 通过构造器构造一个File对象

- exists() 判断文件是否存在

- isDirectory() 判断是否为文件夹

- mkdir() 创建文件夹

- delete() 删除文件

- File.separator 路径分隔符

- createNewFile() 创建文件

- getXxx 可以获取到一些File的基本属性

- listFiles() 得到当前目录下的所有文件

IOStream

- 字节流：处理文本的最小单位

输入流：

- InputStream 应用程序输入数据的方式

- FileInputStream是InputStream的实现类，操作文件用的

read() 每次读一个字节

read(byte[] array) 读一个数组

read(byte[] array, int offset, int len) 从offset开始每次读len长度的字节并放入array中

其他的还有DataInputStream(InputStream), BufferedInputStream(InpuStream),

ObjectInputStream(InputStream)

NOTE: 定义文件结束一般使用-1(EOF)

输出流：

- OutputStream 输出数据的方式

- FileOutputStream 类似FileInputStream，作用相反

write(int b); 每次写一个字节

write(byte[] array); 写一个数组

write(byte[] array, int offset, int len); 从b中的offset开始写入len长度个字节

NOTE: 文件的相对路径，即不加修饰的路径，默认是项目的位置

其他DataOutputStream(OutputStream), BufferedOutputStream(OutpuStream),

ObjectOutputStream(OutputStream)

- 字符流：字符单位

输入流：

- Reader

- InputStreamReader

其他类：FileReader, BufferedReader, PrintReader

输出流：

- Writer

- OutputStreamWriter

其他类：FileWriter, BufferedWriter, PrintWriter

## Java序列化

所有程序的对象操作都是在机器内存中执行、保存的，在应用程序关闭或重启后，当前程序的状态就会丢失。如果需要保存程序的状态或者数据，就需要对程序进行持久化。Java提供的持久化办法就是将对象进行序列化操作。

序列化：将Object（对象）由内存中的实例化对象转换成byte流的过程。通过byte流就可以使用I/O操作对这个对象进行持久化。反之就称为反序列化。

Java中实现序列化操作的接口是java.io.Serializable，这个接口是一个声明式接口。所有需要序列化的程序都必须实现这个接口。（最好是生成一个序列化ID）

- ObjectOutputStream

- ObjectInputStream

3.1 transient 关键字

短暂的，被transient修饰的变量不会被序列化

## Java杂集

泛型是编译时的语法

# Java界面

## Java界面插件

WindowBuilder插件（以下简称“WB”），需要额外安装

用于辅助界面编程

安装使用：<http://java-mans.iteye.com/blog/1639551>

WB插件的作用（类似于VS的窗体）将可视化界面的设置转换成编程代码。

## Java页面布局

<http://www.cnblogs.com/java-class/archive/2012/11/30/2795454.html>

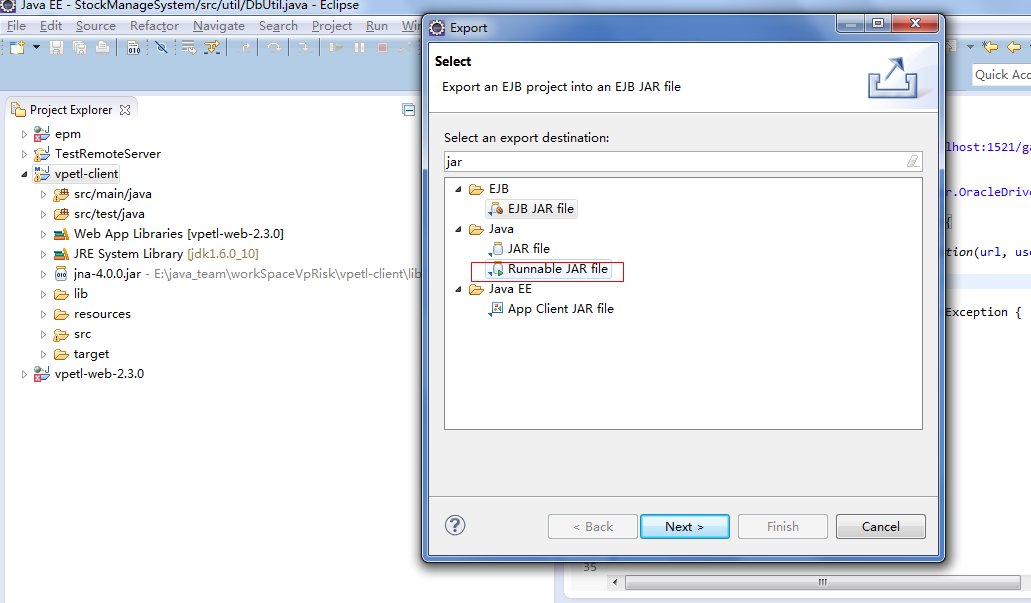
Java页面也可以不使用WB插件直接在代码中编写一个窗体类。

# Java程序

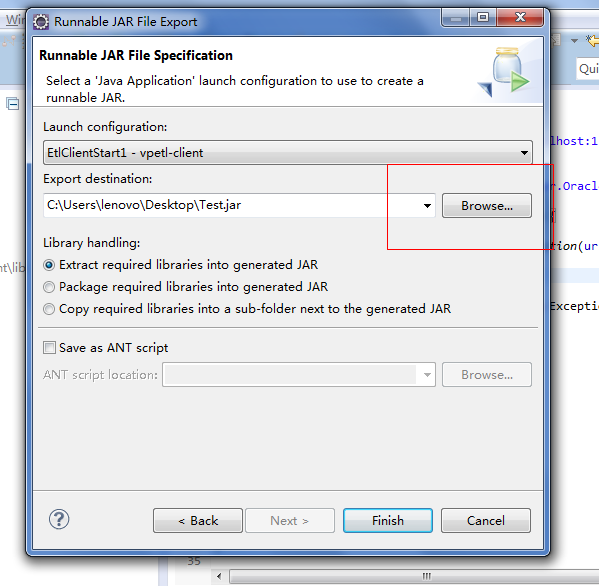
## Java可执行文件

可以引出为”可执行jar“（项目必须有mian方法），然后双击运行即可

第一步：在项目上右击，选择”Export“，之后找到”Runnable Jar file“。

[](https://gss0.baidu.com/-4o3dSag_xI4khGko9WTAnF6hhy/zhidao/pic/item/a9d3fd1f4134970aeceef18790cad1c8a6865dfd.jpg)

第二步：在”Launch“下拉选中选择”main“方法入口类，然后选择”Browse“后，输入”jar“名称，点击”Finsh“即可完成可运行jar制作。

[](https://gss0.baidu.com/9vo3dSag_xI4khGko9WTAnF6hhy/zhidao/pic/item/09fa513d269759ee64338063b7fb43166d22df33.jpg)

注：java项目有依赖第三方Jar包，导出为Runnable JAR file

Java项目没有依赖第三方jar包，导出为JAR file

参考：http://jingyan.baidu.com/article/c843ea0ba4070a77921e4a6c.html

# Java Maven

## 导入Maven

1. 导入Maven项目；
2. 配置repository（reimport）；
3. 配置Configurations（tomcat server）；