**Author:陈龙涛**

**1.cargo.lock的生成会保证写好的rust程序在任何电脑操作系统中运行的效果都相同。**

第一次运行 cargo build 会在顶层目录生成 cargo.lock 文件.该文件负责追踪项目依赖的精确版本

不需要手动修改该文件

**2.变量与常量**

常量(constant)，常量在绑定值以后也是不可变的，但是它与不可变的变量有很多区别:

-不可以使用mut，常量永远都是不可变的

声明常量使用const关键字，它的类型必须被标注

常量可以在任何作用域内进行声明，包括全局作用域

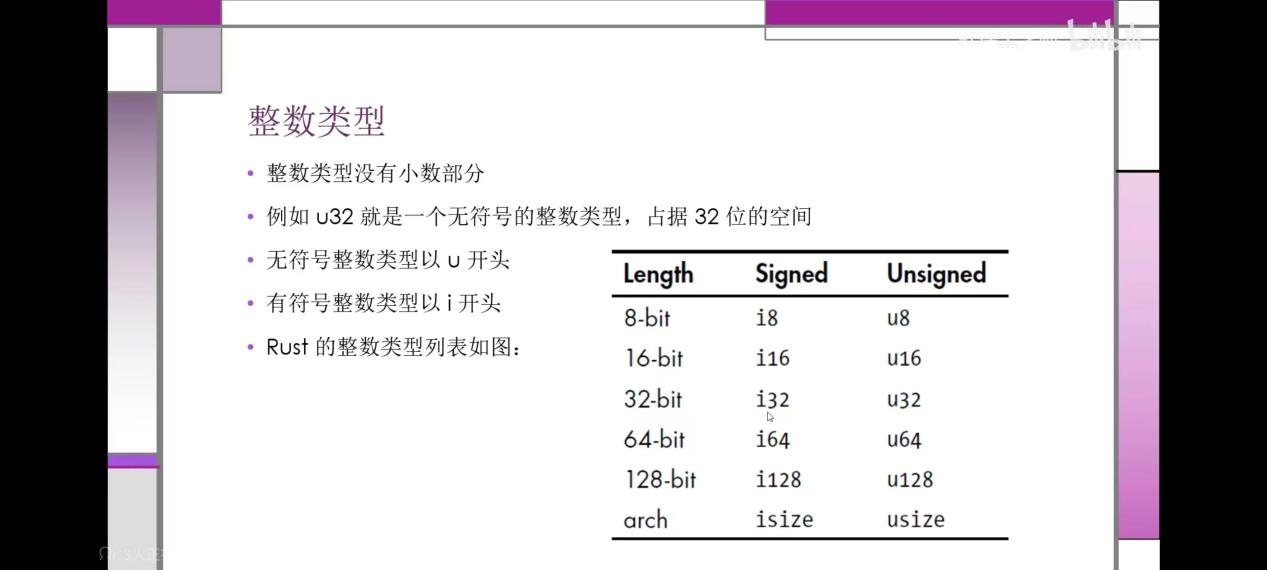
常量只可以绑定到常量表达式，无法绑定到函数的调用结果或只能在运行时才能计算出的值

在程序运行期间，常量在其声明的作用域内一直有效

命名规范:Rust里常量使用全大写字母，每个单词之间用下划线分开，例如:- MAX POINTS

例子:const MAX POINTS:U32 =100 000;

**3.**



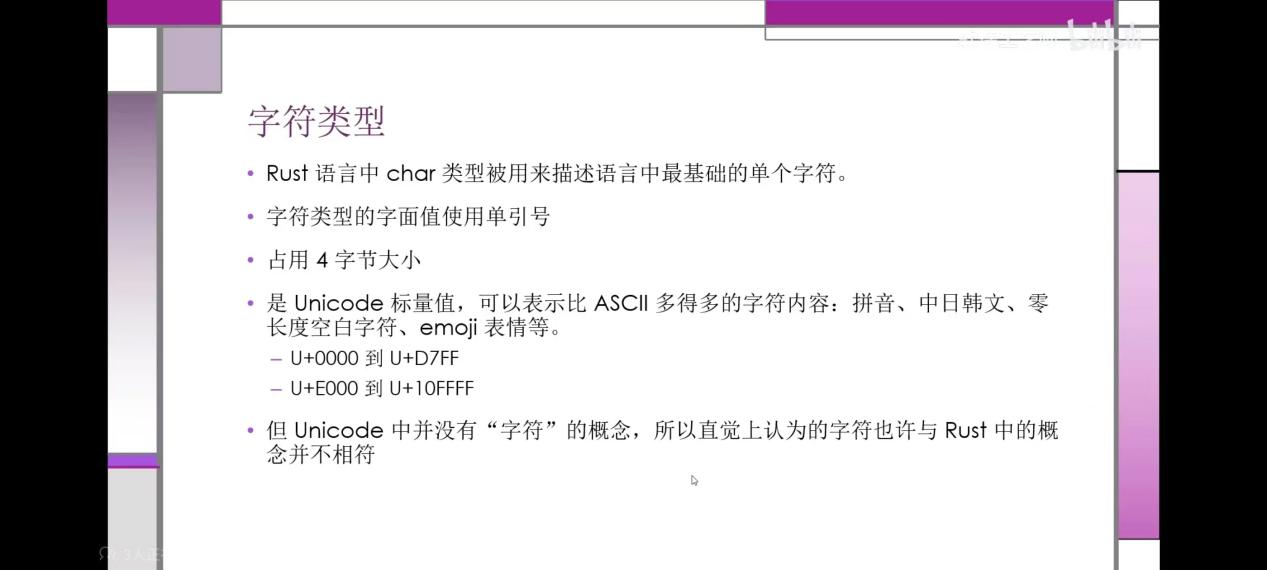
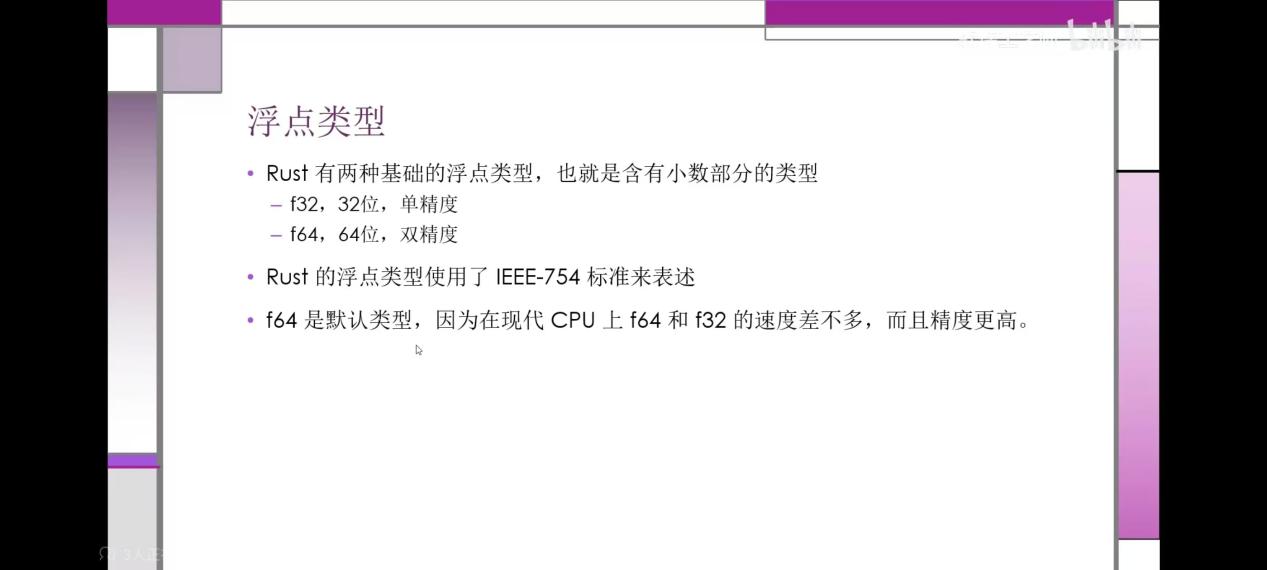
在rust语言中在声明变量时系统必须知道变量的类型，只要出现变量类型不清楚的情况，系统就会报错，这也是rust语言安全性高的原因之一。

