Ppt1

这节课我们来讲一下数据类型。在Rust语言里，每一个值都有一个特定的数据类型。今天我们讨论栈上的数据类型。它分为标量类型和复合类型。当不说明类型的时候编译器会自动推到类型，但是有的时候可推到类型选择比较多，编译器无法推导出来时，我们必须表明数据类型。

Ppt2

标量

标量（scalar） 类型代表一个单独的值。Rust 有四种基本的标量类型：整型、浮点型、布尔类型和字符类型。你可能在其他语言中见过它们。让我们深入了解它们在 Rust 中是如何工作的。

Ppt3

整数

整数是一个没有小数部分的数字。比如 u32 整数类型，该类型声明表明，它关联的值应该是一个占据 32 比特位的无符号整数（有符号整数类型以 i 开头而不是 u）。表格展示了 Rust 内建的整数类型。在有符号列和无符号列中的每一个变体（例如，i16）都可以用来声明整数值的类型。

每一个变体都可以是有符号或无符号的，并有一个明确的大小。有符号 和 无符号 代表数字能否为负值，换句话说，这个数字是否有可能是负数（有符号数），或者永远为正而不需要符号（无符号数）。这有点像在纸上书写数字：当需要考虑符号的时候，数字以加号或减号作为前缀；然而，可以安全地假设为正数时，加号前缀通常省略。每一个有符号的变体可以储存包含从 -(2n - 1) 到 2n - 1 - 1 在内的数字，这里 n 是变体使用的位数。所以 i8 可以储存从 -(27) 到 27 - 1 在内的数字，也就是从 -128 到 127。无符号的变体可以储存从 0 到 2n - 1 的数字，所以 u8 可以储存从 0 到 28 - 1 的数字，也就是从 0 到 255。另外，isize 和 usize 类型依赖运行程序的计算机架构：64 位架构上它们是 64 位的， 32 位架构上它们是 32 位的。

Ppt4

浮点数

Rust 也有两个原生的 浮点数（floating-point numbers）类型，它们是带小数点的数字。Rust 的浮点数类型是 f32 和 f64，分别占 32 位和 64 位。默认类型是 f64，因为在现代 CPU 中，它与 f32 速度几乎一样，不过精度更高。

Ppt5

数值运算

Rust 中的所有数字类型都支持基本数学运算：加法、减法、乘法、除法和取余。整数除法会向下舍入到最接近的整数。

Ppt6

Ppt7

布尔类型

正如其他大部分编程语言一样，Rust 中的布尔类型有两个可能的值：true 和 false。Rust 中的布尔类型使用 bool 表示。

Ppt8

字符类型

我们目前为止只使用到了数字，不过 Rust 也支持字母。Rust 的 char 类型是语言中原生的字母类型。

Ppt9

复合类型

复合类型（Compound types）可以将多个值组合成一个类型。Rust 有两个原生的复合类型：元组（tuple）和数组（array）。

Ppt10

Tuple

元组是一个将多个其他类型的值组合进一个复合类型的主要方式。元组长度固定：一旦声明，其长度不会增大或缩小。我们使用包含在圆括号中的逗号分隔的值列表来创建一个元组。元组中的每一个位置都有一个类型，而且这些不同值的类型也不必是相同的。

Ppt11

我们可以通过destructure来得到Tuple中的元素。Tup变量绑定到整个元组上，因为元组是一个单独的复合元素。为了从元组中获取单个值，可以使用模式匹配（pattern matching）来解构（destructure）元组值。程序首先创建了一个元组并绑定到 tup 变量上。接着使用了 let 和一个模式将 tup 分成了三个不同的变量，x、y 和 z。这叫做 解构（destructuring），因为它将一个元组拆成了三个部分。最后，程序打印出了 y 的值，也就是 6.4。

Ppt12

除了使用模式匹配解构外，也可以使用点号（.）后跟值的索引来直接访问它们。这个程序创建了一个元组，x，并接着使用索引为每个元素创建新变量。跟大多数编程语言一样，元组的第一个索引值是 0。没有任何值的元组 () 是一种特殊的类型，只有一个值，也写成 () 。该类型被称为 单元类型（unit type），而该值被称为 单元值（unit value）。如果表达式不返回任何其他值，则会隐式返回单元值。

Ppt13

数组

另一个包含多个值的方式是 数组（array）。与元组不同，数组中的每个元素的类型必须相同。Rust 中的数组与一些其他语言中的数组不同，因为 Rust 中的数组是固定长度的：一旦声明，它们的长度不能增长或缩小。Rust 中，数组中的值位于中括号内的逗号分隔的列表中。

Ppt14

数组的声明

我们可以像这样编写数组的类型：在方括号中包含每个元素的类型，后跟分号，再后跟数组元素的数量。这里，i32 是每个元素的类型。分号之后，数字 5 表明该数组包含五个元素。以这种方式编写数组的类型看起来类似于初始化数组的另一种语法：如果要为每个元素创建包含相同值的数组，可以指定初始值，后跟分号，然后在方括号中指定数组的长度。变量名为 a 的数组将包含 5 个元素，这些元素的值最初都将被设置为 3。这种写法与 let a = [3, 3, 3, 3, 3]; 效果相同，但更简洁。

