# 

尚学堂 高淇 JAVA300集大型基础视频课程

1. 002 计算机语言发展史\_java历史\_java核心优势\_java三大版本的区别

* java核心优势

跨平台

* java三大版本的区别

1. J2EE(JAVAEE)

Java 2 Enterprise Edition

定位于服务端应用

1. J2SE（JAVASE）

Java 2 Standard Edition

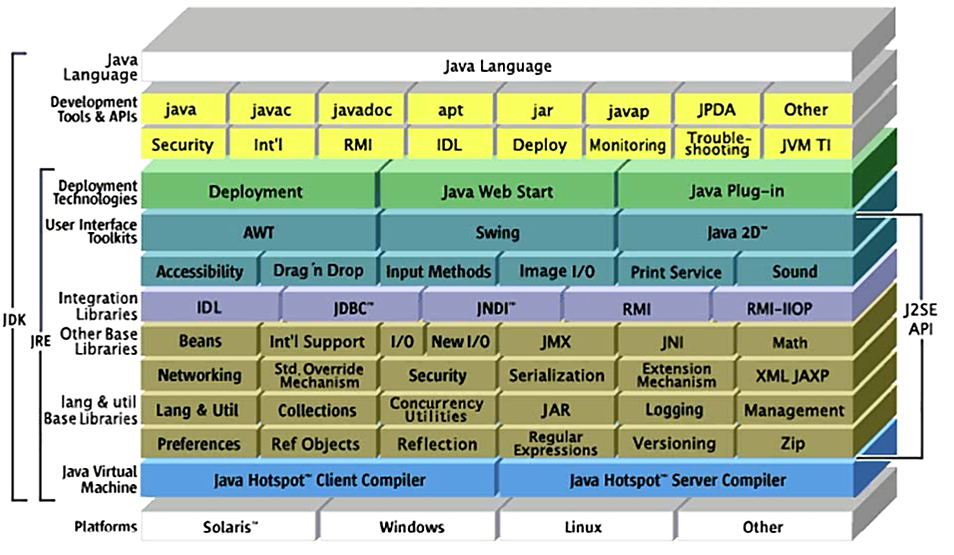
定位于个人计算机上的应用

1. J2ME（JAVAME）

Java 2 Micro Edition

定位在消费性电子产品的应用上

1. 003 JDK和JRE和JVM的区别\_JDK的下载安装\_环境变量配置\_测试安装成功



1. 005 helloworld深化\_文本编辑器的使用\_注释

* 常用dos命令

cd 进入一个目录

cd .. 返回上一层目录

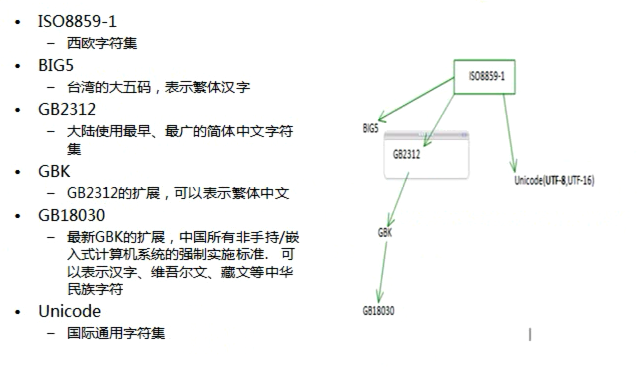
dir 展示目录内容

上下键 查找敲过的命令

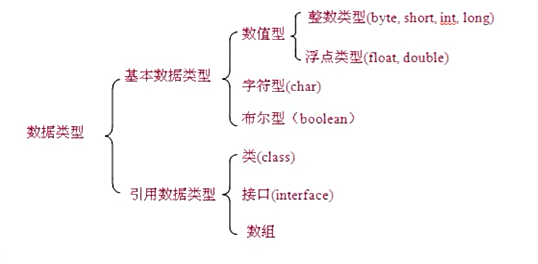
Tab键 命令自动补齐

* /\*\*/ 不支持嵌套注释

1. 006 标识符\_unicode和GBK\_各种字符集的关系



1. 007 整数类型\_进制转换\_编程中的L问题



* Java在java.math包中提供的API类BigDecimal，用来对超过16位有效位的数进行精确的运算。
* 浮点数存在舍入误差，很多数字不能精确表示，如果需要进行不产生舍入误差的精确计算，需要使用BigDecimal。

