Kotlin中的inline、noinline、crossinline

原创 wby93 于 2020-09-26 22:17:29 发布 82 收藏 版权

分类专栏： kotlin

inline

inline表示声明的函数是内联的，调用 内联函数 时，内联函数的函数体会被复制到调用它的地方。

```
fun main() {
    method1 {
        println("main")
    }
}

inline fun method1(block: ()->Unit){
    block()
}
```

以上代码的main函数调用了内联函数，经过 反编译 后的代码如下：

```
public final class HelloKotlinKt {
   public static final void main() {
      int $i$f$method1 = false;
      int var1 = false;
      //直接将内联函数的函数体复制到此处
      String var2 = "main";
      boolean var3 = false;
      System.out.println(var2);
   }

   // $FF: synthetic method
   public static void main(String[] var0) {
      main();
   }

   public static final void method1(@NotNull Function0 block) {
      int $i$f$method1 = 0;
      Intrinsics.checkParameterIsNotNull(block, "block");
      block.invoke();
   }
}
```

如果将method1（）方法的inline去掉，再次反编译得到：

```
public final class HelloKotlinKt {
   public static final void main() {
   //JVM不支持高阶函数，所以把函数作为参数传递必须通过对象实现。
      method1((Function0)null.INSTANCE);
   }

   // $FF: synthetic method
   public static void main(String[] var0) {
      main();
   }

   public static final void method1(@NotNull Function0 block) {
      Intrinsics.checkParameterIsNotNull(block, "block");
```

```
        block.invoke();
    }
}
```

Kotlin 中的高阶函数在JVM中必须通过创建对象实现，如果在for循环中多次进行函数参数的传递，就会创建很多个对象，占用内存空间，而inline就可以解决这样的问题。

inline会导致函数体内的代码在每次调用的时候被复制一份，不必要的使用inline导致代码体积增大，因此应该在函数中有函数类型的参数时使用inline。

noinline

noinline只能作用于内联函数中函数类型的参数，表示该函数类型的参数不参与内联。

如果内联函数的返回值是函数类型时，如果函数类型的参数不使用noinline，那么该函数类型的参数不能作为返回值使用。这是函数类型的返回值本质上是一个对象，如果函数类型的参数参与inline，那么函数类型的参数就展开为函数体，不再是一个对象，因此必须加上noinline才能作为返回值使用。

```
fun main() {
    method2({
        println("block1")
    },{
        println("block2")
    }).invoke()
}

inline fun method2(noinline block1: ()->Unit,block2: ()->Unit):()->Unit{
    return block1
    //return block2 报错
}
```

crossinline

crossinline同样作用于内联函数中函数类型的参数。

在以下代码中，当在Lambda表达式中使用return时，代码无法判断return结束的是method3函数还是main函数，如果method3函数是内联函数，那么return显然就作用于main函数，因此kotlin规定， Lambda表达式中不能使用return，除非这个函数是内联函数的参数，所以Lambda表达式中的return实际上结束的就是此处的main函数。

```
fun main() {
    method3{
        println()
        return //报错,method3加上inline则不报错
    }
}

fun method3( block: ()->Unit){

}
```

当需要在内联函数中对函数类型的参数间接调用时，如下所示，return与main函数之间被thread切断，kotlin规定内联函数里的函数类型的参数不允许间接调用。

```
fun main() {
    method3{
        println()
        return//return与main的关系被切断
    }
}

inline fun method3( block: ()->Unit){
    thread {
```

```
        block()//报错
    }
}
```

如果一定需要间接调用函数类型的参数，那么在该参数前使用crossinline，但使用crossinline之后，该函数类型的参数将不能使用return。

```
fun main() {
    method3{
        println()
        return //报错，使用crossinline的函数类型的参数
               //中不能使用return
   }
}

inline fun method3(crossinline block: ()->Unit){
    thread {
        block()
    }
}
```