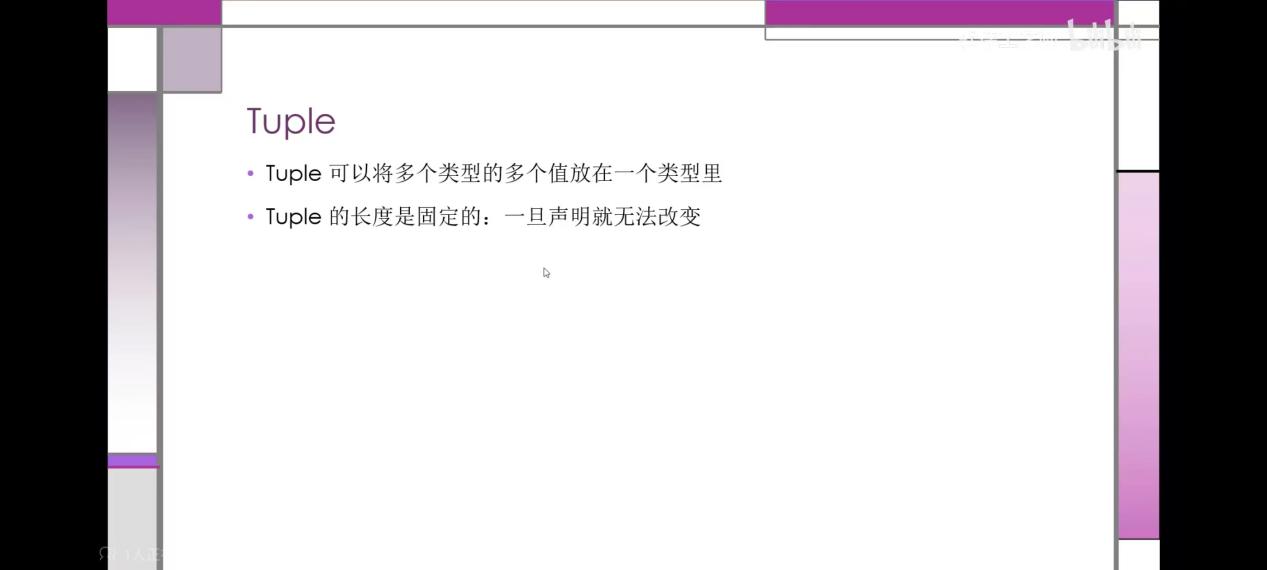
**整数溢出**

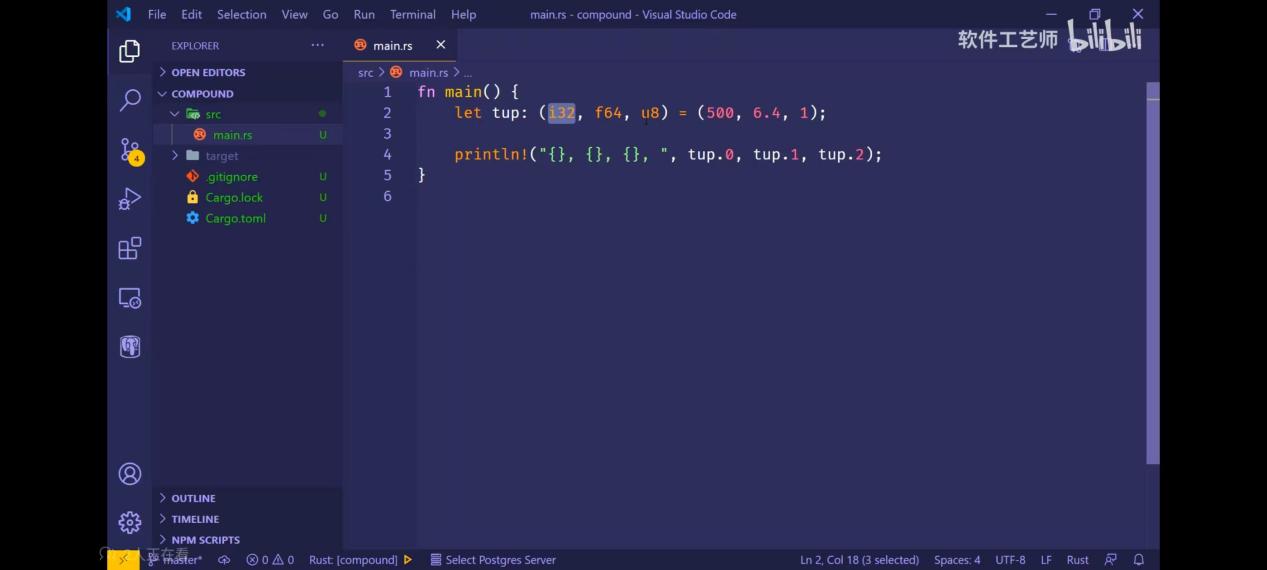
例如:u8的范围是0-255，如果你把一个8变量的值设为256，那么:-调试模式下编译:Rust会检查整数溢出，如果发生溢出，程序在运行时就会panic发布模式下(--release)编译:Rust不会检查可能导致panic的整数溢出·如果溢出发生:Rust会执行“环绕”操作:

-256变成0，257变成1..

·但程序不会 panic（即系统恐慌）





Tuple类型的创建]

