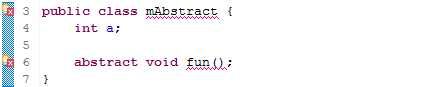
JAVA面试讲义

2017年4月

1. 抽象类与抽象方法

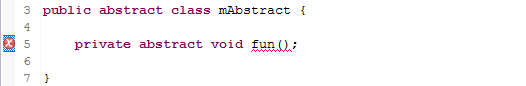
定义：如果一个类含有抽象方法，则称这个类为抽象类，抽象类必须在类前用abstract关键字修饰。

1.1 抽象方法只能在抽象类中定义



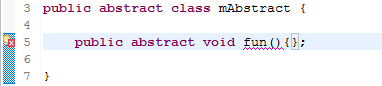
*//The abstract method fun in type mAbstract can only be defined by an abstract class*

* 1. 因为抽象类中含有无具体实现的方法，所以不能用抽象类创建对象。（注：抽象方法虽然只能在抽象类中定义，但是抽象类中可以不包含抽象方法。此外，抽象类中可以包含成员变量及普通的成员方法）
  2. 抽象方法必须为public或者protected（因为如果为private，则不能被子类继承，子类便无法实现该方法），缺省情况下默认为public。



*//The abstract method fun in type mAbstract can only set a visibility modifier, one of public or protected*

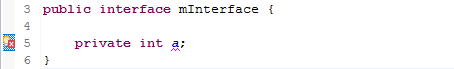
* 1. 如果一个类继承于一个抽象类，则子类必须实现父类的抽象方法。如果子类没有实现父类的抽象方法，则必须将子类也定义为为abstract类。
  2. 抽象类不用写函数体（即不能有大括号，接口同样如此）



*// Abstract methods do not specify a body*

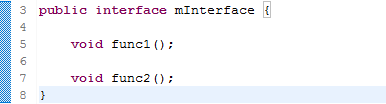
1. 接口

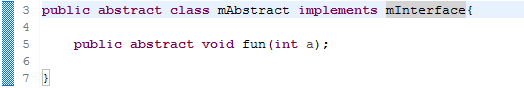
2.1 接口中可以含有变量和方法。但是要注意，接口中的变量会被隐式地指定为public static final变量（并且只能是public static final变量，用private修饰会报编译错误）。而方法会被隐式地指定为public abstract方法且只能是public abstract方法（用其他关键字，比如private、protected、static、 final等修饰会报编译错误）

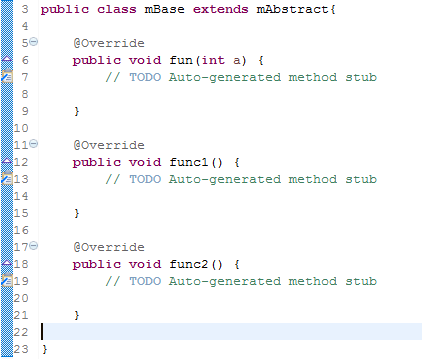


*// Illegal modifier for the interface field mInterface.a; only public, static & final are permitted*

* 1. 接口中所有的方法不能有具体的实现，也就是说，接口中的方法必须都是抽象方法。
  2. 允许一个类遵循多个特定的接口。如果一个非抽象类遵循了某个接口，就必须实现该接口中的所有方法。对于遵循某个接口的抽象类，可以不实现该接口中的抽象方法。





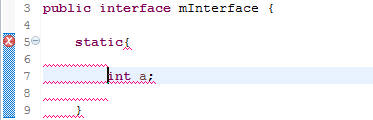


1. 接口与抽象类的区别

3.1 抽象类可以提供成员方法的实现细节，而接口中只能存在public abstract 方法；

3.2 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是public static final类型的；

3.3 接口中不能含有静态代码块以及静态方法，而抽象类可以有静态代码块和静态方法；



*// The interface mInterface cannot define an initializer*

3.4 一个类只能继承一个抽象类，而一个类却可以实现多个接口。

3.5 抽象类使用关键字implements，接口使用关键字extends。

1. ArrayList与Linked区别

4.1 ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。

4.2 对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。

4.3 .对于新增和删除操作add和remove，LinedList比较占优势，因为ArrayList要移动数据。

1. RAM与ROM区别

5.1 RAM-RamdomAccessMemory易挥发性随机存取存储器，高速存取，读写时间相等，且与地址无关，如计算机内存等。

5.2 ROM-Read Only Memory只读存储器。断电后信息不丢失，如计算机启动用的BIOS芯片。存取速度很低，（较RAM而言）且不能改写。由于不能改写信息，不能升级，现已很少使用。

ROM和RAM是计算机内存储器的两种型号，ROM表示的是只读存储器，即：它只能读出信息，不能写入信息，计算机关闭电源后其内的信息仍旧保存，一般用它存储固定的系统软件和字库等。RAM表示的是读写存储器，可其中的任一存储单元进行读或写操作，计算机关闭电源后其内的信息将不在保存，再次开机需要重新装入，通常用来存放操作系统，各种正在运行的软件、输入和输出数据、中间结果及与外存交换信息等，我们常说的内存主要是指RAM。