1. 00? 浮点数\_浮点数误差问题

* float f = 0.1f;

double d = 1.0 / 10;

System.out.println(f == d);

结果为false

1. 011 【补充】JDK7新特性\_二进制整数\_下划线分隔符

* int a = 0b0000\_0000\_0000\_0000\_0000\_0011; //二进制 前面加0b

int b = 3698\_5248; //十进制

1. 012 变量\_成员变量和局部变量\_常量\_命名规范

* final double PI = 3.14; //声明变量
* 命名规范

要求所有变量、方法、类名：见名知意

1. 变量名、方法名

首字母小写&驼峰原则

例：run() runRun() age monthSalary

1. 常量

大写字母和下划线：MAX\_VALUE

1. 类名

首字母大写&驼峰原则

例：Man GoodMan

1. 013 运算符\_01\_算术\_逻辑\_位运算符\_扩展运算符

* << 左移运算法 相当于乘以2

>> 右移运算法 相当于除以2

1. 014 运算符\_02\_字符串连接\_三元运算符\_优先级问题

* 字符串相连：加号两边只要有一个为字符串，则变为字符串连接符，整个结果为字符串。

1. 019 jdk7.0新特性\_增强switch语句

* 增强switch

JDK7之前：表达式结果只能是int(可以自动转换为int的byte、short、char)，枚举类型。

表达式的结果可以是：字符串

1. 023 break和continue\_带标签的break和continue

* 设置标签

标签名: 语句

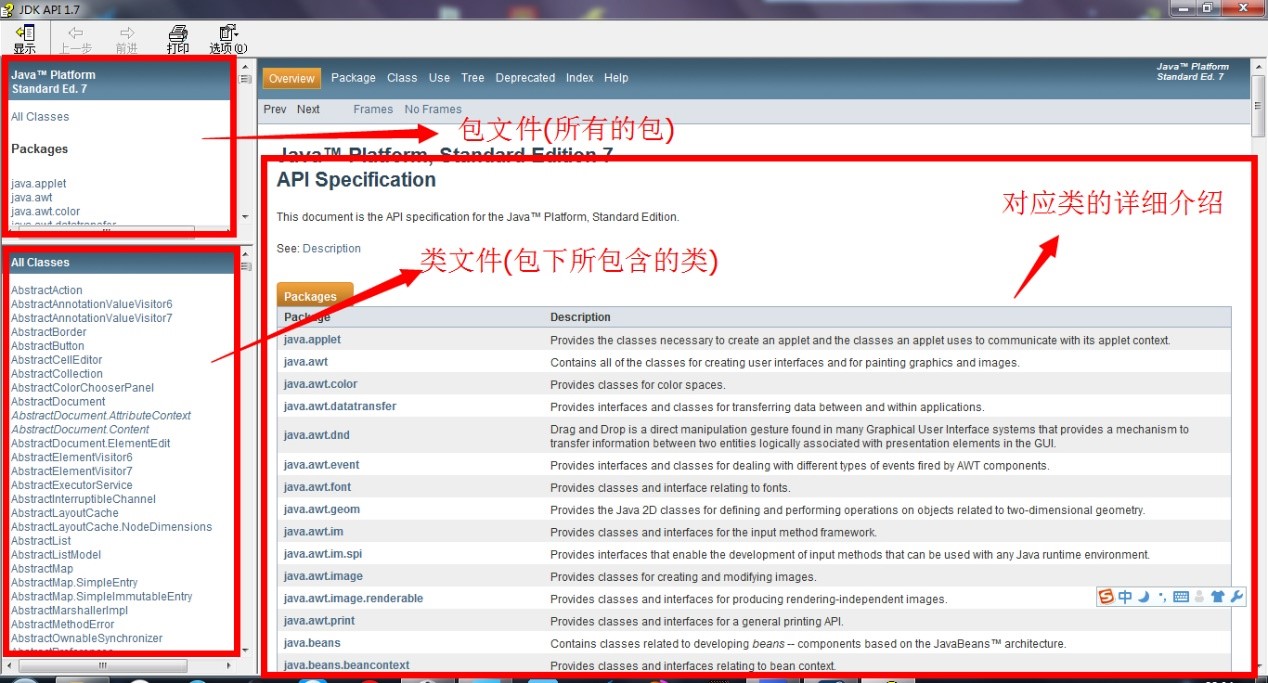
* break和continue可以带标签

1. 024 方法\_方法的本质\_形参\_实参\_return语句

* JAVA只有值传递，没有引用传递！
* 设计方法的时候，最好保持方法的原子性，就是一个方法只完成一个功能，这样利于程序后期的扩展。

1. 026 api文档\_package的概念\_生成自己项目的api文档

* API文档阅读



* API是什么？

Application Programming Interface 应用程序编程接口

* 为什么需要package？

1. 为了解决类之间的重名问题
