java重要基础知识汇总

关于java语言设计理念

抽象类与接口的区别是什么

java为什么不支持多继承

java里的枚举实现机制是什么

java中的内部类

关于类Collections,Arrays,Objects

java里的方法重载

关于java里的main方法

JVM,JDK,JRE三者的关系是什么

关于java语法

continue关键字和break关键字

this关键字

final关键字

transient关键字

Super关键字

return关键字

instanceof关键字

switch语句

关于java中的size()方法和length属性

java基本数据类型介绍

Float和Double详解

四种权限控制关键字

java重要运算符

算术运算符

关系运算符

位运算符

通用知识

递归

关于资源池

什么是SOA

关于java语言设计理念

下面的知识是关于java语言从设计角度出发的一些知识点。

抽象类与接口的区别是什么

首先来看看接口的定义：接口，在JAVA编程语言中是一个抽象类型，主要是抽象方法的集合，接口中的变量定义必须为public static final类型。接口通常以interface来声明。

抽象类: 从面向对象的角度来讲，我们知道所有的对象都是通过类来描绘的，但是反过来却不是这样，并不是 所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描绘一个具体的对象，这样的类就可以认为是抽象类。。抽象类除了不能实例化对象之外，类的其它功能依然存在，成员变量、成员方法和构造方法的访问方式和普通类一样。由于抽象类不能实例化对象，所以抽象类必须被继承，才能被使用。

从定义角度来看，接口和抽象类是两个几乎没有太多联系的设计。接口只是一个抽象方法的集合。而抽象类本质上是一个类，但是它不能被实例化，但是类具备的大多特性抽象类都有。抽象类和interface在Java语言中都是用来进行抽象的，他们除了都是一个用于抽象的东西之外几乎没有任何相同之处。事实上对于一个java里的类来说，无外乎由两种成分组成，即变量和方法（静态代码块可以写在类里面但从实际效果角度并没有影响一个类）。因此接口的功能仅仅包含其中一部分，即方法的集合以及一部分静态成员变量。这样来看，接口是一种非常高的抽象，里面定义的东西被认为是不会改变的。抽象类里面就可以定义普通的成员变量，抽象类的抽象程度相对接口来说会低一点。

但是对于java来说个人认为设计接口的最大原因是为了支持多继承，从这个角度来说，抽象类和接口最重要的区别应该是在使用的时候的区别：

类可以实现多个接口，但是只能继承一个类

其他的区别就非常多了，本质上是一些java语言规则方面的区别。例如：抽象类可以写方法实现，接口也可以写方法实现，不过需要加上default修饰，等等。

问题：什么时候用接口什么时候用抽象类？

抽象类的关键好处在于 能够实现面向对象设计的一个最核心的原则OCP(Open-ClosedPrinciple)。因此当我有一部分内容是不想让子类修改的，但是子类又都通用，同时各个自乐又有自己的特点，那么就适合使用抽象类。

在面向对象领域，抽象类主要用来进行类型隐藏。 我们可以构造出一个固定的一组行为的抽象描 述，但是这组行为却能够有任意个可能的具体实现方式。这个抽象描述就是抽象类，而这一组任意个可能的具体实现则表现为所有可能的派生类。模块可以操作一个 抽象体。由于模块依赖于一个固定的抽象体，因此它可以是不允许修改的；同时，通过从这个抽象体派生，也可扩展此模块的行为功能。熟悉OCP的读者一定知 道，为了能够实现面向对象设计的一个最核心的原则OCP(Open-Closed Principle)，抽象类是其中的关键所在。

符合开发封闭原则，我可以对抽象出来的类进行扩展，但是只要是这个抽象类的子类，那么他必然能够。

从语法层面上讲，java单继承多实现，而接口可以多实现。

java为什么不支持多继承

典型的支持多继承的语言就是C++。在OOP的世界里，单根继承意味着所有的类都会有一个最上面的终极类，java里面这个类就是Object。单根继承既可以说是一门语言的特性，也可以说是一门语言的一个选择。从纯粹技术的角度来说，java也可以做到多继承，只是如果那样的话那么java就不会再是我们今天所认识的java。除此之外，单根继承还有下面这些优点：