1. 栈与堆区别

*（*[*http://blog.sina.com.cn/s/blog\_65ca444f01011q14.html*](http://blog.sina.com.cn/s/blog_65ca444f01011q14.html)*）*

6.1 堆存储的全部是对象，每个对象都包含一个与之对应的class的信息。

6.2 栈中只保存基础数据类型的对象和自定义对象的引用(不是对象)，对象都存放在堆区中

在函数中定义的一些基本类型的变量和对象的引用变量都是在函数的栈内存中分配。当在一段代码块中定义一个变量时，java就在栈中为这个变量分配内存空间，当超过变量的作用域后，java会自动释放掉为该变量分配的内存空间，该内存空间可以立刻被另作他用。

堆内存用于存放由new创建的对象和数组。在堆中分配的内存，由java虚拟机自动垃圾回收器来管理。在堆中产生了一个数组或者对象后，还可以在栈中定义一个特殊的变量，这个变量的取值等于数组或者对象在堆内存中的首地址，在栈中的这个特殊的变量就变成了数组或者对象的引用变量，以后就可以在程序中使用栈内存中的引用变量来访问堆中的数组或者对象，引用变量相当于为数组或者对象起的一个别名，或者代号。

引用变量是普通变量，定义时在栈中分配内存，引用变量在程序运行到作用域外释放。而数组＆对象本身在堆中分配，即使程序运行到使用new产生数组和对象的语句所在地代码块之外，数组和对象本身占用的堆内存也不会被释放，数组和对象在没有引用变量指向它的时候（比如先前的引用变量x=null时），才变成垃圾，不能再被使用，但是仍然占着内存，在随后的一个不确定的时间被垃圾回收器释放掉。这个也是java比较占内存的主要原因。

以上段落来自于某一本Java程序设计的书中，实际上，栈中的变量指向堆内存中的变量，这就是Java中的指针。

总结起来就是对象存储在堆内存，引用变量存储在栈内存。栈内存指向堆内存。

1. 重写与重载的区别

7.1 重写：

-- 父类与子类之间的多态性，对父类的函数进行重新定义。如果在子类中定义某

方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。在Java中，子类可继承父类中的方法，而不需要重新编写相同的方法。但有时子类并不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。方法重写又称方法覆盖。

-- 若子类中的方法与父类中的某一方法具有相同的方法名、返回类型和参数表，

则新方法将覆盖原有的方法。如需父类中原有的方法，可使用super关键字，该关键 字引用了当前类的父类。

-- 子类函数的访问修饰权限不能少于父类的。

7.2 重载：

-- 方法重载是让类以统一的方式处理不同类型数据的一种手段。多个同名函数同时存在，具有不同的参数个数/类型。重载Overloading是一个类中多态性的一种表现。

-- Java的方法重载，就是在类中可以创建多个方法，它们具有相同的名字，但具

有不同的参数和不同的定义。调用方法时通过传递给它们的不同参数个数和参数类型

来决定具体使用哪个方法, 这就是多态性。

-- 重载的时候，方法名要一样，但是参数类型和个数不一样，返回值类型可以相同也可以不相同。无法以返回型别作为重载函数的区分标准。

1. .java文件

一个java文件中只能存在一个类是用public修饰的（除内部类以外，内部类可以是public），并且这个类必须与类名一致，文件中其他的类不能是public权限的，但可以有很多个类（protected，private的类）。



1. 静态变量（static）

static修饰的变量为静态变量。 静态变量是线程共享，无论谁调用，都指向同一个变量。存储在方法区。 方法区是各个线程共享的内存区域，存放已经被虚拟机加载的类、常量、静态变量、即时编译器编译后的代码。

1. Java反射机制

普通的java对象是通过new关键字把对应类的字节码文件加载到内存，然后创建该对象的。

反射是通过一个名为Class的特殊类，用Class.forName("className");得到类的字节码对象，然后用newInstance()方法在虚拟机内部构造这个对象（针对无参构造函数）。