2. 为了便于管理类：合适的类位于合适的包

* package怎么用？

通常是类的第一句非注释行语句

包名：域名倒着写即可，再加上模块名，并与内部管理类

* 注意事项

1. 写项目是都要加包，不要使用默认包
2. com.gao和com.gao.cao，这两个包没有包含关系，是两个完全独立的包。只是逻辑上看起来后者是前者的一部分

* JDK中主要的包

j**ava.lang**

包含一些Java语言的核心类,如String ,Math, Integer, System和Thread提供常用功能

(核心包中的类可以拿到后直接实用,而不需要导包)

**java.awt**

包含了抽象窗口工具集(abstract window toolkits)的多个类,这些类被用来构建和管理应用程序的图形用户界面(GUI)

**java.net**

包含执行与网络相关的操作的类

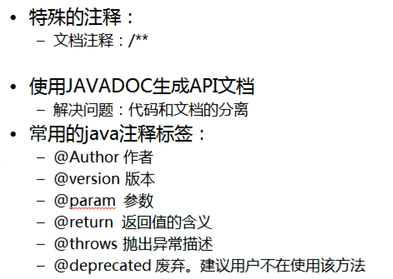
**java.io**

包含能提供多种输入/输出功能的类

**java.util**

包含一些实用工具类,如定义系统特性,实用与日期日历相关的函数.

* 生成自己的API文档



1. 027 Scanner类的使用\_import简单入门

* input的使用

Scanner scan = new Scanner(System.in);

System.out.println("请输入一个加数：");

int a = scan.nextInt();

System.out.println("请输入一个被加数：");

int b = scan.nextInt();

System.out.println("和为：" + (a+b));

1. 031 面向对象\_04\_程序执行过程的内存分析\_01

* Java中除基本类型之外的变量雷翔都称之为引用类型
* Java中的对象是通过引用reference对其操作的
* 类的属性(成员变量)
* 属性是用于定义该类或该类对象包含的数据，或者说静态属性。
* 作用范围：作用于整个类体
* 在定义成员变量时可以对其初始化，如果不对其初始化，Java使用默认值对其初始化。

默认值：

数值:0/0.0, char: \u0000, boolean: false

所有引用类型都是null

* 成员变量有默认值,而局部变量不会有默认值
* 格式:

[修饰符] 属性类名 属性名 = [默认值]