1. 数组

数组也可以将多个值放在一个类型里

数组中每个元素的类型必须相同

数组的长度也是固定的

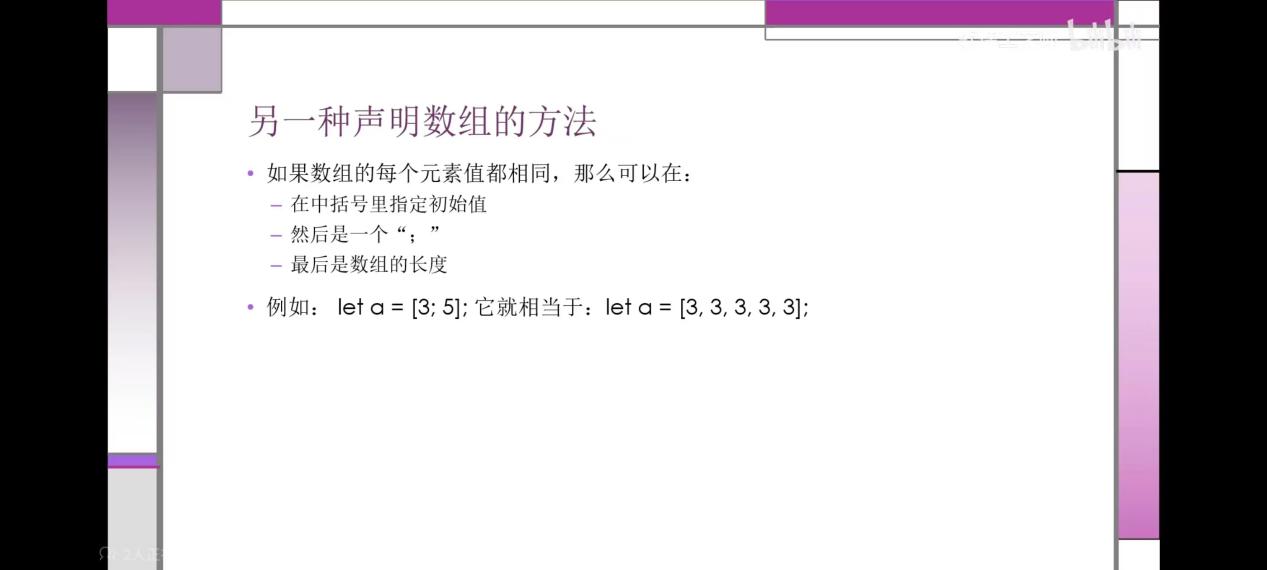
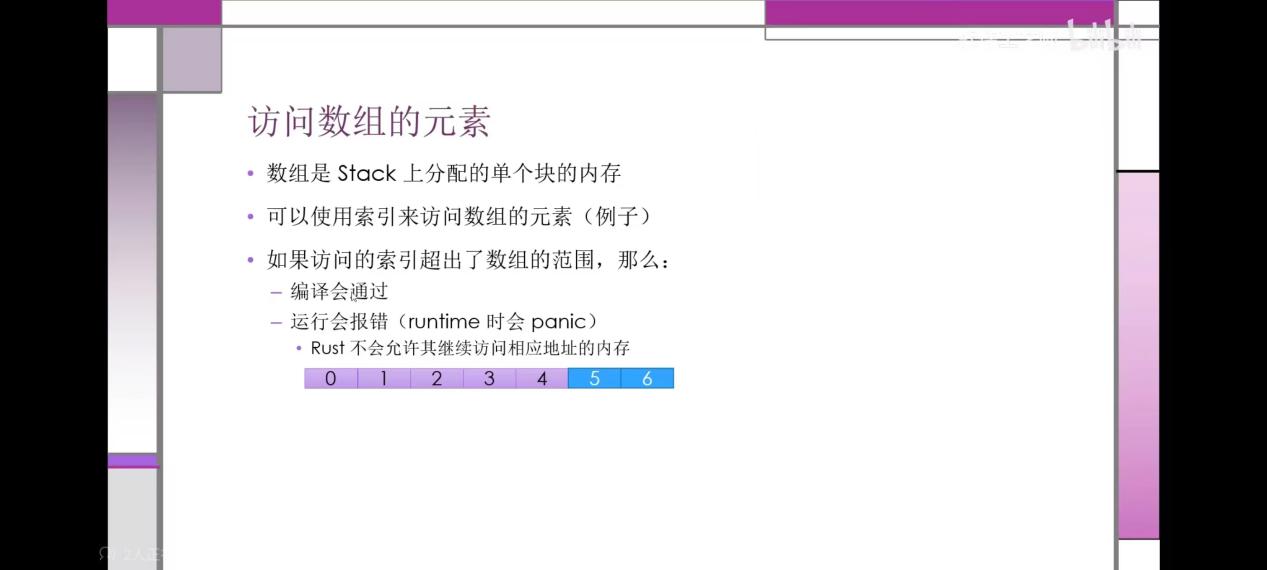
数组的好处：如果想让你的数据存放在stack(栈)上而不是heap(堆)上，或者想保证有固定数量的元素，这时使用数组更有好处

数组没有 Vector 灵活(以后再讲)。

Vector和数组类似，它由标准库提供Vector 的长度可以改变

如果你不确定应该用数组还是 Vector，那么估计你应该用 Vector。





**访问数组的元素**：

数组是 Stack上分配的单个块的内存

可以使用索引来访问数组的元素(例子)

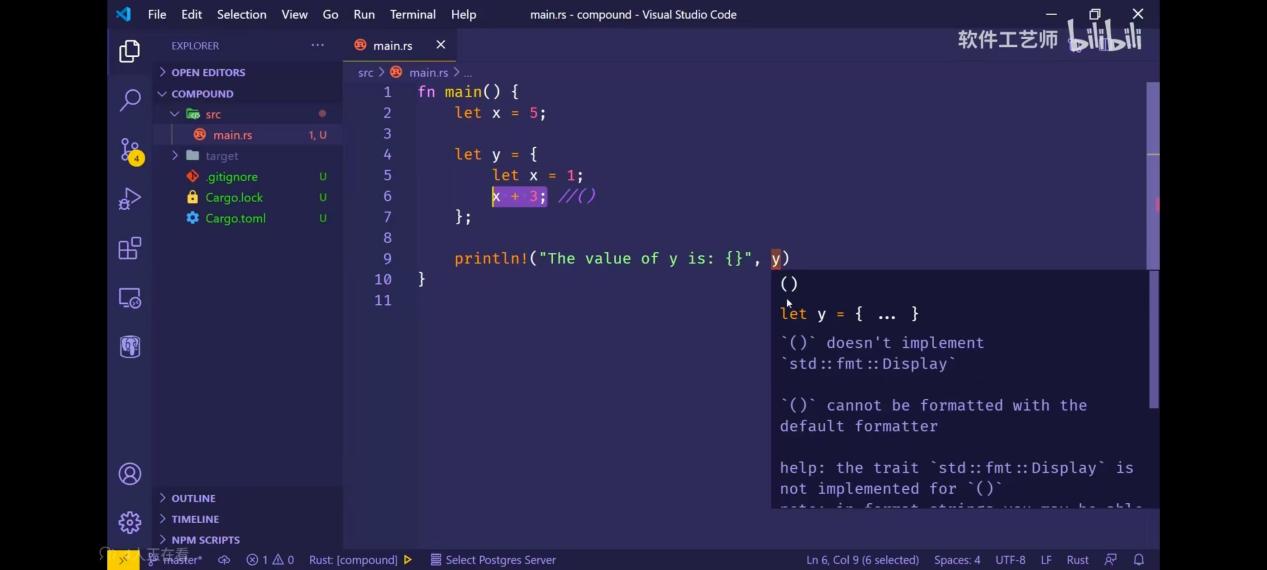
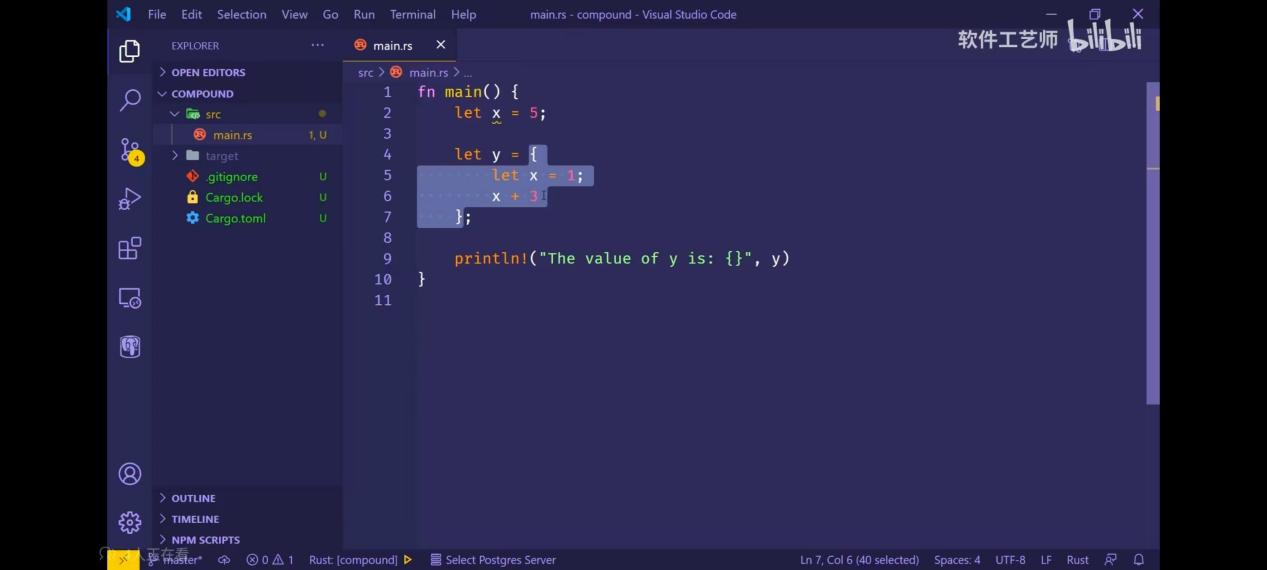
如果访问的索引超出了数组的范围，那么:编译会通过