单根继承的优点1:兼容性

单根继承带来的一个效果就是所有的对象归根到底都是相同的基本类型。这带来的好处就是任何java出现的新类库中，兼容性的问题会大大降低，这一点很好理解。但是在C++之中，总是会有一些不兼容的接口，这虽然带来了一定的灵活性，但是对于不兼容的接口，往往就是要通过多继承来解决。

单根继承的优点2: 便利性

因为单根继承，所有的对象都会具备某些一样的功能，比如所有的对象都会有hashcode方法，有euqals方法。因此拿到一个对象时，无论这个对象从哪里来，我们都知道可以对他执行某些基本操作。参数传递也得到了简化。

单根继承的优点3: 垃圾回收

单根继承会使得垃圾回收变得简单很多。因为所有对象都保证具有其类型信息，因此不会因为无法确定类型信息而带来不便。垃圾回收正是java相对于C++的重要改进之一。

java里的枚举实现机制是什么

枚举类型在编译器处理之后，是由一个final的继承Enum类的类实现的。该类是一个实实在在存在的类。在该类当中，编译器还帮助我们生成了每个枚举类型的实例对象，这些对象分别对应枚举中定义的每个枚举类型本身。

java中的内部类

定义：将一个类的定义放在另一个类的定义内部，即为内部类。

内部类本质上是java的一种"语法糖"。为什么这样说呢？举例说明，假设现在有如下代码：

public class A {

private int a;

static class B {

}

class C {

public void test(){

int b = a;

}

}

}

类A是一个普通的类，在他的内部定义了两个类B，以及C。从代码结构上来看，B类和C类为A类的内部，但是在使用编译器编译之后，它们并不是一个类，而是会变成符合一定名称规则的三个类，如下图所示：

它在编译之后会产生三个.class文件，分别是：A.class, A$B.class, A$C.class. 因此，本质上它们还是三个类，只是借助于java编译器的语法糖支持，我们可以写在一个类里面，从这个例子我们不难推断出，在java 里任何一个类，无论是以怎样的形式定义，在编译之后生成字节码文件之后，其必然是一个单独存在的类。理解java的类加载机制的话对这句话理解起来就更加容易，java加载任何一个类的时候都是会首先从加载其class文件开始，若一个类不存在对应的class文件，那么它必然无法被加载也无法被使用。

同时，知道了上述知识之后，我们来看这样两个问题：

内部类可以被继承吗？答案是肯定可以的，只是从java语法来说写起来会稍微有点区别

内部类的方法可以被覆盖吗？答案一样是可以的。

为何java编译器会支持定义内部类这样的使用方式，原因在于当一个类定义在另一个类内部之后，许多操作会变得简单一些，比如一个内部类可以直接访问外部类的任何成员。为什么内部类能直接访问外部类的任何成员呢？原因在于java编译器对内部类的功能t提供了支持，让我们再来看上述代码反编译回来的结果：

public class A {

private int a;

public A() {

}

class C {

C() {

}

public void test() {

int b = A.this.a;//通过类名加this关键字

}

}

static class B {

B() {

}

}

}

可以看到是通过类名跟上.this关键字实现的对外部成员的访问，这相当于是隐式的持有了一个外部类引用，即建立了一个内部类和外部类之间的联系。

同时，这里我们需要注意这里的B类声明成了static的类，C类则没有。我们常常把用static修饰的内部类成为嵌套类。

他们的区别在于，嵌套类与外围类之间是没有联系的。这意味这创建嵌套类无需外部类，当然，也不能从嵌套类的对象中访问非静态的外围类对象。同时，在嵌套类的内部可以使用static关键字，而普通的内部类不能使用static关键字。

用途：从代码的组织结构来说，使用内部类可以把逻辑相关的类组织在一起。内部类访问外部类将非常方便，内部类能访问外围对象的所有成员，且不需要任何特殊条件。但这不是最主要的原因，从设计角度出发，使用内部类最大的原因在于：每个内部类可以独立但继承一个类，这意味着虽然java的类是单继承的，但是通过使用内部类，可以达到类似多重继承的效果。因此，如果不需要解决多重继承的问题，使用内部类就并不是必须的了，因为其他的编码方式都能实现一样的效果。

关于类Collections,Arrays,Objects

在jdk源代码中提供了很多有用的工具类，它们的命名也有一定的规律。

Collections类提供了很多给容器使用的实用方法。

Arrays类提供了很多给给数组容器有用的方法。例如想创建一个数组对象可以直接调用方法Arrays.asList(...)