* 方法
* 面向对象中，整个程序的基本单位是类，方法是从属于类的
* 方法的格式：

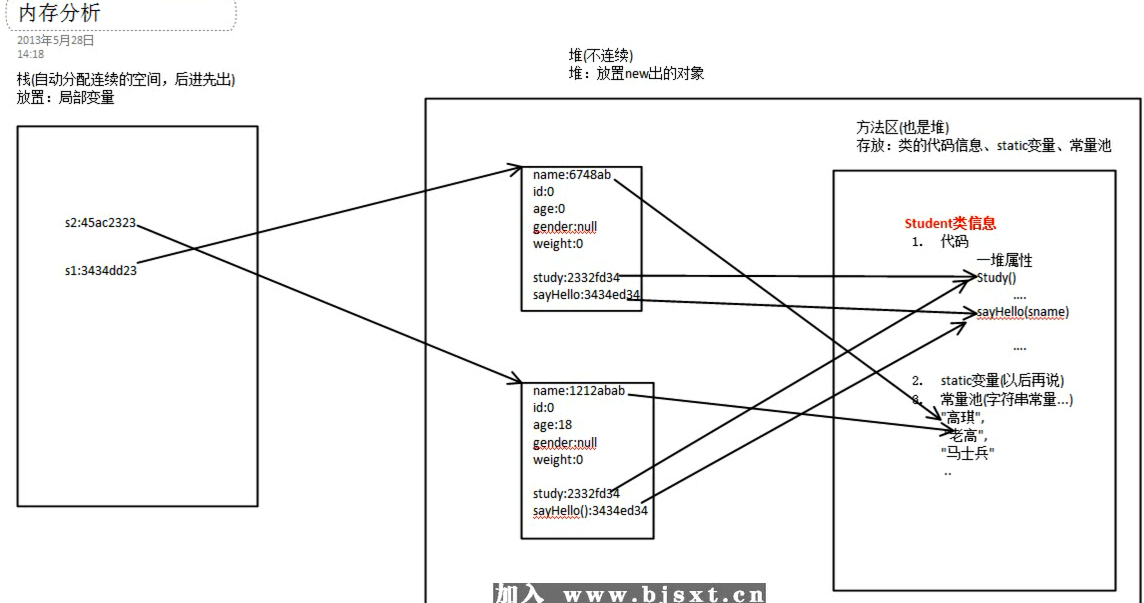
[修饰符] 方法返回类型 方法名(形参列表){

语句

}

* JAVA中方法参数传递是：值传递
* 栈
* 自动分配连续空间，先进先出
* 放置：局部变量
* 堆
* 不连续的空间
* 存放new出来的对象
* **方法区**

存放类的信息(代码)，static变量常量池(字符串常量)等



1. 033 面向对象\_06\_虚拟机内存管理\_垃圾回收机制\_c++和java的比较

* 垃圾回收机制(Garbage Collection)
* 对象空间的分配

使用new关键字创建即可

* 对象空间的释放

将对象赋值NULL即可。垃圾回收器将负责回收所有“不可达“对象的内存空间

* 要点

1. 程序员无权调用垃圾回收器
2. 可通过System.gc()，通知GC运行。但java规范并不能保证立即运行
3. finalize()方法，是JAVA提供给程序员用来释放对象或资源的方法，但是尽量少使用
4. 034 面向对象\_07\_构造方法

* 构造方法也是方法,具有普通方法的特性
* 作用：构造该类的实例
* 格式：

[修饰符] 类名 (形参列表){

//n条语句

}

* 构造方法是一种特殊的方法

1. 调用时用new来调用
2. 构造器虽然有返回值,但是不能定义返回类型(返回值的类型肯定是本类),不能再构造器里调用return
3. 如果我们没有定义构造器，则系统会自动定义一个无参的构造函数
4. 构造器的方法名必须和类名一致
5. 作用：构造该类的对象，经常也用来初始化对象的属性
6. 035 08面向对象\_方法重载\_构造方法重载