Ppt15

获取数组的元素

我们可以使用索引来访问数组的元素。在这个例子中，叫做 first 的变量的值是 1，因为它是数组索引 [0] 的值。变量 second 将会是数组索引 [1] 的值 2。

Ppt1

In this lesson we will talk about data types. In the Rust language, every value has a specific data type. Today we discuss the data types on the stack. It is divided into scalar type and composite type. When the type is not specified, the compiler will automatically deduce the type, but sometimes there are more choices for the derivable type, and when the compiler cannot deduce it, we must indicate the data type.

Ppt2

The scalar type represents a single value. Rust has four basic scalar types: integer, float, boolean, and char types. Let's dive into how they work in Rust.

Ppt3

An integer is a number without a decimal part. For example, the u32 integer type. The type declaration indicates that its associated value should be an unsigned integer occupying 32 bits (a signed integer type starts with i instead of u). The table shows the built-in integer types in Rust. Each of the variants in signed and unsigned columns (for example, i16) can be used to declare the type of integer values.

Each variant can be signed or unsigned, and has a definite size. Signed and unsigned represent whether the number can be negative, in other words, is it possible that the number is negative (signed number), or is always positive without a sign (unsigned number). This is a bit like writing numbers on paper: when you need to consider symbols, the numbers are prefixed with a plus or minus sign; however, when it is safe to assume a positive number, the plus sign prefix is usually omitted. Each signed variant can store numbers from -(2n-1) to 2n-1-1, where n is the number of digits used by the variant. So i8 can store numbers from -(27) to 27-1, that is, from -128 to 127. The unsigned variant can store numbers from 0 to 2n-1, so u8 can store numbers from 0 to 28-1, that is, from 0 to 255. In addition, the isize and usize types depend on the computer architecture of the running program: they are 64-bit on 64-bit architecture, and 32-bit on 32-bit architecture.

Ppt4

Rust also has two native float numbers types, which are numbers with decimal points. Rust's float number types are f32 and f64, which occupy 32 bits and 64 bits respectively. The default type is f64, because in modern CPUs, it is almost the same speed as f32, but with higher precision.

Ppt5

All numeric types in Rust support basic mathematical operations: addition, subtraction, multiplication, division, and remainder. Integer division is rounded down to the nearest integer.

Ppt6

Ppt7

Like most other programming languages, the Boolean type in Rust has two possible values: true and false. Boolean types in Rust are represented by bool.

Ppt8

We have only used numbers so far, but Rust also supports letters. Rust's char type is the native letter type in the language.

Ppt9

Compound types can combine multiple values into one type. Rust has two native composite types: tuple and array.

Ppt10

Tuples are the main way to combine multiple values of other types into a composite type. The tuple length is fixed: once declared, its length will not increase or decrease. We use a comma-separated list of values enclosed in parentheses to create a tuple. Each position in the tuple has a type, and the types of these different values need not be the same.

Ppt11

We can get the elements in Tuple through destructure. The Tup variable is bound to the entire tuple because the tuple is a single compound element. In order to obtain a single value from a tuple, pattern matching can be used to destructure the tuple value. The program first creates a tuple and binds it to the tup variable. Then use let and a pattern to separate tup into three different variables, x, y, and z. This is called destructuring because it breaks a tuple into three parts. Finally, the program prints out the value of y, which is 6.4.

Ppt12

In addition to using pattern matching destructuring, you can also use the dot (.) followed by the index of the value to directly access them. This program creates a tuple, x, and then uses the index to create a new variable for each element. Like most programming languages, the first index value of a tuple is 0. A tuple without any value () is a special type, only one value, also written as (). This type is called the unit type, and the value is called the unit value. If the expression does not return any other value, the cell value will be returned.

Ppt13

Another way to contain multiple values is an array. Unlike tuples, each element in the array must be of the same type. Arrays in Rust are different from arrays in some other languages because arrays in Rust are fixed-length: once declared, their length cannot grow or shrink. In Rust, the values in an array are in a comma-separated list within square brackets.

Ppt14

We can write the type of the array like this: Enclose the type of each element in square brackets, followed by a semicolon, and then the number of array elements. Here, i32 is the type of each element. After the semicolon, the number 5 indicates that the array contains five elements. Writing the type of an array in this way looks similar to another syntax for initializing an array: if you want to create an array containing the same value for each element, you can specify the initial value, followed by a semicolon, and then specify the value of the array in square brackets length. The array with the variable name a will contain 5 elements, and the value of these elements will all be set to 3 initially. This way of writing has the same effect as let a = [3, 3, 3, 3, 3]; but is more concise.

Ppt15

We can use the index to access the elements of the array. In this example, the value of the variable called first is 1 because it is the value of array index [0]. The variable second will be the value 2 of the array index [1].