Objects类提供了一些给Object类中的实用方法.

这些类都位于jdk的java.util包下面。除了上面提到的三个类以外，util包下面还有许多非常有用且也经常被用到的类和包，例如正则表达式相关的类，基本类型转换的类，以及流编程的类等等，感兴趣的读者可以自行查看。

java里的方法重载

所谓方法重载，指的是当两个方法具有相同的方法名称的时候，他们共存的一种方式。下面是两个重载方法的例子:

public class TestOverloading {

public void method(String s){

}

public void method(int i){

}

}

对于方法名称相同的：

参数数量一样，类型不一样，可以重载

参数数量不一样可以重载

参数数量一样，相同位置类型不一样，可以重载

对于第三种情况,实例代码如下：

public class TestOverloading {

public void method(String s, int i){

}

public void method(int i, String s){

}

}

注意，方法返回值不作为方法标识之一，下列方法是不能重载的

public class TestOverloading {

public int method( int i){

return 1;

}

public String method(int i){

return "2";

}

}

关于java里的main方法

所有的java程序无论大小，只要能运行必定有一个main方法，关于java的main方法的分析请参考博客：[java] 关于main方法的一切。

JVM,JDK,JRE三者的关系是什么

JVM指的是java virtual machine, 即java虚拟机。

是运行java代码的引擎。在其他主要的编程语言中，代码的编译器都只会为某个特定系统生成编译之后的代码，也就是说这部分代码只能在特定的系统上执行。而java编译器不为特定的平台生成字节码，而是为Java虚拟机生成字节码，编译生成的字节码是可用于在任何平台上运行的源代码，因此jvm成为了将Java代码编译为字节码的媒介，它在不同的机器上进行解释，使的java语言与平台/操作系统独立。字节码相当于是Java源代码和主机系统之间的中介语言。

一个java虚拟机主要做了下面几件事：

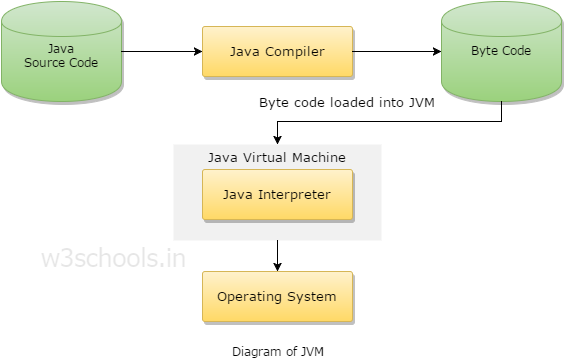
阅读字节码。

验证字节码。

将代码与库链接。

可以用下图解释jvm主要做的事情

Diagram of JVM



JDK指的是Java SE Development Kit，即java se开发库。

JDK包括完整的JRE（Java运行时环境）以及用于开发，调试和监视Java应用程序的工具（例如jconsole等）。 JDK是构建和运行Java应用程序和applet所必需的。 它可以被视为一个套件， 程序员和开发人员大多使用它。

JRE指的是Java runtime environment，即java运行时环境。

JRE代表Java Runtime Environment，用于在运行时提供环境。 它是JVM能够支持多平台的原因。 它包含一组库以及JVM在运行时使用的核心类和各种其他文件。 JRE是JDK（Java Development Toolkit）的一部分，但可以单独下载。

JRE由各种组件组成，如下：

Java Web Start和Java Plug-in。

用户界面工具包，包括抽象窗口工具包（AWT），Swing，图像输入/输出，辅助功能，拖放等。

其他不同的基础库，包括输入/输出，扩展机制，bean，JMX，JNI，网络，覆盖机制等。

Lang和util库，管理，版本控制，集合包等等。

集成库，包括接口定义语言（IDL），Java数据库连接（JDBC），Java命名和目录接口（JNDI），远程方法调用（RMI）。

关于java语法

下面知识点是关于java编程语法，比如关键字，循环结构等知识的介绍。

continue关键字和break关键字

continue关键字

使用地方：continue关键字只能用于循环结构。

作用: 跳过本次循环，重新开始下一趟循环。

例子：

public class HelloWorld {

public static void main(String[] args) {

//打印单数

for (int j = 0; j < 10; j++) {

if(0==j%2)

continue; //如果是双数，后面的代码不执行，直接进行下一次循环

System.out.println(j);

