当你的执行逻辑只有一行代码时，{ }可以省略。这样再来看上述代码就很好理解了吧？

除了精确匹配之外，when 语句还允许进行类型匹配。什么是类型匹配呢？这里我再举个例子。定义一个 checkNumber()函数，如下所示：

```
fun checkNumber(num: Number) {
    when (num) {
        is Int -> println("number is Int")
        is Double -> println("number is Double")
        else -> println("number not support")
    }
}
```

上述代码中，is 关键字就是类型匹配的核心，它相当于 Java 中的 instanceof 关键字。由于 checkNumber()函数接收一个 Number 类型的参数，这是 Kotlin 内置的一个抽象类，像 Int、Long、Float、Double 等与数字相关的类都是它的子类，所以这里就可以使用类型匹配来判断传入的参数到底属于什么类型，如果是 Int 型或 Double 型，就将该类型打印出来，否则就打印不支持该参数的类型。

现在我们可以尝试在 main()函数中调用 checkNumber()函数，如下所示：

```
fun main() {
    val num = 10
    checkNumber(num)
}
```

这里向 checkNumber()函数传入了一个 Int 型参数。运行一下程序，结果如图 2.12 所示。

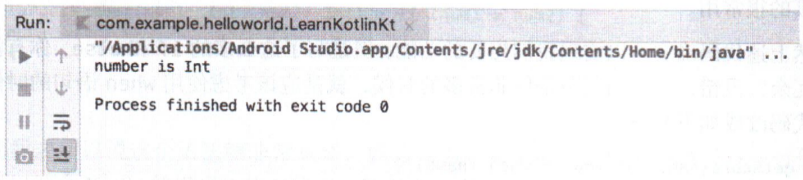

图 2.12　checkNumber()函数传入 Int 型参数

可以看到，这里成功判断出了参数是 Int 类型。

而如果我们将参数改为 Long 型：

```
fun main() {
    val num = 10L
    checkNumber(num)
}
```

重新运行一下程序，结果如图 2.13 所示。