运行会报错(runtime 时会panic)

RUst不会允许其继续访问相应地址的内存

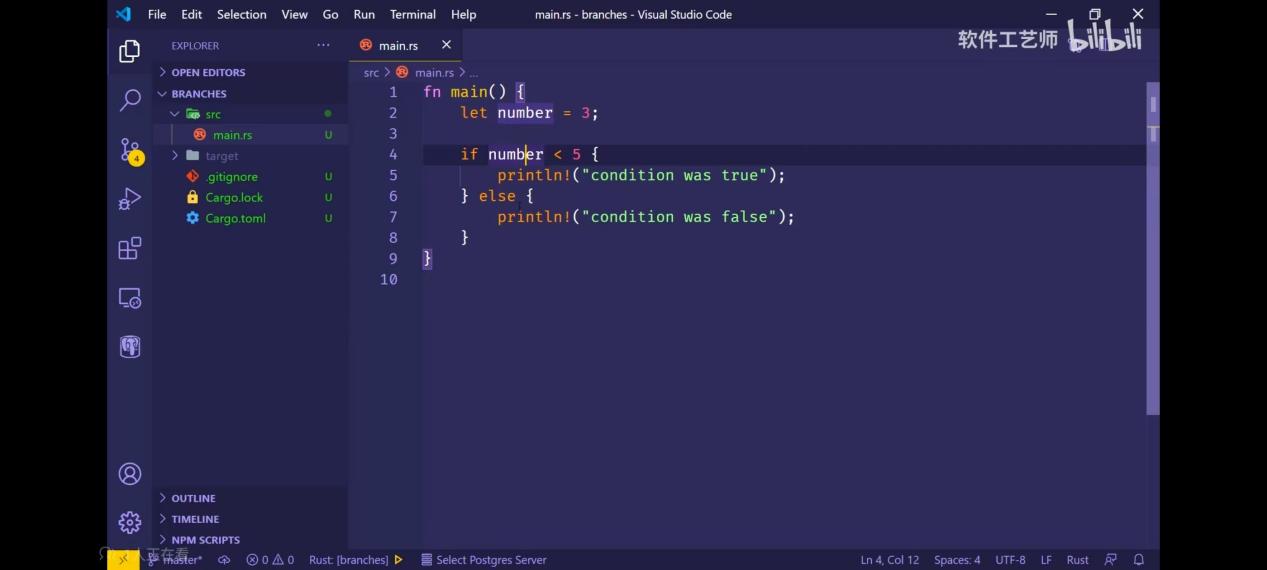
！！！在C++中，越界访问数组是不会报错的，会出现一个随机的东西，但在RUST中会在编译时报错。

