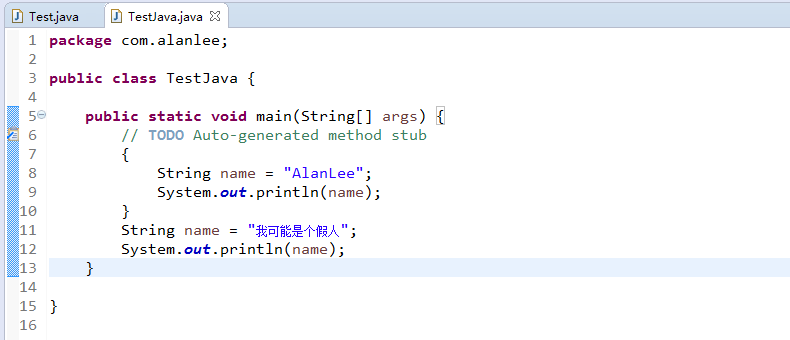
Java的作用域修饰词public、protected、private、[default]解析

Java代码以其总是带着这样那样的修饰词而自(chou)成(ming)风(yuan)格(bo)o(╯□╰)o，开玩笑的，其实每一个程序员都应该热爱自己所使用的语言，为什么呢？你爱不爱它你每天还是要和它接触，长时间和自己不喜欢的东西接触是非常痛苦的事情，就算是编码也要快乐呀~ 算了 开始正题

在介绍作用域修饰词之前，我们先要了解Java的作用域基本规则：

* 以 { } 花括号对为一个**代码空间**。
* 代码空间内定义的变量，在这个空间结束后被回收，空间外无法再对其进行访问。

一个栗子：下面图中的代码都是合法的。



* 定义和声明是一体的（后续使用“定义“这种说法）（主要针对C、C++类语言来说）。

这一点只有同时学过Java和C的人才会有体会，那就是C类语言中的定义和声明可以分开简直是太奇怪了！！

在C类语言中，我们定义一个函数之后不一定就能使用这个函数，因为会出现作用域的问题，函数只能调用定义在自身“前面”[*这里的前面和后文的后面是指代码的前后位置*]的函数和变量，而如果现在你要使用“后面”定义的函数和变量，那么你就需要在使用之前进行**声明**，告诉编译器：后面的代码中有这个函数或者变量。

但是实际上，这种方式是反人类反人性的，因为我们写代码当然不会完全想好编写顺序再开始动手写呀！可能觉得某一个函数简单，先用几分钟把它敲出来，有时候突然想到了某一函数的实现方案，就把这些函数先写出来，此外，正常的编码顺序应该是先在main函数中搭好框架，然后在后续补充实现代码，这同样符合代码阅读习惯，即：先从main函数看出程序大致流程，再去看自己感兴趣的实现部分，打开一个项目，反过来：首先看到一大堆不知道在干什么的函数当然会让人脑阔疼。

另外，通过编译器的一些改进，是可以实现代码的非顺序安放的，这些繁琐的事本就应该由编译器来完成，而Java编译器是做到了这一点的。

* 一个代码空间内定义的函数、变量是可以在代码位置之后的任意位置进行访问的，

又是一个栗子：

String name = "a";  
{  
 String name = "b";  
 System.*out*.print(name);  
}

这里红色显示部分在编译时报错，因为这里发生了重复定义。

* **只有**一个**类的直接成员变量/常量/函数**可以用作用域修饰符来修饰，直接的意思是不能是其子空间.

这意味着你不能在一个方法中用public、private等修饰词修饰。

举例：这里的public会报错，因为只有类中可以定义public变量。

public void test(){  
 public String name = "a";  
}

又一个栗子：这里面的public会报错，因为其不是类的直接成员。

public class MainActivity extends AppCompatActivity {  
 private TextView mTextMessage;

{  
 public TextView t = mTextMessage;  
 }

}

* 函数只能定义在类中，函数中不能再定义函数。
* int x(){
* int y(){
* return 0;
* }  
   return y();  
  }

这里会报错。

* 函数在类中只要有定义，那么在类中的任意位置均可访问到。这一点的意义在于你可以访问在你之后定义的函数。

其中作用域一般被放在一条语句最开头的位置，例如：

@Override  
public void requestChildFocus(View child, View focused) {  
 if (*DBG*) {  
 System.*out*.println(this + " requestChildFocus()");  
 }

这种，最开头的public就是作用域修饰词.

首先我们上一张表格简单表明其中的大致关系。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | class内部 | class外部 |  |  |
| **public** |  |  |  |  |
| **[default]** |  |  |  |  |
| **protected** |  |  |  |  |
| **private** |  |  |  |  |
|  |  |  |  |  |

下面分别对不同的修饰词进行解析。