}

}

}

打印结果是：1 3 5 7 9。

break关键字

使用地方：用于switch结构和循环结构

作用:

1.如果用于switch结构，跳出当前的case语句

2.如果用于循环结构中，跳出当前循环结构。

例子：

public class HelloWorld {

public static void main(String[] args) {

//打印单数

for (int j = 0; j < 10; j++) {

if(0==j%2)

break; //如果是双数，直接结束循环

System.out.println(j);

}

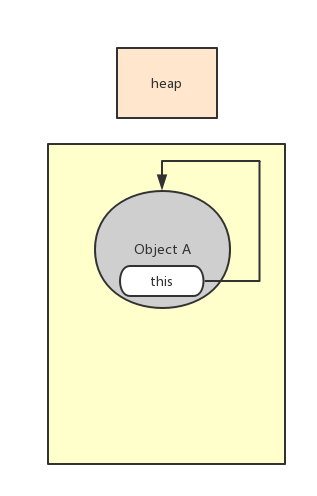
}

}

打印结果是：什么都不打印

this关键字

this关键字代表对象自身，如果new了一个对象之后，则这个this就是一个指向这个对象自己的一个引用，如下图：



根据上述描述，我们知道this是用于在方法内部获取当前对象的引用的关键字，其用法与其他引用并无不同。

用途：

this调用本类中的属性，也就是类中的成员变量

this调用本类中的其他方法；

this调用本类中的其他构造方法，调用时要放在构造方法的首行。

这几点都非常好理解，因为指向的是对象自身，能操作的自然都是成员变量以及类的方法。有了这些知识之后，我们再来看一个java代码里经常看到的构造方法写法：

public class Test{

private String s;

public Test(String s){

this.s = s;

}

}

初学者常常对这里有两个s感到困惑，现在我们就知道了，等号左边的this.s获取的是当前对象即外部的s字符串，而等号右边的s则必然是括号里的参数，因此这就把括号里的参数赋值给了成员变量s。this常常用于这种成员变量名字和方法参数一样的情形。

最后，我们来扩展一点题外话，关于static关键字。理解了this关键字之后，static关键字就能更加全面的被理解。static方法就是没有this的方法。注意这里需要仔细理解这个没有的意思。为什么说没有，是因为在static方法内部不能调用非static的方法，但反过来是可以的，且不需要创建对象。因此static方法很很像全局方法，关于static的用法，请参考另一篇单独的博客：[java] static关键字总结.

final关键字

final是java的一个关键字，他可以用于修饰类，方法，变量。此关键字的作用在于表明它所修饰的内容是不可改变的。

1 修饰变量

final顾名思义是最终的意思，他修饰变量意味着这个变量的值不能再更改。一个即是static又是final的常量会占据一段不能改变的内存空间。对于这样的常量，我们的命名规范是约定俗成的大写字母加下划线，如下：

private static final int TEST\_FINAL = 1;

在修饰引用变量的时候，这个值是指的其引用不变，但引用的对象的内容是可变的。而在修饰基本变量如int时，则意味着这个基本类型的值不能改变。值得注意的是,final在修饰String类的对象时，其值和修饰基本变量类型是一样不能改变的。答案可以在这篇博客中找到：[Java] String类深度解析。这里不再展开。

2 修饰方法

final修饰的一个方法代表这个方法可以被继承，但是不可以被子类重写。一般对一个方法加上final关键字作为修饰，代表这个方法你认为不需要再重写和修改，而是想要确保在继承中方法的行为保持不变。

在过去，建议使用final 方法的另一个原因在于效率，final方法是在程序编译的时候就静态绑定了，但是在如今的jvm中这个速度已经可以忽略不计了。并且，也不再建议出于效率目的使用final。