1. 选择判断语句



有；时，代表这是一个语句，他的表达式相当于是一个空的（ ），意思是没有表达式，所以y接收的返回类型就是空的（ ）。

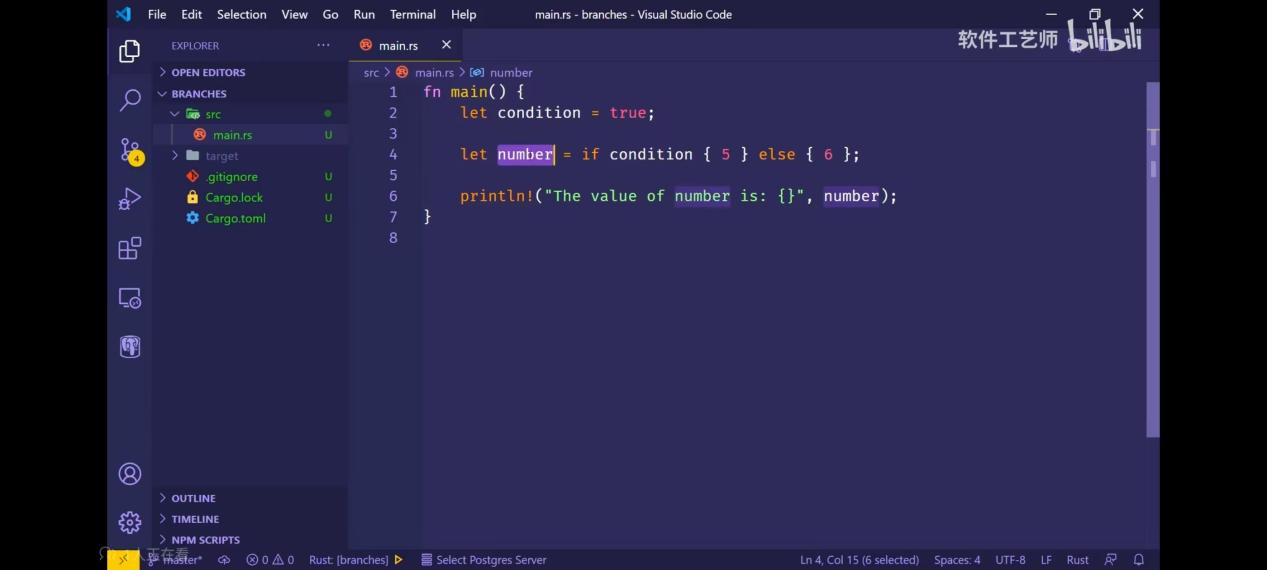
没有；时，最后一句的x+3就是表达式，所以y接受的返回值就是4





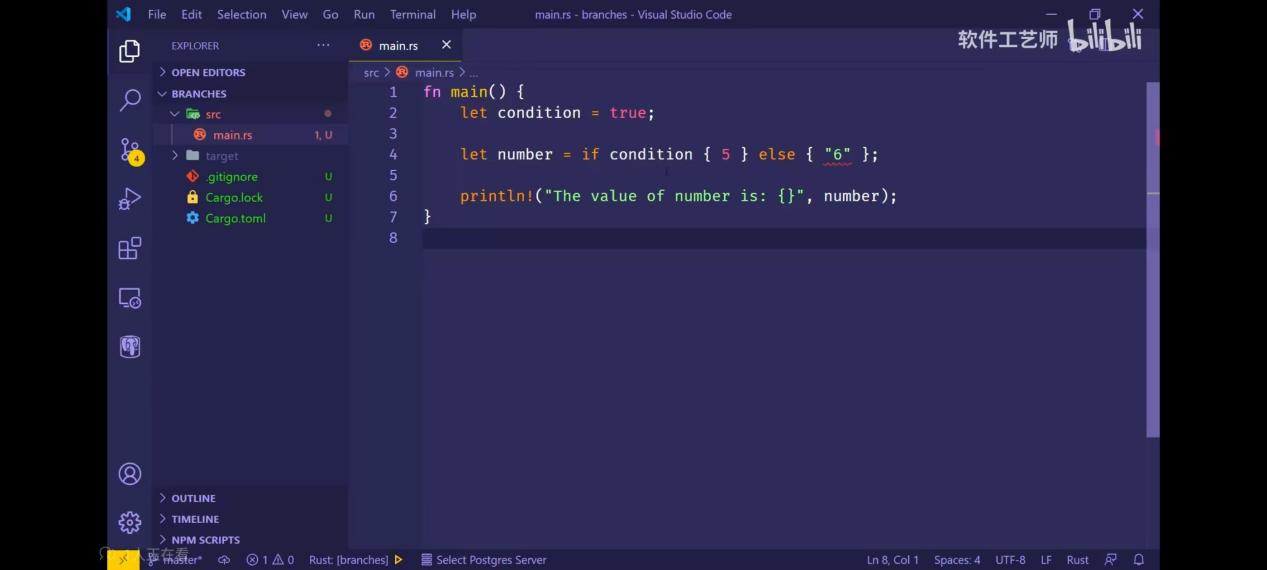
而且，if也可以是一个表达式



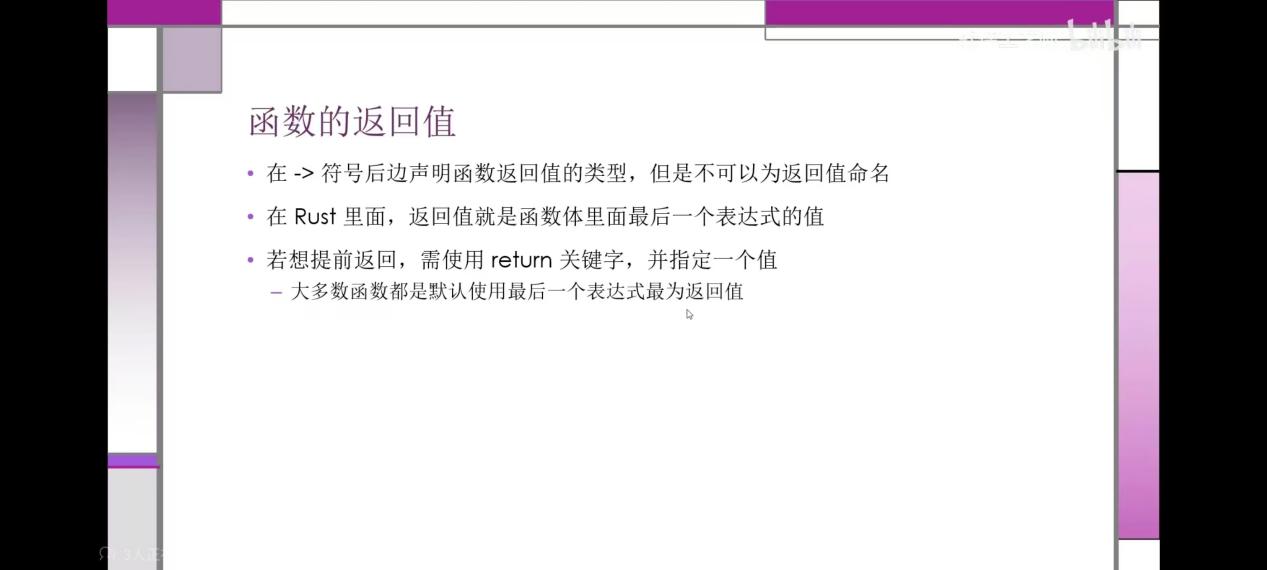


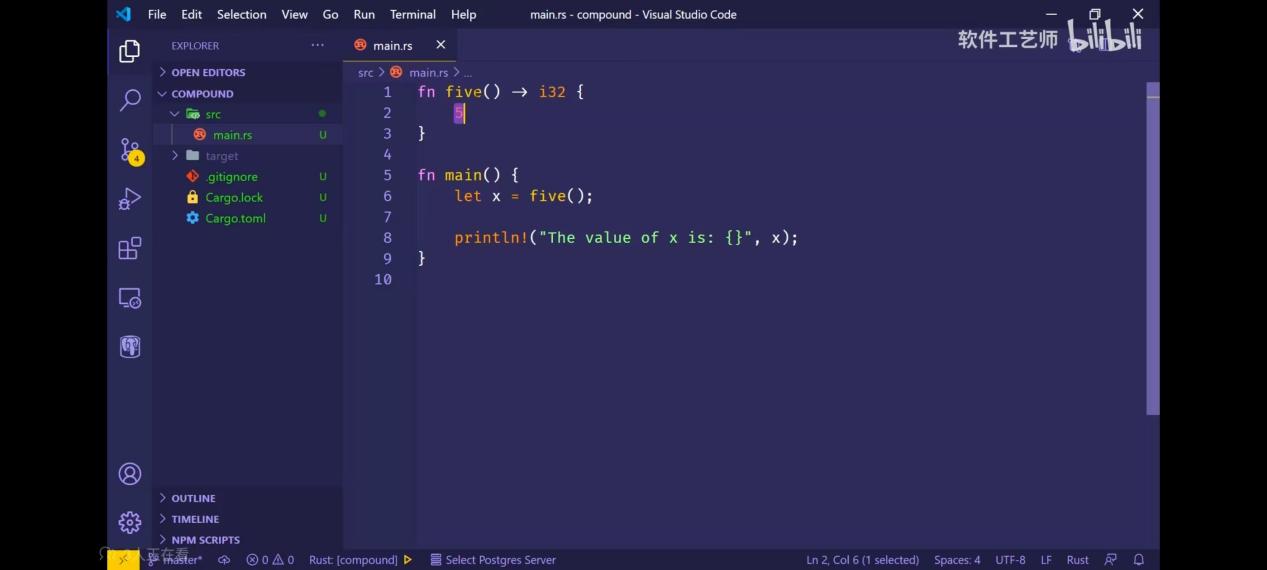
而且if else中的类型必须是相同的

因为rust是强类型语言，在编译时必须知道变量的类型是什么，这样足够安全，如果类型不同，系统就不知道number是什么类型，就会报错，下边是报错例子：



1. 函数

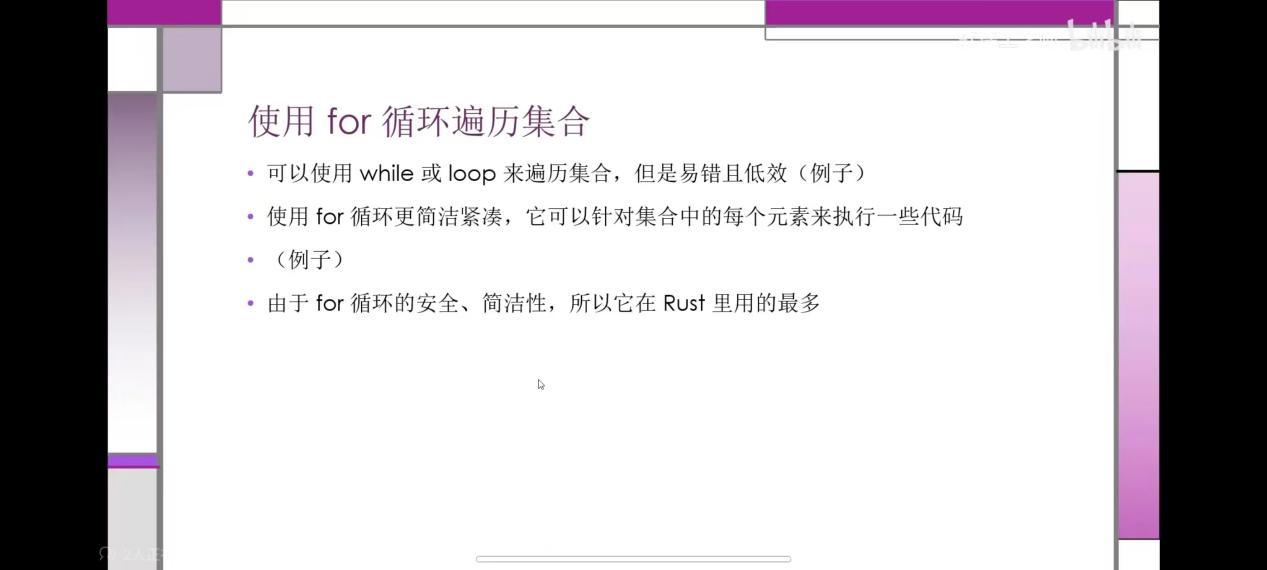