* 重载定义

指一个类中可以定义有名字相同但参数不同的方法

* 特点（两同三不同）

同一个类，同一个方法名

参数类型不同、参数个数不同、参数顺序不同

* 注意

形式参数的名字不同、返回值不同时 均不构成重载

构造方法也可以重载

1. 036 09\_static变量和方法\_内存分析static

* 在类中，用static声明的成员变量为静态变量，或者叫做：类属性、类变量
* 它为该类的公用变量，属于类，被该类所有的实例共享，在类被载入时被显式初始化
* 对该类的所有对象来说，static成员变量只有一份，被该类的所有对象共享
* 可以使用“对象.类属性”来调用。不过，一般都是用“类名.类属性”
* static变量置于方法区中
* 用static声明的方法为静态方法
* 不需要对象，就可以调用（类名.方法名）
* 在调用该方法时，不会将对象的引用传递给它，所以在static方法中不可访问非static的成员

1. 037 面向对象\_10\_this隐式参数\_内存分析

隐式参数：在调用方法的时候JAVA会默认传入this参数

这里this有方法的引用

普通方法中， this总是指调用该方法的对象

构造方法中，this总是指正要初始化的对象

this不能用于static方法

1. 038 面向对象\_11\_继承\_基本概念

* 定义

类是对对象的抽象,继承是对某一批类的抽象,从而事项对现实世界更好的建模

* 作用

提高代码的复用性

* extends

extends是”扩展”,子类是父类的扩展

* 重写：override

在子类中写一个与父类中名字相同的方法，覆盖父类中的方法，但处理方式不同

重写后子类依然可以调用父类的方法super.run()；

* 特性
* 子类继承父类，可以得到父类的全部属性和方法（除了父类的构造器）
* JAVA中的类只有单继承，没有多继承。（只能有一个直接父类）

多继承是为了实现代码的复用性，却引入了复杂性，使得系统类之间的关系混乱

* java中的多继承，可以通过接口来实现
* 定义一个类时，没有调用extends，则它的父类是java.lang.Object
* 不同的叫法：超类、父类、基类、子类、派生类

