

public 是 scala中默认的访问级别。所以在scala中当一个变量的修饰符是public 时，这个public 可以不写。

Fields are also known as *instance variables*

in the absence of any explicit return statement, a Scala method returns the last value computed by the method.

scala方法简写：

class ChecksumAccumulator {

private var sum = 0

def add(b: Byte): Unit = {

sum += b

}

def checksum(): Int = { return ~(sum & 0xFF) + 1

} }

可以简写成如下形式：

class ChecksumAccumulator {  
private var sum = 0  
def add(b: Byte) = sum += b  
def checksum() = ~(sum & 0xFF) + 1

}

最完美的写法是：

class ChecksumAccumulator {  
private var sum = 0  
def add(b: Byte): Unit = { sum += b }

def checksum(): Int = ~(sum & 0xFF) + 1

}

总结： scala中的方法简写包括：

1. 方法的return 值可以不写
2. 方法参数都是val 来修饰的
3. 方法的返回值是一个简单的语句的话，可以把方法体中的大括号省略，如果这个语句不是太长的话，可以直接跟def写在一行如：

def add(b: Byte): Unit = { sum += b }

A method that is executed only for its side effects is known as a *procedure*.

推断分号的规则：

The rules of semicolon inference

4.3

The precise rules for statement separation are surprisingly simple for how well they work. In short, a line ending is treated as a semicolon unless one of the following conditions is true:

1. The line in question ends in a word that would not be legal as the end of a statement, such as a period or an infix operator.

2. The next line begins with a word that cannot start a statement.

3. The line ends while inside parentheses (...) or brackets [...], because these cannot contain multiple statements anyway

Singleton Objects

When a singleton object shares the same name with a class, it is called that class’s *companion object*. You must define both the class and its companion object in the same source file. The class is called the *companion class* of the singleton object. A class and its companion object can access each other’s private members.

Singleton objects 与 classes 的区别

One difference between classes and singleton objects is that singleton objects cannot take parameters, whereas classes can.

A singleton object that does not share the same name with a companion class is called a *standalone object*

4The name of the synthetic class is the object name plus a dollar sign. Thus the synthetic class for the singleton object named ChecksumAccumulator is ChecksumAccumulator$.

Each singleton object is implemented as an instance of a *synthetic class* referenced from a static variable, so they have the same initialization semantics as Java statics.4

*scala的默认导入：*

Scala implicitly imports members of packages java.lang and scala, as well as the members of a singleton object named Predef, into every Scala source file. Predef, which resides in package scala, contains many useful methods. For example, when you say println in a Scala source file, you’re actually invoking println on Predef. (Predef.println turns around and invokes Console.println, which does the real work.) When you say assert, you’re invoking Predef.assert.