此外，一个类中所有的private方法都是隐式的被指定为final 的。因此对private方法添加final关键字没有任何意义。private方法不能被取用，因此也无法覆盖。

3 修饰类

final类不能被继承，其修饰的类功能通常是完整的。

与finally区别？finally 通常用在异常处理中。异常处理可以参考这篇文章：java异常机制详解。

与finalize区别？finalize是在Object类中定义的方法，是在垃圾回收时虚拟机调用来终结对象的方法。

transient关键字

我们都知道一个对象只要实现了Serilizable接口，这个对象就可以被序列化，java的这种序列化模式为开发者提供了很多便利，我们可以不必关系具体序列化的过程，只要这个类实现了Serilizable接口，这个类的所有属性和方法都会自动序列化。

然而在实际开发过程中，我们常常会遇到这样的问题，这个类的有些属性需要序列化，而其他属性不需要被序列化，打个比方，如果一个用户有一些敏感信息（如密码，银行卡号等），为了安全起见，不希望在网络操作（主要涉及到序列化操作，本地序列化缓存也适用）中被传输，这些信息对应的变量就可以加上transient关键字。换句话说，这个字段的生命周期仅存于调用者的内存中而不会写到磁盘里持久化。

总之，java 的transient关键字为我们提供了便利，你只需要实现Serilizable接口，将不需要序列化的属性前添加关键字transient，序列化对象的时候，这个属性就不会序列化到指定的目的地中。关于此关键字需注意下面三点：

一旦变量被transient修饰，变量将不再是对象持久化的一部分，该变量内容在序列化后无法获得访问。

transient关键字只能修饰变量，而不能修饰方法和类。注意，本地变量是不能被transient关键字修饰的。变量如果是用户自定义类变量，则该类需要实现Serializable接口。

被transient关键字修饰的变量不再能被序列化，一个静态变量不管是否被transient修饰，均不能被序列化。

Super关键字

主要存在于子类方法中，用于指向子类对象中父类对象。其作用包括下面几种

访问父类的属性

访问父类的函数

访问父类的构造函数

return关键字

return 关键字在java里面有两方面用途：

指定一个方法返回什么值

退出当前方法，并返回指定的值

对于返回值为void的方法，我们可以不写return语句，原因在于结尾处会生成一个隐式的return。

instanceof关键字

Instanceof关键字的作用是返回一个布尔值，告诉我们一个对象是不是某个特定类的一个实例。例如:

if(x instanceof Dog){

}

即在判断x对象是不是Dog类的一个实例。

Instanceof关键字是RTTI，动态类型判断技术的其中一种表现形式。关于RTTI的更多内容，请参考这篇博客:[java] 反射和多态实现原理详解以及对比.

switch语句

switch可以看作是一种选择语句。它的作用是根据表达式的值，从一系列代码中选择一段去执行。其格式如下：

switch(selector){

case value1:

statement; break;

case value2:

statement; break;

case value3:

statement; break;

default:

statement;

}

其中selector指的是一个参数，或是一个表达式，或是一个方法。在java5之前，它必须是一个能返回整数或者char类型的表达式。在java5当中，引入了对枚举Enum类型支持，在java7当中，switch以及可以匹配字符串了，下面是关于枚举和字符串的示例代码：

package basic;

public class TestSwitch {

public enum Color { RED, GREEN, BLANK, YELLOW }

public void testSwitch(){

String s= "0";

switch(s){

case "0":

; break;

case "1":

; break;

case "2":

; break;

default:

;

}

switch(Color.GREEN){

case RED:

; break;

case GREEN:

; break;

case BLANK:

; break;

case YELLOW:

; break;

default:

;

