第七章的标题是Managing Growing Projects with Packages, Crates, and Modules，介绍了Rust的项目结构。7.1介绍了crate、module和package的概念、从属关系，7.2-7.5介绍了Rust的相关机制，介绍了如何组织一个项目。

在介绍组织项目的具体内容之前，先看一下7.1的部分，包和crates。书里说一个crate是“the smallest amount of code that the Rust compiler considers at a time“，编译器一次考虑的最小代码量。再具体一点，crate指src文件夹下的main.rs、lib.rs和src/bin文件夹内的.rs文件。

比如说看一下这道题

Imagine you see a Rust package foobar with the following files:

foobar

├── Cargo.toml

├── build.rs

└── src/

├── main.rs

├── util.rs

├── lib.rs

└── bin/

└── alt.rs

How many crates does this package contain? Write your answer as a digit, e.g. 0, 1, and so on.

这道题的答案是3，一共有main.rs、lib.rs和bin文件夹下的alt.rs三个文件。

包是一组一个或多个crate的捆绑。而模块是crate的内部结构，比如说在一个lib.rs里可以写很多mod，它们之间的关系是package > crate > module。

以上是7.1涉及的内容。在明确了基本概念后，7.2-7.5四节分别介绍了定义模块、路径、use关键字和用多个文件组织模块。

7.2和7.5介绍的是项目的组织结构，7.3-7.4介绍的是路径和路径的使用。

/\*我用一个简单的例子来介绍一下7.2和7.5，这个项目是根据原书里的代码扩展的，模拟了餐厅中顾客点餐的场景。这个项目包含一个main.rs、一个lib.rs以及一些其它文件，结构是这样的

Src

├── lib.rs

├── front\_of\_house.rs

└── front\_of\_house/

└── hosting.rs

整个项目主要包含两部分，front\_of\_house和back\_of\_house（前厅和后厨），其中前厅模块分散在了多个文件夹里，并且是pub的，后厨模块完全实现在lib.rs中，是private。

从这个例子对比public和private，

模块分private和public，默认为private，实现public需要加pub关键字。例如在这个示例项目中，前厅部分是public，因为前厅与顾客直接联系，而后厨部分没有pub关键字，因为顾客不需要自己进入后厨。

/\*此外，如果父模块使用子模块，子模块内的函数、变量、结构体等也需要标记为pub，但子模块可以访问父模块内的private量。\*/

注意枚举类型和结构体比较特殊：对于（这里的）枚举类型，只要这个量被标记为pub就可以访问，但对于结构体，不仅要结构体本身是public，结构体的成员也需要被标记为public。例如在这里，尽管结构体Breakfast是public的，season\_fruit仍然无法访问，只能访问标记了的toast。但在枚举类型Appetizer（[ˈæpɪˌtaɪzəz] ）被标记为public后，不需要其它处理就可以访问这里的salad和soup。

将模块分到多个文件内，这里的front\_of\_house已经这么做了。

首先看根部的crate中的main.rs。在use之前，有一句mod front\_of\_house;。这句话告诉rust，项目中有一个front\_of\_house模块，然后rust就会去src/front\_of\_house.rs中寻找这个模块。而front\_of\_house中有一个hosting模块，rust就会去src/front\_of\_house/hosting.rs寻找这个新的模块，用这种方法，可以将庞大的模块层层相扣，分散到不同的文件中。

到这里基本上已经可以组织rust项目。但是还有一个部分要说明（刚才的例子），就是这两句

crate::front\_of\_house::hosting::add\_to\_waitlist();

这里包与包之间使用::连接的，但是在包之前，还有一个crate。src/main.rs 和 src/lib.rs 被称为 crate 根。这两个文件中的任何一个的内容都形成了一个名为 crate 的模块，该模块位于 crate 模块结构的根目录。编译器会从crate根来找后面的模块、函数的位置。从这里就引入了7.3的内容，路径。

和一般的路径一样，rust中的路径也分为绝对路径和相对路径两种。刚才提到了crate可以视作模块的“根“，所以绝对路径都是从crate开始，沿着子模块一路向下。而相对路径是由当前开始的。另外，相对路径不只可以“向下”使用内层模块，也可以用super关键字向上。

7.2-7.4是相互衔接的。既然了解了路径，就要使用路径并设法将其简化，需要使用use关键字。

例如，有模块D<C<B<A，与其一直重复使用A::B::C::D，写一句use A::B::C::D;后，就可以直接使用D。需要注意的是，use关键字只在它所属的作用域中起作用。此外，use关键字还有一些相关的语法。

use std::fmt::Result;

use std::io::Result as IoResult;

用as关键字重命名

use std::{cmp::Ordering, io};

use std::io::{self, Write};

引入前缀相同的部分

use std::collections::\*;

引入路径中所有公共项目