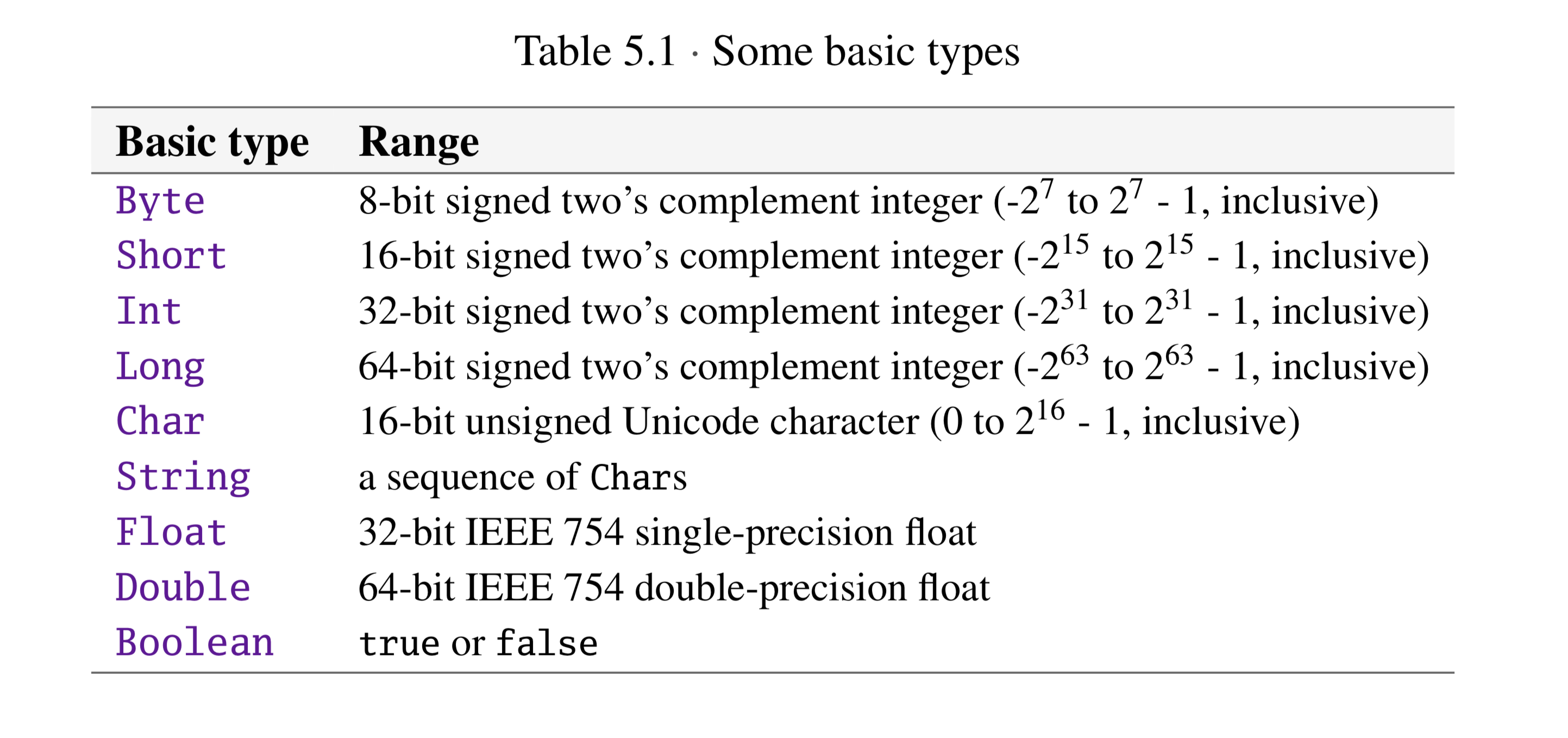
Chapter5 Basic types and Operations

scala的基本类型包括：Byte Short Int Long

Double Float Char Boolean ， String.

整形类型和数值类型：

types Byte, Short, Int, Long, and Char are called *integral types*. The integral types plus Float and Double are called *numeric types*.



注意scala不支持8进制

Scala does not support octal literals; integer literals that start with a 0, such as 031, do not compile

Symbol literals:

Symbol literals are typically used in situations where you would use just an identifier in a dynamically typed language.

Operators

Any method can be an operator

In Scala operators are not special language syntax; any method can  
be an operator. What makes a method an operator is how you *use* it. When you write “s.indexOf('o')”, indexOf is not an operator. But when you write “s indexOf 'o'”, indexOf *is* an operator, because you’re using it in operator notation.

In contrast to the infix operator notation—in which operators take two operands, one to the left and the other to the right—prefix and postfix oper- ators are *unary*:

Postfix operators are methods that take no arguments, when they are in- voked without a dot or parentheses. In Scala, you can leave off empty paren- theses on method calls.

scala中比较两个对象是否相等用==

不相等用!=

How Scala’s == differs from Java’s

In Java, you can use == to compare both primitive and reference types. On primitive types, Java’s == compares value equality, as in Scala. On reference types, however, Java’s == compares *reference equality*, which means the two variables point to the same object on the JVM’s heap. Scala provides a facility for comparing reference equality, as well, under the name eq. However, eq and its opposite, ne, only apply to objects that directly map to Java objects. The full details about eq and ne are given in Sections 11.1 and 11.2. Also, see Chapter 30 on how to write a good equals method.