也就是说反射机制让我们可以先拿到java类对应的字节码对象，然后动态的进行任何可能的操作，包括：

在运行时判断任意一个对象所属的类；

在运行时构造任意一个类的对象；

在运行时判断任意一个类所具有的成员变量和方法；

在运行时调用任意一个对象的方法；

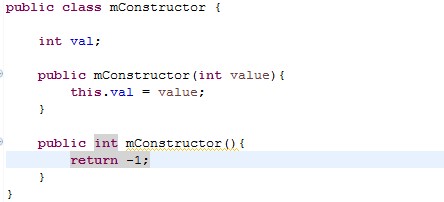
生成动态代理。

这些都是反射的功能。使用反射的主要作用是方便程序的扩展。

1. 构造方法

11.1 Java构造方法可以有很多个并且名字与类名相同，且不需要返回值。

11.2成员方法可以与类名相同



1. JVM内存配置参数

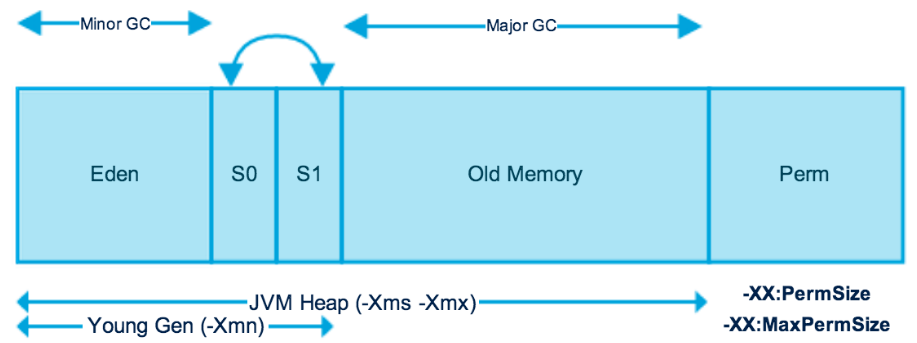
-Xmx10240m -Xms10240m -Xmn5120m -XXSurvivorRatio=3

-Xmx ：堆的最大值

-Xms ：堆的最小值

-Xmn ：堆年轻代大小

-XXSurvivorRatio：Eden区和Survior区的占用比例.

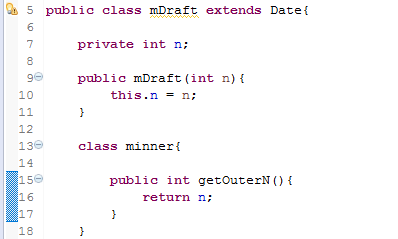


Survior区有两个，就是图中的S0和S1，而Eden区只用一个， -XXSurvivorRatio参数是Eden区和单个Survior区的比例。

1. 内部类

13.1 内部类可以直接访问外部类的私有变量。

13.2 外部类也可以直接访问内部类的私有变量。



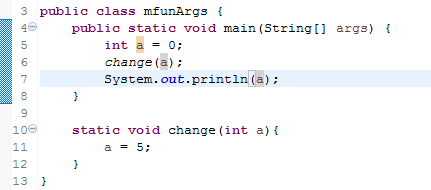
1. 依赖注入和控制反转

（http://baitai.iteye.com/blog/792980）

依赖注入和控制反转是对同一件事情的不同描述，从某个方面讲，就是它们描述的角度不同。依赖注入是从应用程序的角度在描述，可以把依赖注入描述完整点：应用程序依赖容器创建并注入它所需要的外部资源；而控制反转是从容器的角度在描述，描述完整点：容器控制应用程序，由容器反向的向应用程序注入应用程序所需要的外部资源。

