Chapter10一共有3节，分别是泛型、trait和生命周期。相对来说这3节的内容连接没有那么紧密，在书中10.1泛型和10.2trait是两个相互独立又相辅相成的部分，而10.3中的生命周期是一种特殊的泛型。

Rust的泛型有点类似于C++的模板。按照书上的说法，在结构体、函数等定义部分使用泛型，可以让这些函数、结构体能应用于多种不同的数据类型。在泛型使用中，书上大致分成了四类：结构体、枚举、函数、方法。这四类在语法上是大致相同的，不过在实际使用的时候，尤其是函数和方法，有一些特殊的地方。先来看一下基本语法。

结构体

struct Point<T, U> {

x: T,

y: U,

}

枚举

enum Option<T> {

Some(T),

None,

}

函数

fn largest<T>(list: &[T]) -> &T

方法

struct Point<T> {

x: T,

y: T,

}

impl<T> Point<T> {

fn x(&self) -> &T {

&self.x

}

}

语法就是在名字后加尖括号，括号内是泛型名称，可以有一个或多个。在下文中，这些名称可以被当作类型使用。

在定义之外，泛型有一些需要注意的点。

首先，对于某个结构体的方法，方法可以包含不属于结构体的泛型。struct Point<T> {

struct Point<T> {

x: T,

y: T,

}

impl<T> Point<T> {

fn x(&self) -> &T {

&self.x

}

}

此外，尖括号内的不一定是泛型，可以是某个具体类型，用于限制

impl Point<f32> {

fn distance\_from\_origin(&self) -> f32 {

(self.x.powi(2) + self.y.powi(2)).sqrt()

}

}

最后，泛型的使用是有一定限制的。例如对于这段代码

fn largest<T>(list: &[T]) -> &T {

let mut largest = &list[0];

for item in list {

if item > largest {

largest = item;

}

}

largest

}

fn main() {

let number\_list = vec![34, 50, 25, 100, 65];

let result = largest(&number\_list);

println!("The largest number is {result}");

let char\_list = vec!['y', 'm', 'a', 'q'];

let result = largest(&char\_list);

println!("The largest char is {result}");

}

无法通过编译。因为代码里的 if item > largest，显然，不是所有类型都重载了>号。更准确地说，看一下编译错误的内容，只有拥有std::cmp::PartialOrd的类型可以使用这个符号。这个std::cmp::PartialOrd被称作一个trait。不同的类型有不同的trait，很多时候我们不需要一个结构体或者函数可以应用于所有的类型，只需要它应用于有某些特质的类型。Rust使用trait来区分，就进入了10.2的部分。

关于trait的部分，以官方代码的整合版为例。

Trait的使用大概可以分为3个部分：

1. 定义trait，对应代码中的这一部分
2. 对于具有trait的相关结构体，实现trait中的函数。
3. 使用trait。

而使用trait在书里分为两种，一种是将trait作为返回值，一种是用trait有条件地实现方法。我比较关注后一种。