这一章探讨的是一个rust的核心概念，所有权。关于所有权rust有一系列管理内存的规则，然后编译的时候编译器会对程序进行检查，只有不违法规则的程序才能通过编译。

这章通过string类型，4.1介绍了所有权的基本概念，4.2介绍了借用和引用，4.3介绍了字符串中的切片。

先介绍两个概念：堆栈空间、浅拷贝和深拷贝

栈（Stack）和堆（Heap）是两种主要的内存分配空间，它们在内存管理中扮演着不同的角色：

1. **栈（Stack）**：
   * 堆栈上的数据大小在编译时必须已知，因此它们通常用于存储固定大小的数据类型。
   * 当函数返回时，其局部变量会被自动弹出堆栈，内存被立即释放。
2. **堆（Heap）**：
   * 堆是一种用于动态内存分配的空间，大小在运行时可以变化。
   * 用于存储大小在编译时未知或可能变化的数据结构，如String、Vec<T>等。
   * 需要通过指针来间接访问。
   * 堆内存的分配和释放通常需要手动管理，或者由垃圾收集器（GC）处理，但Rust语言通过所有权和生命周期的概念来自动管理堆内存的释放，无需GC。
3. **浅拷贝（Shallow Copy）**：
   * 浅拷贝创建的是原始数据的引用或副本，但是它复制的只是数据的顶层结构。
   * 对于简单的数据类型（如整数、浮点数、布尔值等），浅拷贝通常就是复制其值。
   * 对于复合数据类型（如数组、列表、结构体等），浅拷贝只复制了指向堆上数据的指针或引用，而不复制数据本身。
   * 因此，浅拷贝的速度通常很快，但是如果你修改了复制后的数据，可能会影响原始数据（如果数据是可变的）。
4. **深拷贝（Deep Copy）**：
   * 深拷贝会创建原始数据的一个完全独立的副本，包括所有层级的数据。
   * 深拷贝不仅复制了数据的顶层结构，还递归地复制了所有被引用的数据，确保副本和原始数据没有任何共享的内存。
   * 对于复合数据类型，深拷贝会分配新的内存，并将原始数据的所有元素都复制到新内存中。
   * 深拷贝的成本通常比浅拷贝高，因为它需要更多的时间和内存来复制所有的数据。

关于所有权，先来看一下所有权的基本规则：

* Each value in Rust has an *owner*.
* There can only be one owner at a time.
* When the owner goes out of scope, the value will be dropped.
* 每个值都有一个所有者。
* 一次只能有一个所有者。
* 当所有者超出作用域时，值将被丢弃。

对于 String 类型，为了支持一个可变的、可增长的文本片段，我们需要在堆上分配一块编译时未知的内存来存储内容。在对字符串的使用完成后，需要将内存返回给分配器。

Rust在这一部分采取的策略是，一旦拥有它的变量超出作用域，内存就会自动返回。

变量从声明它的那一刻起，直到当前作用域的结束都是有效的。

Rust中的Copy trait可以自动实现浅拷贝，适用于那些存储在栈上且大小固定的简单类型。

所有权的转移：对于字符串等存储在堆里的内容，如果执行s2 = s1，默认执行浅拷贝，也有是让s2指向s1，导致作用域结束会释放两次。Rust为了避免这个问题，将s1的所有权转移给s2，s1无效避免了释放两次的问题。

如果执行深拷贝，需要clone函数。

Let x = 5

Let y = x

另外，在栈中存储的数据类型（整数、浮点数、字符等），它们的大小是确定的，可以直接执行拷贝，不存在所有权的转移问题。

函数和返回值

当字符串作为参数的时候，原字符串的所有值会被转移给参数，即原字符串无效。

字符串作为返回值时，函数内字符串的所有值会返回给相应的对象。

字符串还要使用就必须返回->引用（不可变引用 可变引用 悬垂引用）

引用分为mutable和immutable，即可变引用和不可变引用。 &是不可变引用，&mut是可变引用。引用作为函数参数时，函数声明和使用的部分都要加上&（或&mut）

引用有以下两条规则：

At any given time, you can have either one mutable reference or any number of immutable references.

References must always be valid.

在任何给定的时刻，你可以有一个可变引用或者任意数量的不可变引用。

引用必须始终是有效的。（rust不允许悬垂引用（引用指向被释放的内存））

a reference’s scope starts from where it is introduced and continues through the last time that reference is used.

引用的作用域不是指花括号中间，而是从它被引入的地方开始，并持续到它最后一次被使用的时候。

这章的最后一部分是关于切片：

切片是一种引用（没有所有权），允许引用集合中连续的一系列元素。同样以字符串切片为例。

例子：获取字符串的第一个单词，返回空格所在的下标

Clear后原来获取的坐标没有意义。

切片不引用整个字符串，而是引用字符串的一部分[x..y] x是第一个位置，y是最后一个+1，

省略x从头开始，省略y到字符串最后。

the compiler will ensure the references into the String remain valid.

编译器会保证对字符串的引用有效。

切片也可以作为函数参数传递。