}

}

}

关于java中的size()方法和length属性

在编程时经常会用到的两个东西，老是记混，因此特地看了一下。

length是数组的一个属性，在获取值的时候是按属性的方法获取。

而size()是链表的一个方法，用于获取链表的长度

java基本数据类型介绍

简单类型

boolean

byte

char

short

Int

long

float

double

void

二进制位数

1

8

16

16

32

64

32

64

--

封装器类

Boolean

Byte

Character

Short

Integer

Long

Float

Double

Void

byte：

byte 数据类型是8位、有符号的，以二进制补码表示的整数；

最小值是 -128（-2^7）；

最大值是 127（2^7-1）；

默认值是 0；

byte 类型用在大型数组中节约空间，主要代替整数，因为 byte 变量占用的空间只有 int 类型的四分之一；

例子：byte a = 100，byte b = -50。

short：

short 数据类型是 16 位、有符号的以二进制补码表示的整数

最小值是 -32768（-2^15）；

最大值是 32767（2^15 - 1）；

Short 数据类型也可以像 byte 那样节省空间。一个short变量是int型变量所占空间的二分之一；

默认值是 0；

例子：short s = 1000，short r = -20000。

int：

int 数据类型是32位、有符号的以二进制补码表示的整数；

最小值是 -2,147,483,648（-2^31）；

最大值是 2,147,483,647（2^31 - 1）；

一般地整型变量默认为 int 类型；

默认值是 0 ；

例子：int a = 100000, int b = -200000。

long：

long 数据类型是 64 位、有符号的以二进制补码表示的整数；

最小值是 -9,223,372,036,854,775,808（-2^63）；

最大值是 9,223,372,036,854,775,807（2^63 -1）；

这种类型主要使用在需要比较大整数的系统上；

默认值是 0L；

例子： long a = 100000L，Long b = -200000L。

"L"理论上不分大小写，但是若写成"l"容易与数字"1"混淆，不容易分辩。所以最好大写。

float：

float 数据类型是单精度、32位、符合IEEE 754标准的浮点数；

float 在储存大型浮点数组的时候可节省内存空间；

默认值是 0.0f；

浮点数不能用来表示精确的值，如货币；

例子：float f1 = 234.5f。

double：

double 数据类型是双精度、64 位、符合IEEE 754标准的浮点数；

浮点数的默认类型为double类型；

double类型同样不能表示精确的值，如货币；

默认值是 0.0d；

例子：double d1 = 123.4。

boolean：

boolean数据类型表示一位的信息；

只有两个取值：true 和 false；

这种类型只作为一种标志来记录 true/false 情况；

默认值是 false；

例子：boolean one = true。

char：

char类型是一个单一的 16 位 Unicode 字符；

最小值是 \u0000（即为0）；

最大值是 \uffff（即为65,535）；

char 数据类型可以储存任何字符；

例子：char letter = 'A';。

Float和Double详解

关于这两个数据类型，看过很多资料和博客也没有搞明白，博客质量也参差不齐。Float和Double就是java对于单精度和双精度提供的两种支持。

Double

事实上所谓的一个Double类型的数就是一个双精度类型的数。

双精度类型数据：Double precision data。 双精度型（DOUBLE）数据是具有更高精度的一种数据型数据。

双精度型数据采用固定长充浮点格式存储，占用8个字节.在计算机中每个双精度型数据占用8个字节（64位）的存储空间，可表示的正数范围是：4.94065645841247\*10^-324~1.79769313486232\*10^308，可表示的负数范围是：-1.79769313486232\*10^308~-4.94065645841247\*10^-324。双精度型数据最多可以有15位有效数字

Float

与双精度对应的就是单精度，单精度。

单精度数是指计算机表达实数近似值的一种方式。它的范围在负数的时候是从 -3.402823E38 到 -1.401298E-45，而在正数的时候是从 1.401298E-45 到 3.402823E38

内存只分配32位,双精度分配64位内存,所以说双精度的精确度更高,但占用的内存也大,像金钱什么的要高度精确的就用它。

我们也常常把单精度类型数据叫做浮点类型的数。

下面内容为引用：

记住java一定要用double，就算数值不大也要用double。

了解java虚拟机的底层会知道，float放在内存中其实是当作double来处理的，它不会比double更节约内存资源，对应的double虚拟机会直接以double形式来进行处理，快速而且精度高，但是如果用float，不但不会节约内存资源，虚拟机为了校验float的精度，会花费更多的系统资源，例如cpu时钟，程序执行步骤等等。

相对于这点，整数类型，能用int就用int，不要用什么short类型，道理是一样，其实虚拟机中short,char,boolean,byte在内存中都是以int形式来处理的，为了校验精度，虚拟机还会付出格外的开销，这样其实得不偿失，不要自作聪明以为节约了内存，其实错了。当然long类型例外，虽然long类型也会增加资源的开销，但是毕竟能完成int完成不了的功能。