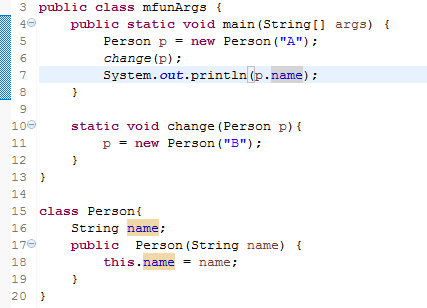
1. Java函数的传值与传引用

15.1 基本数据类型：传值，方法不会改变实参的值。

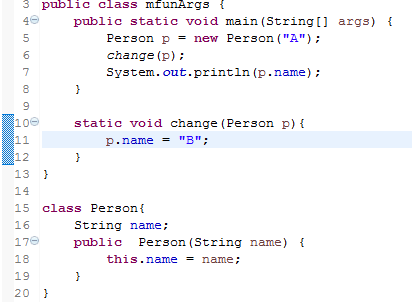


//out: 0

15.2 对象类型参数：传引用，方法体内改变形参引用，不会改变实参的引用，但有可能改变实参对象的属性值。



//out: A



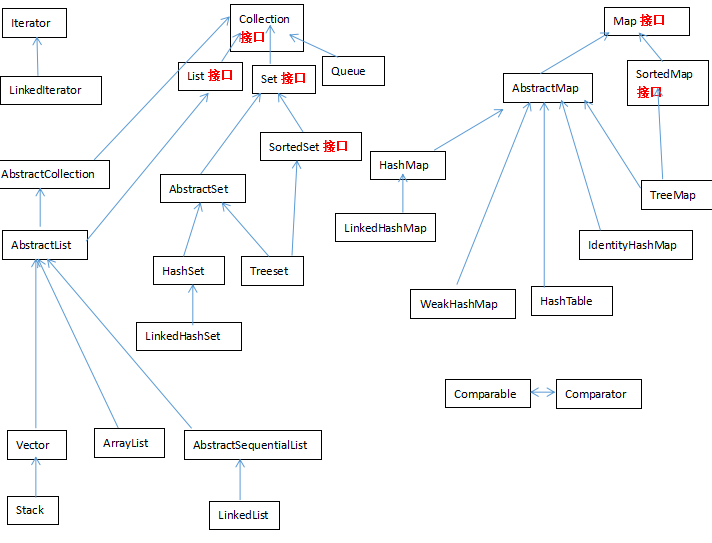
//out: B

1. Java反射机制getMethods

16.1 public Method[] getMethods()：返回某个类的所有公用（public）方法包括其继承类的公用方法，当然也包括它所实现接口的方法。

16.2 public Method[] getDeclaredMethods()：对象表示的类或接口声明的所有方法， 包括公共、保护、默认（包）访问和私有方法，但不包括继承的方法。当然也包括它所实现接口的方法。

1. Iterator Collection Map集成关系图



1. 构造函数的执行顺序

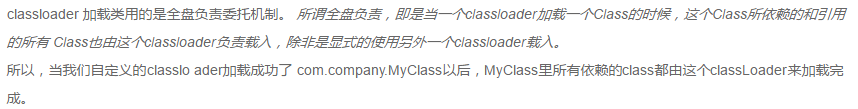
父类静态代码块—>子类静态代码块—>父类构造代码块—>父类构造函数—>子类构造代码块—>子类构造函数。

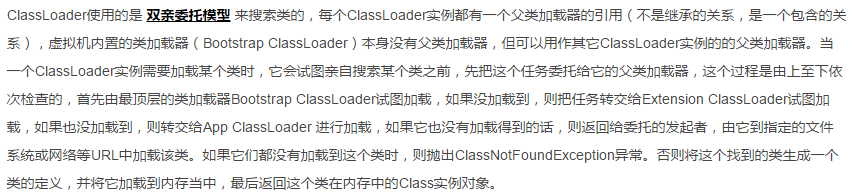
1. java classloader

19.1 Bootstrap ClassLoader：负责加载java基础类，主要是 %JRE\_HOME/lib/ 目录下的rt.jar、resources.jar、charsets.jar和class等；

19.2 Extension ClassLoader：负责加载java扩展类，主要是 %JRE\_HOME/lib/ext 目录下的jar和class

19.3 App ClassLoader：负责加载当前java应用的classpath中的所有类。





1. Servlet生命周期

Servlet的生命周期分为5个阶段：加载、创建、初始化、处理客户请求、卸载。

(1)加载：容器通过类加载器使用servlet类对应的文件加载servlet

(2)创建：通过调用servlet构造函数创建一个servlet对象

(3)初始化：调用init方法初始化

(4)处理客户请求：每当有一个客户请求，容器会创建一个线程来处理客户请求

(5)卸载：调用destroy方法让servlet自己释放其占用的资源

1. 泛型

泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。没有泛型的情况的下，通过对类型Object的引用来实现参数的“任意化”，“任意化”带来的缺点是要做显式的强制类型转换，而这种转换是要求开发者对实际参数类型可以预知的情况下进行的。对于强制类型转换错误的情况，编译器可能不提示错误，在运行的时候才出现异常，这是一个安全隐患。

泛型的好处是在编译的时候检查类型安全，并且所有的强制转换都是自动和隐式的，以提高代码的重用率。