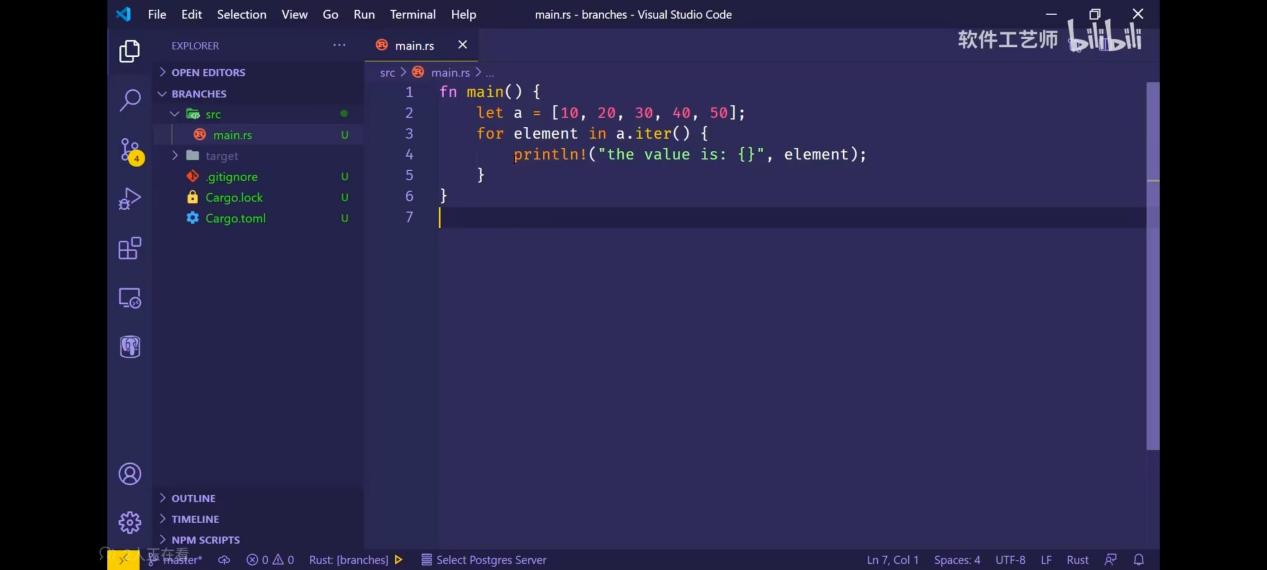


因此，在rust中，函数中的最后一个语句必须是表达式，这个表达式表示函数的返回值（也可以是return的返回语句）

小技巧，想要更改函数名，但这个函数名在程序中多次出现，可以选中一个之后按f2，然后进行更改，就可以把所有的函数名进行同时更改

1. 循环语句

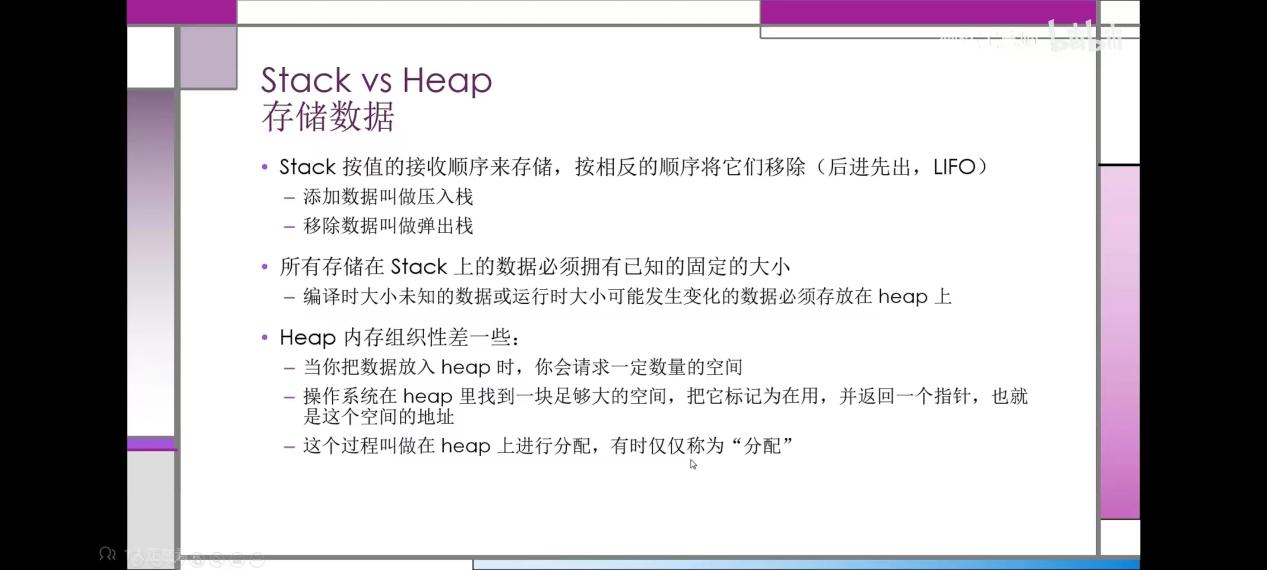




1. rust中的栈和堆

在像 Rust 这样的系统级编程语言里，一个值是在stack上还是在 heap 上对语言e的行为和你为什么要做某些决定是有更大的影响的

在你的代码运行的时候，Stack和Heap 都是你可用的内存，但他们的结构很不相同。





9.**所有权（我自己的理解）**：所有权主要是为了解决堆上数据内存分配量不确定所导致的内存不及时释放或者其他问题导致的内存浪费。

所有权解决的问题:

跟踪代码的哪些部分正在使用 heap 的哪些数据

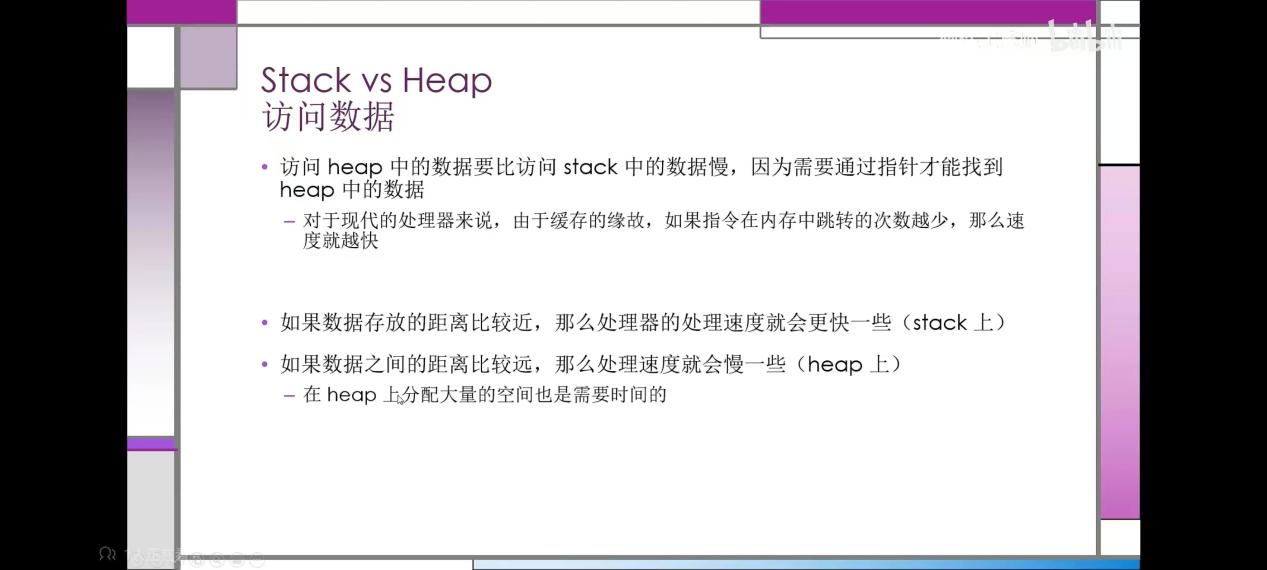
最小化 heap 上的重复数据量

-清理 heap 上未使用的数据以避免空间不足。

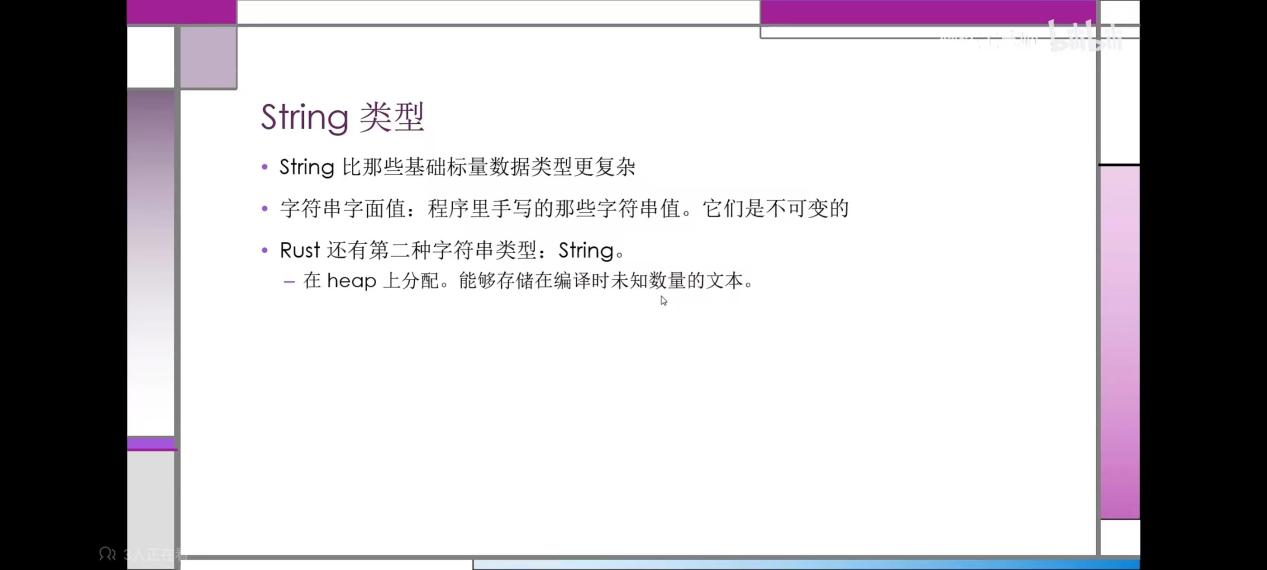
-旦你懂的了所有权，那么就不需要经常去想stack或 heap 了。

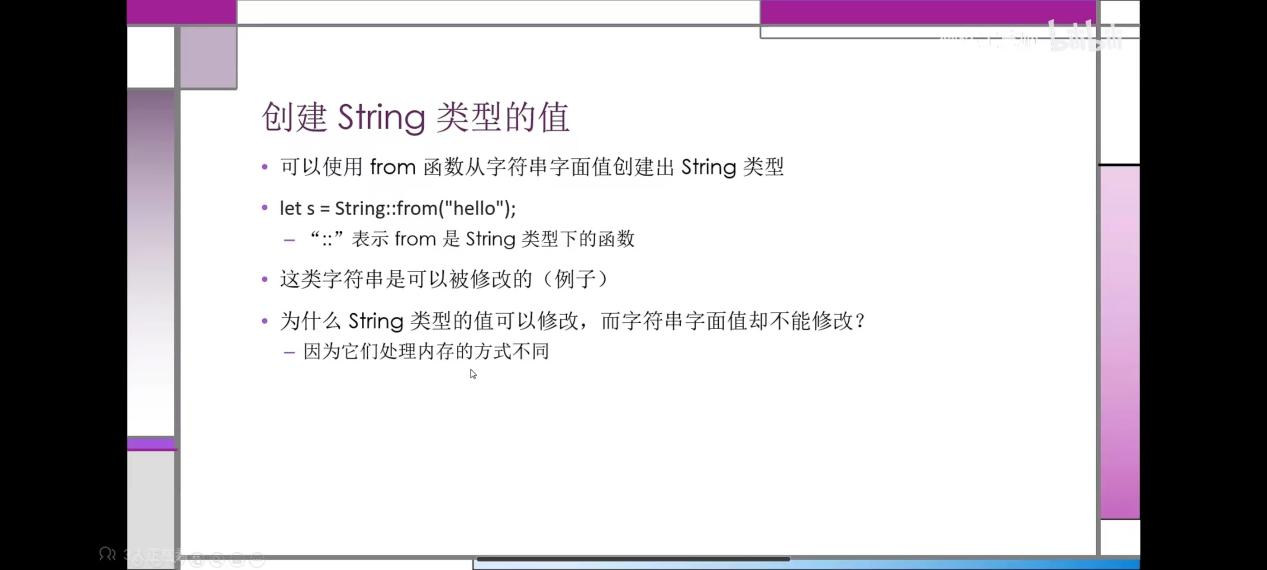
但是知道管理 heap 数据是所有权存在的原因，这有助于解释它为什么会这样工作





这里表示，因为堆上的数据的内存量不知道有多少，所以系统在分配内存时会分配足够大的空间给数据，也就导致了数据与数据有大量的空白内存使得数据直接距离比较远。

**10.string**



字符串字面值，在编译时就知道它的内容了，其文本内容直接被硬编码到最终的可执行文件里-速度快、高效。是因为其不可变性。

String 类型，为了支持可变性，需要在 heap 上分配内存来保存编译时未知的文本内容:

-操作系统必须在运行时来请求内存

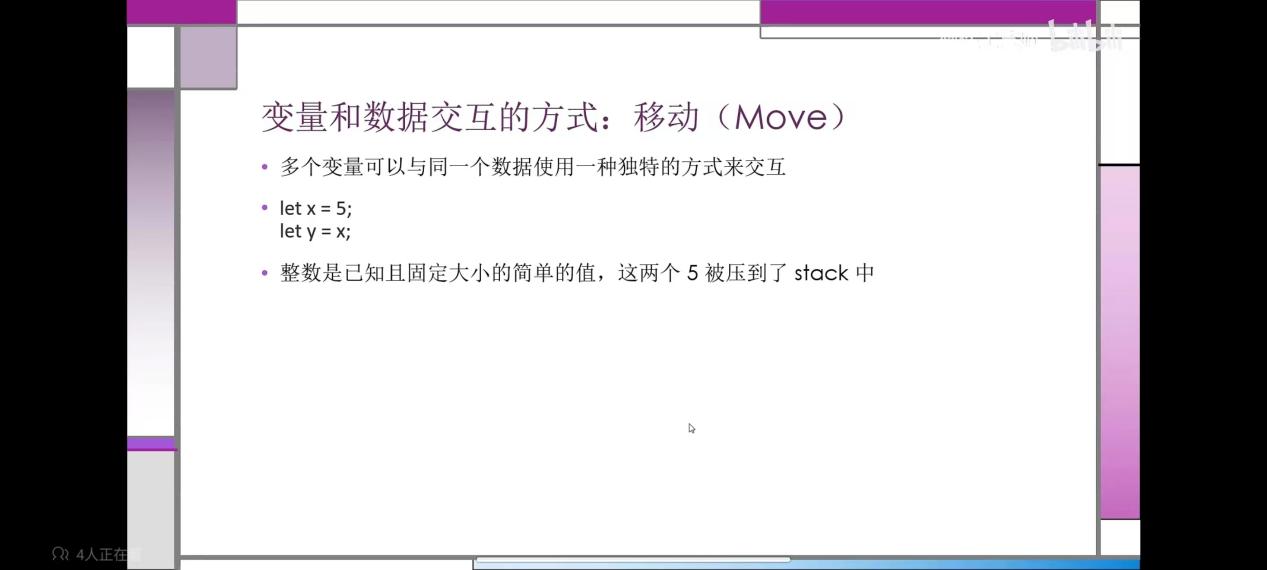
·这步通过调用 String::from 来实现

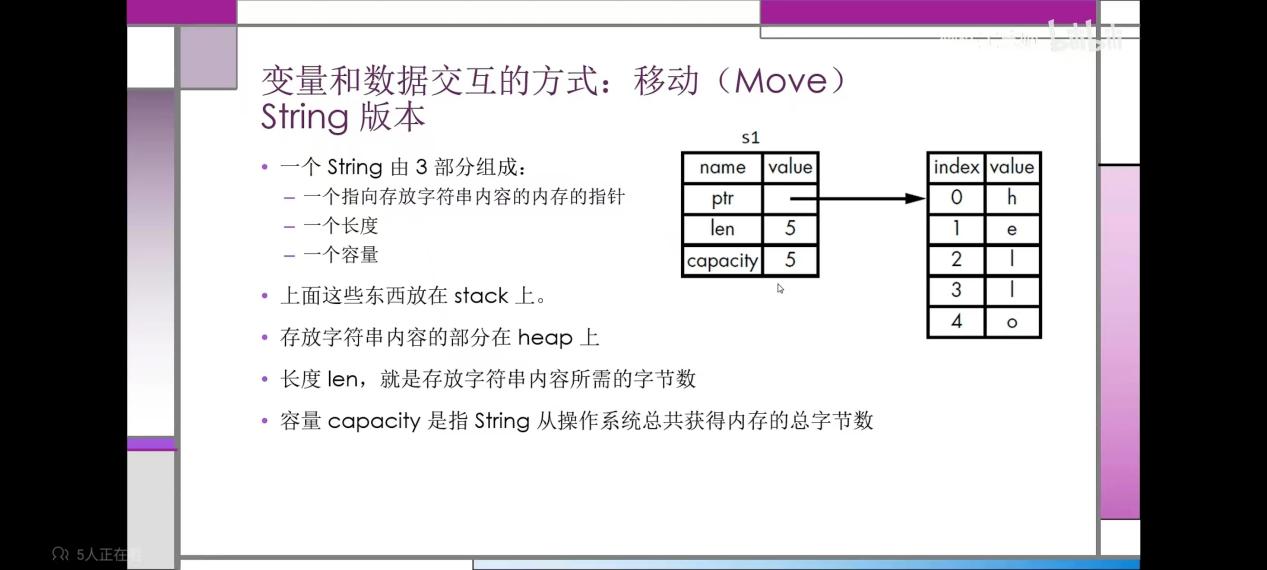
当用完 String 之后，需要使用某种方式将内存返回给操作系统。（其实是在堆上内存分配的所有权）

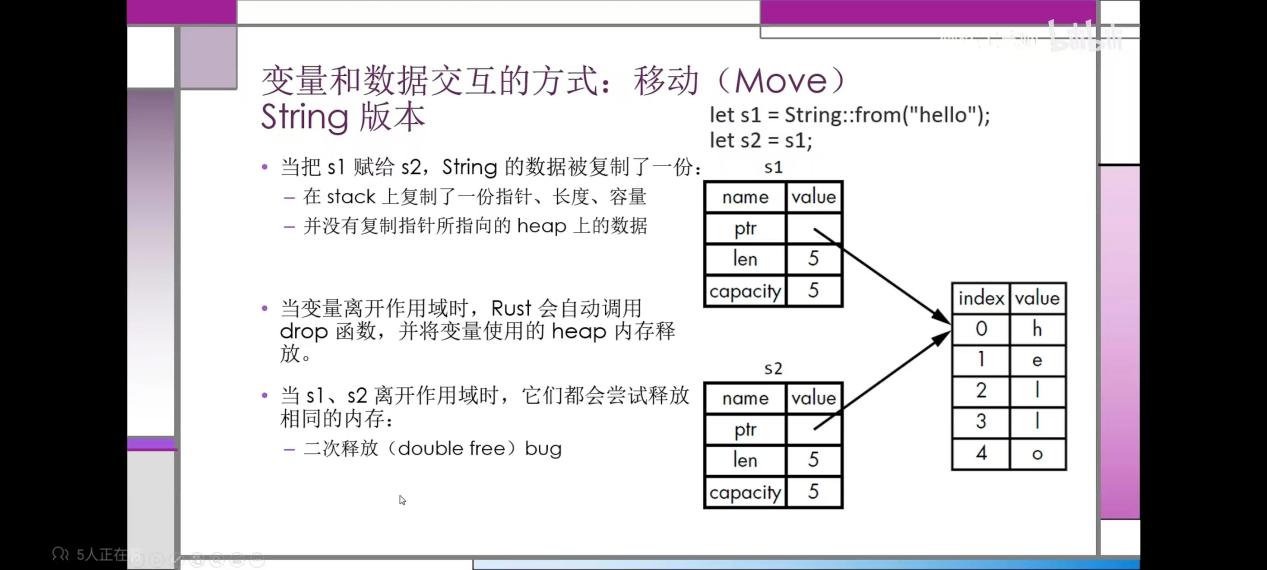
对于rust而言，对于某个值来说，当拥有它的变量走出作用范围时，内存会立即自动的交还给操作系统。

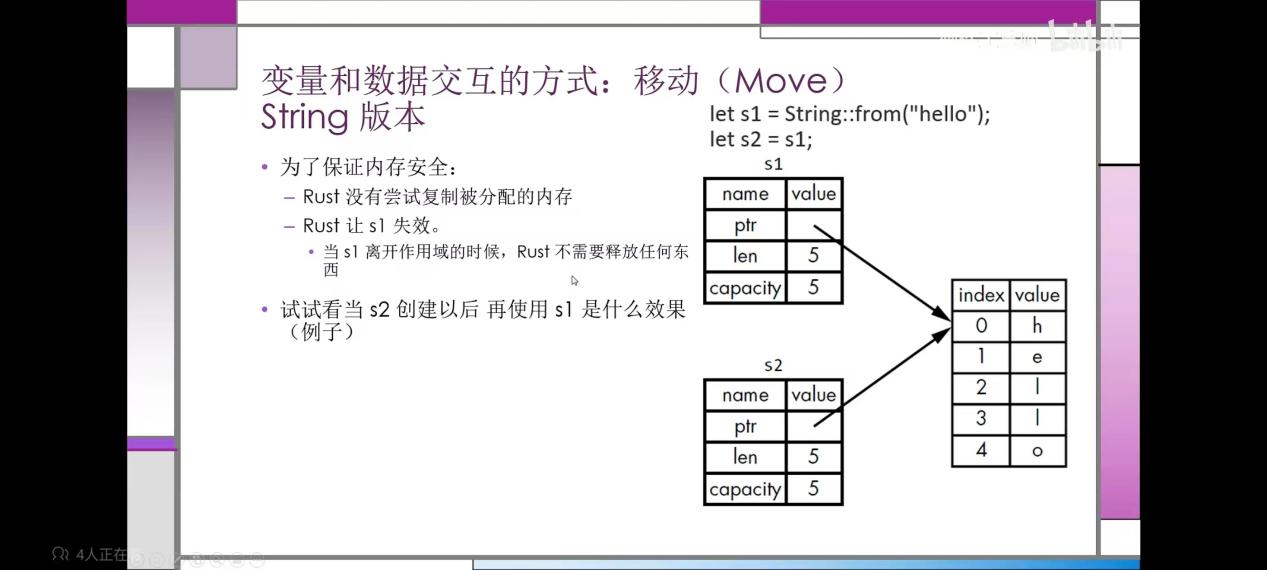
即超出生命周期，内存会自己调用drop函数自动释放内存，而C++中需要delete手动释放。

1. **rust的值与变量名（即值的所有者关系）**

****







我们发现当打印s1时，程序出现bug，显示s1没有值，s1发生了转移.

所以，任何一个值都只有一个拥有者，数字x=y是把x的值在栈中重新复制了一个相同的值（因为整数是固定大小已知的值），它的拥有者是y

但是对于string来说，不会在heap种重新复制新的值，而只是在stack种复制一个新的属性表，而这个新的属性表s2取代s1成为新的拥有者

这样也可以避免二次释放内存导致的bug。

那么，我们怎么在保存s1的情况下把s1赋值给s2呢：用到clone