还有，其实这些资源的开销对于整个应用程序和现有硬件资源而言就是九牛一毛，微乎其微，没有必要过于在意。就用习惯的形式即可。不要自作聪明的用特别的数据类型，浮点就double，整形就int，长整型就long，其它的必要性都不大(byte的话，用来做数组还是很方便的，除此不推荐使用。

关于精度：https://blog.csdn.net/a327369238/article/details/52354811/

四种权限控制关键字

修饰词 本类 同一个包的类 继承类 其他类

private √ × × ×

无（默认） √ √ × ×

protected √ √ √ ×

public √ √ √ √

其中，protected关键字的用途主要在于，有些东西对外部是隐藏的，但是对于他们的导出类是可见的，关键字protected就是起这个作用的。

java重要运算符

算术运算符

假设整数变量A的值为10，变量B的值为20：

操作符 描述 例子

+ 加法 - 相加运算符两侧的值 A + B 等于 30

- 减法 - 左操作数减去右操作数 A – B 等于 -10

\* 乘法 - 相乘操作符两侧的值 A \* B等于200

/ 除法 - 左操作数除以右操作数 B / A等于2

％ 取余 - 左操作数除以右操作数的余数 B%A等于0

++ 自增: 操作数的值增加1 B++ 或 ++B 等于 21（区别详见下文）

-- 自减: 操作数的值减少1 B-- 或 --B 等于 19（区别详见下文）

关系运算符

实例整数变量A的值为10，变量B的值为20：

运算符 描述 例子

== 检查如果两个操作数的值是否相等，如果相等则条件为真。 （A == B）为假(非真)。

!= 检查如果两个操作数的值是否相等，如果值不相等则条件为真。 (A != B) 为真。

> 检查左操作数的值是否大于右操作数的值，如果是那么条件为真。 （A> B）非真。

< 检查左操作数的值是否小于右操作数的值，如果是那么条件为真。 （A <B）为真。

>= 检查左操作数的值是否大于或等于右操作数的值，如果是那么条件为真。 （A> = B）为假。

<= 检查左操作数的值是否小于或等于右操作数的值，如果是那么条件为真。 （A <= B）为真。

位运算符

Java定义了位运算符，应用于整数类型(int)，长整型(long)，短整型(short)，字符型(char)，和字节型(byte)等类型。

位运算符作用在所有的位上，并且按位运算。假设a = 60，b = 13;它们的二进制格式表示将如下

A = 0011 1100

B = 0000 1101

-----------------

A&b = 0000 1100

A | B = 0011 1101

A ^ B = 0011 0001

~A= 1100 0011

整数变量A的值为60和变量B的值为13

操作符 描述 例子

＆ 如果相对应位都是1，则结果为1，否则为0 （A＆B），得到12，即0000 1100

| 如果相对应位都是0，则结果为0，否则为1 （A | B）得到61，即 0011 1101

^ 如果相对应位值相同，则结果为0，否则为1 （A ^ B）得到49，即 0011 0001

〜 按位取反运算符翻转操作数的每一位，即0变成1，1变成0。 （〜A）得到-61，即1100 0011

<< 按位左移运算符。左操作数按位左移右操作数指定的位数。 A << 2得到240，即 1111 0000

>> 按位右移运算符。左操作数按位右移右操作数指定的位数。 A >> 2得到15即 1111

>>> 按位右移补零操作符。左操作数的值按右操作数指定的位数右移，移动得到的空位以零填充。 A>>>2得到15即0000 1111

着重说一下一下几个

（1）<<

A为60，60对应的二进制为 0011 1100或者11 1100

A<<2即60 << 2即 0011 1100左移2位 得：1111 0000 。 换算成十进制就是240。

在这里我们可以得出移位运算的思路就是：把一个数化成对应的二进制，然后在进行对应的移位操作，再将移位后的二进制换算成需要的其他进制即可。

（2）>>

0011 1100 右移两位得: 0000 1111 。换算成十进制就是15.