1. 040面向对象\_13\_继承\_Object类用法\_toString和equals方法\_重写toString

* Ctrl + T

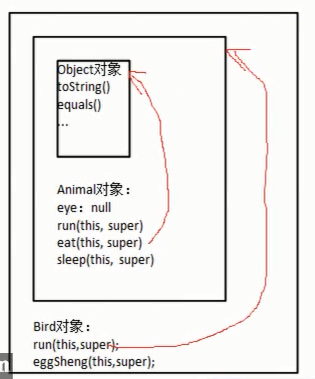
显示该类的继承方式

1. 041面向对象\_14\_继承\_super\_构造器的调用\_继承的内存分析

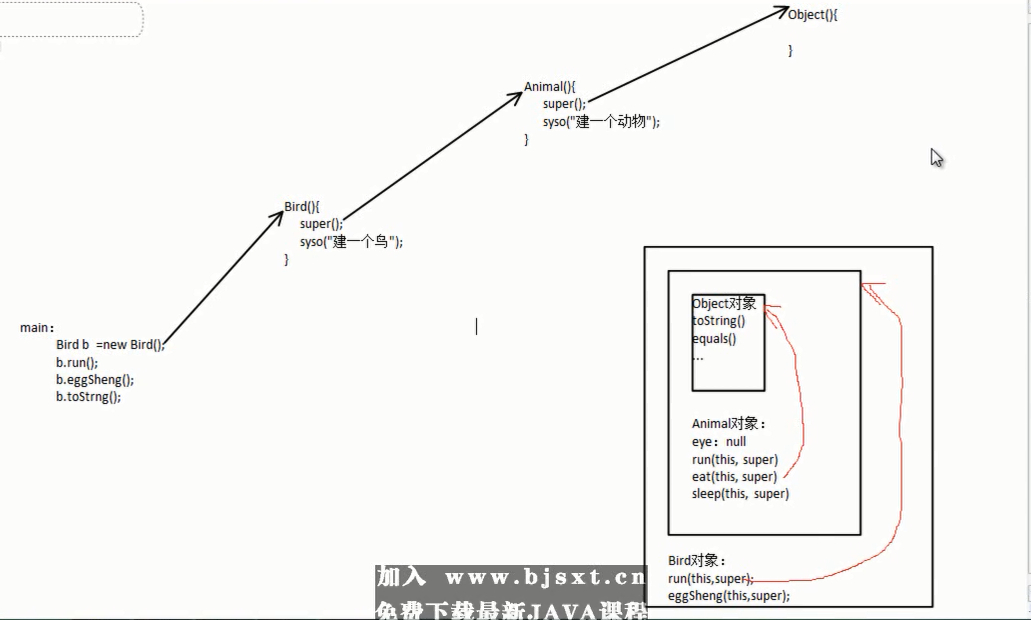
* super
* super是指直接父类对象的引用，可以通过super来访问父类中被子类覆盖的方法或属性
* 构造器中（除了Object）第一句话总是super()，自动调用父类中的构造器
* 每个方法中都有两个隐式参数：this和super

1. this指向当前参数（通过this可以找到自己）
2. super指向父类参数（通过super可以找到父类对象）

* 在内存中有类似包裹的关系



* 继承图像解释



1. 042 面向对象\_15\_继承\_组合

* 组合

定义：把想复用的对象以属性的方式放到现有的方法中，能灵活的实现代码的复用

* 使用选择

is-a关系使用继承

has-a关系用组合

1. 043 面向对象\_16\_final\_修饰变量\_方法\_类

* final关键字

表示最终的定制：不能继承，不能修改不能重写

位置：public后面

* 修饰变量

常量

* 修饰方法

被修饰的方法不能被重写，但是可以被重载

* 修饰类

修饰类事不能被继承

1. 044 面向对象\_17\_封装

* 高内聚，低耦合

高内聚：就是类的内部数据操作细节自己完成，不允许外部干涉

低耦合：仅暴露少量的方法给外部使用

* 权限控制



* 封装的要点
* 类的属性处理

1. 一般使用private（除非本属性确定会让子类继承）
2. 提供相应的get/set方法来访问相关属性，这些方法通常是public，从而提供对属性的读取操作（注意：boolean的get方法使用is开头）
3. 一些只适用于本类的辅助方法可以用private

* 希望其他类调用的方法用public

1. 045 面向对象\_18\_多态\_基本概念\_强制转型问题\_instanceof运算符

* 多态存在的三个必要条件

1. 要有继承
2. 要有方法重写
3. 父类引用指向子类对象

* instanceof

if (cat instanceof Cat) { //如果cat是Cat的实例

((Cat) cat).catchMouse();

}

1. 048 面向对象\_21\_抽象类\_抽象方法

* 关键字：abstract
* 为什么需要抽象类？如何定义抽象类？
* 抽象类是一种模板模式，其为所有的子类提供一个通用的模板，子类可以在这个模板基础上进行扩展
* 通过抽象类，可以避免子类设计的随意性，可以严格限制子类的设计，使子类之间更加通用，同时一个抽象类也可以继承另一个抽象类
* 要点
* 有抽象方法的类只能定义抽象类
* 抽象方法不能被实例化，以及不能用new来实例化抽象类
* 抽象类可以包含属性、方法、构造方法，但是构造方法不能用来new实例，只能通过子类调用
* 抽象类存在的意义就是用来继承
* 抽象方法必须被子类实现
* 抽象类中可以有普通方法
* 意义