（3）～

～A 即 0011 1100取反，得 1100 0011.换算成十进制就是-61

通用知识

下面是一些跨编程语言的知识。

递归

1 定义：程序调用自身的编程技巧称为递归（ recursion)

特点：一个过程或函数在其定义或说明中有直接或间接调用自身的一种方法，它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。递归的能力在于用有限的语句来定义对象的无限集合

2 组成

边界条件

递归前进段

递归返回段

当边界条件不满足时，递归前进；当边界条件满足时，递归返回。

3 条件

子问题须与原始问题为同样的事，且更为简单；

不能无限制地调用本身，须有个出口，化简为非递归状况处理

4 生活中递归的表现

德罗斯特效应是递归的一种视觉形式。图中女性手持的物体中有一幅她本人手持同一物体的小图片，进而小图片中还有更小的一幅她手持同一物体的图片，依此类推。

我们在两面相对的镜子之间放一根正在燃烧的蜡烛，我们会从其中一面镜子里看到一根蜡烛，蜡烛后面又有一面镜子，镜子里面又有一根蜡烛……这也是递归的表现

从小就听过的例子：从前有座山，山里有座庙，庙里有个和尚，和尚在讲故事，从前有座山，山里有座庙，庙里有个和尚，和尚在讲故事，从前有座山...

但是上述例子是和递归相似的场景，但是软件当中定义的递归是要有一个终止条件的，否则就是死循环了。

5 递归算法一般用于解决三类问题

数据的定义是按递归定义的。（Fibonacci函数）

问题解法按递归算法实现。这有的虽则本身没有明显的递归结构，但用递归求解比迭代求解更简单，如Hanoi问题。

数据的结构形式是按递归定义的。

如二叉树、广义表等，由于结构本身固有的递归特性，则它们的操作可递归地描述

6 递归的缺点

递归算法解题相对常用的算法如普通循环等，运行效率较低

在递归调用的过程当中系统为每一层的返回点、局部量等开辟了栈来存储。递归次数过多容易造成栈溢出等。

关于资源池

资源池不仅仅是在java当中存在的概念，在软件世界里也是广泛应用的一个东西。

我们非常常见的资源池有：

数据库连接池

web容器中的线程池

web容器中的request,response对象池

使用资源池的原因和好处大致是相似的。无外乎创建关闭维护每一个资源是比较耗费成本的。因此我们可以维护一定数量的资源，来寻求一个效率上的平衡。

资源池引入的目的:提高性能

资源池运作机制：由资源池管理器提供一定数目的目标资源，当有请求该资源时，资源池分配给一个，然后给该资源标识为忙， 标 示为忙的资源不能再被分配使用，

资源池常有的参数 初始资源的数目：资源池启动时，一次建立的资源数目，资源池最少要保证在这个数目上. 最大资源的数目：当请求的资源超出这个数目，就等待。

什么是SOA

SOA 即 service-oriented architecture，翻译成中文就是面向服务的架构。是一种服务器架构理念。

是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。

SOA具有以下五个特征：

1、可重用

一个服务创建后能用于多个应用和业务流程。

2、松耦合

服务请求者到服务提供者的绑定与服务之间应该是松耦合的。因此，服务请求者不需要知道服务提供者实现的技术细节，例如程序语言、底层平台等等。

3、明确定义的接口

服务交互必须是明确定义的。Web服务描述语言（Web Services Description Language，WSDL）是用于描述服务请求者所要求的绑定到服务提供者的细节。WSDL不包括服务实现的任何技术细节。服务请求者不知道也不关心服务究竟是由哪种程序设计语言编写的。

4、无状态的服务设计

服务应该是独立的、自包含的请求，在实现时它不需要获取从一个请求到另一个请求的信息或状态。服务不应该依赖于其他服务的上下文和状态。当产生依赖时，它们可以定义成通用业务流程、函数和 数据模型。

5、基于开放标准

当前SOA的实现形式是Web服务，基于的是公开的W3C及其他公认标准．采用第一代Web服务定义的SOAP、WSDL和UDDI以及第二代Web服务定义的WS-\*来实现SOA。

————————————————

版权声明：本文为CSDN博主「topEngineerray」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/topdeveloperr/article/details/81414318