将方法的设计与方法的意义实现分离了

1. 049面向对象\_22\_接口详解

* 关键字：interface
* 格式

[访问修饰符] interface 接口名 [extends 父接口1 父接口2 …]{

--常量定义 //总是public static final

--方法定义 //总是public abstract

}

* 注意
* 接口中只有常量和抽象方法
* 接口实现规范和实现的分离，把不变的东西设计成接口
* 接口中的常量默认有：public static final
* 接口中的方法不能是私有，默认为public abstract
* 接口支持多继承
* 子类通过implements来实现接口中的规范
* 接口不能创建实例，但是可用于声明引用变量类型
* 一个类实现了接口，必须实现接口中所有的方法，并且这些方法只能是public的
* 可以在接口名前添加一个字母I表示接口

1. 051 面向对象\_24\_内部类详解

* 定义

一般情况下，我们把类定义成独立的单元，有些情况下，我们把一个类放在两一个类的内部定义，称为内部类

* 作用

1. 内部类提供了更好的封装，只能让外部类直接访问。不允许同一个包中的其他类直接访问
2. 内部类可以直接访问外部类的私有属性，内部类被当成其外部类的成员，但外部类不能访问内部类的属性

* 使用的场合

由于内部类提供了更好的封装特性，并且可以很方便的访问外部类的属性，所以，通常内部类在只为所在外部类提供服务的情况下优先使用

* 分类

1. 成员内部类（可以使用private 、protected、public 任意进行修饰。类文件：外部类$内部类.class）
2. 非静态内部类（外部类里使用非静态内部类和平时使用其他类没有什么不同）
3. 非静态内部类必须寄存在一个外部对象里。因此，如果有一个非静态内部类对象那么一定存在相应的外部类对象
4. 非静态内部类可以使用外部类成员，但是外部类不能直接访问非静态内部类成员
5. 非静态内部类不能有静态方法、静态属性、静态初始化块
6. 静态成员不能访问非静态成员：外部类的静态方法、静态代码块不能访问非静态内部类，包括不能使用非静态内部定义变量、创建实例
7. 成员变量访问要点
8. 内部类里方法的局部变量：变量名
9. 内部类属性：this.变量名
10. 外部类属性：外部类名.this.变量名
11. 内部类的访问
12. 外部类中定义内部类：new InnerClass()

例: new Face()

1. 外部类以外的地方使用非静态内部类

Out.Inner varname = OuterObject.new Inner();

例：Nose n = new Face().new Nose();

1. ………
2. ………
3. 052 数组\_数组基本概念\_内存分析

* 定义
* 数组是相同类型数据的有序集合
* 数组描述的是相同类型的若干个数据，按照一定的先后次序排列组合而成，其中，每个数据称作一个数组元素，每个数组元素可以通过一个下表来访问它们
* 数组的本质就是对象
* 特点

1. 其长度是确定的，数组一旦被创建，它的大小就是不可以改变的
2. 其元素必须是相同类型，不允许数显混合类型
3. 数组中的元素可以是任何数据类型,包括基本类型和引用类型

* 长度的调用

数组对象.length; //其中length不是方法而是属性

1. 053 数组\_数组的三种初始化方式

* 数组的声明

int[] a;

int b [];

* 创建数组对象

a = new int[3];

b = new int[4];

* 数组的初始化

1. 默认初始化：数组元素相当于对象的成员变量，默认值跟成员变量的规则一样，数字0，布尔false，char\u000，引用null
2. 动态初始化：一个个赋值，（使用循环）
3. 静态初始化：int c [] = {23,42,56,94};（长度:4 , 索引范围是[0,3]）
4. 063 常用类\_包装类\_Integer\_Number\_JDK源码分析

* 为什么需要包装类?

JAVA并不是纯面向对象的语言（基本数据类型就不是对象），

实际使用中我们需要对象类型，便于操作。所以我们需要将基本数据类型转化成对象！

包装类均位于java.lang包

|  |  |
| --- | --- |
| 基本数据类型 | 包装类 |
| byte | Byte |
| boolean | Boolean |
| short | Short |
| char | Character |
| **int** | **Integer** |
| long | Long |
| float | Float |
| double | Double |

* 常用方法

//把基本类型转换成包装类

Integer i1 = new Integer(2);

//把String类型转换成包装类

Integer i2 = new Integer("123");

//JDK5.0提供了自动解封箱

//自动封箱

Integer i3 = 3;

//自动解封箱

int i4 = i3;

//把基本类型转换成包装类型

//Integer Integer = Integer.valueOf(i);

Integer i5 = Integer.valueOf(5);

String sN = "123";

//解析字符串为包装类型 基本类型

Integer i6 = Integer.parseInt("sN");

int i7 = Integer.parseInt("sN");

//获取最值

Integer i8 = Integer.MAX\_VALUE;

Integer i9 = Integer.MIN\_VALUE;

//转换成二进制

String s1 = Integer.toBinaryString(i7);

//转换成八进制

String s2 = Integer.toOctalString(i7);

//转换成十六进制

String s3 = Integer.toHexString(i7);

包装类型的一些特点:

Integer a = 1234;

Integer b = 1234;

a == b ---->false //包装类型时对象类型,虽然值相等,但是引用地址不同

a.equals b ------>true

Integer c = 123;

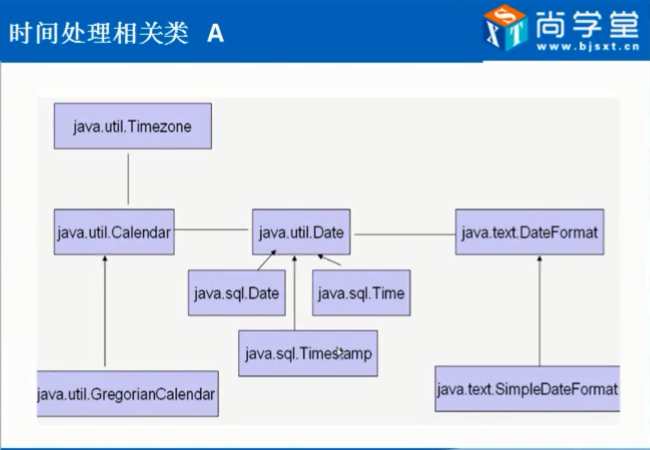
Integer d = 123;

c == d ---->true //当值在[-128,127]之间时,便默认为基本数据类型

c.equals b ------>true

1. 065 常用类\_Date类的使用\_JDk源码分析

* 时间类的相关关系



* Date时间类(java.util.Date)

Java中标示时间是选用一个特定的瞬间,精确到毫秒

从(标准纪元)1970.1.1 0点开始到某个时刻的毫秒数(作为0刻度) 类型时long

常用方法:

getTime()，获得当期时刻的毫秒数

setTime()，设置当前时刻

两个时间的比较

before()

after()

toString()

1. 066 常用类\_DateFormat和SimpleDateFormat\_时间和字符串的互相转换

* 2

1. 1

* 2

1. 1

* 2

1. 1

* 2
* 2

1. 1

* 2

1. 1

* 2

1. 1

* 2
* 2

1. 1

* 2

1. 1

* 2

1. 1

* 2
* 2

1. 1

* 2

1. 1

* 2

1. 1

* 2
* 2

1. 1

* 2

1. 1

* 2

1. 1

* 2
* 2

1. 1

* 2

1. 1

* 2

1. 1

* 2
* 2

1. 1

* 2

1. 1

* 2

1. 1

* 2
* 2

1. 1

* 2

1. 1

* 2

1. 1

* 2

1. 1

* 2

1. 快捷键

Ctrl + / 注释

Alt + / 代码提示

Ctrl + Shift + F 格式化代码

Ctrl + T 显示该类的继承方式

Ctrl + 1 快速修正