inside-rust-std-library

本书主要对RUST的标准库代码进行分析。
本书尽可能给读者找出一条标准库代码的阅读脉络。同时，分析不仅仅针对代码的功能，也针对代码背后的需求及若干代码设计的思路。
C语言精通的标志是对指针的精通。RUST的裸指针也是RUST的最基础及最核心的难点之一。 所以，将裸指针及相关的内存模块作为代码分析的起始点，熟悉了裸指针及内存，自然也就对所有权，借用，生命周期的本质有了深刻的理解，RUST语言的最难关便过了。
泛型是RUST不可分割的语法之一，而对于其他语言，没有泛型不影响语言的使用。泛型及基于trait的泛型约束是RUST的另一个代码基础。
针对基本类型的分析，可以看到RUST利用trait语法使之具备了无限的扩展性，这是RUST更有表现力的语法能力的展现。
Option/Result<T,E>等类型实际完全是由标准库定义的，并不是RUST语言最底层的基本内容，可以从代码分析中发现这一点。
所有的运算符都可以重载，且可以跨越类型重载，RUST的运算符重载揭示了RUST很多的编码奥秘及技巧。
Iterator加闭包是函数式编程的基础构架，Iterator的适配器构成了函数式编程的基础设施，RUST完整的实现了这些内容，并且几乎为每个类型都实现了迭代器，并尽可能的为函数式编程做好了准备。
Cell/RefCell/Pin/Lazy代码证明了在RUST的基础语法下，如何创造性的解决问题。
Box/RawVec是两个堆内存申请的基本结构，善用这两个结构，除非写内存管理，基本上就不必再接触底层的堆内存申请及释放。
每一个智能指针实际上也是RUST对经典的数据结构实现的精妙例程。
RUST对不同操作系统的适配让程序员不必象C那样再重复的耗费精力并且还沾沾自喜于此份工作。
仅支持异步编程的async/await，Future也体现了RUST的作最基础的工作的态度。
…
…

(This book focuses on the analysis of RUST’s standard library code.
This book is as far as possible to find a reading context for the standard library code. At the same time, the analysis is not only for the function of the code, but also for the requirements behind the code and some ideas of code design.
The hallmark of C proficiency is mastery of pointer. The raw pointer in RUST is also one of the most basic and core difficulties of RUST. Therefor, the raw pointer and associated memory modules are used as the starting point for code analysis, and the familiarity with raw pointer and memory naturally leads to a profound understanding of the nature of ownership, borrowing, and the life cycle. The hardest part of RUST is over.
Generics are an integral part of RUST’s syntax, and for other languages, the absence of generics does not affect language use. Generics and their trait - based generic

constraints are another code base of RUST.
Based on the analysis of primitive types, it can be seen that RUST makes use of trait syntax to make primitive types infinitely extensible, which is a demonstration of RUST’s more expressive syntax ability.
Types such as Option/Result<T, E> are actually defined entirely by the standard library and are not the basic content of the lowest level of the RUST language, as you can see from code analysis.
All operators can be overloaded and can cross type overloading. Operator overloading in RUST reveals many of RUST’s coding secrets and tricks.
Iterator plus closures are the foundation of functional programming. The adapters of Iterator make up the infrastructure of functional programming. RUST implements them completely, implementing iterators for almost every type and preparing them as well as possible for functional programming.
The Cell/RefCell/Pin/Lazy source code demonstrates how creative problem solving can be done within RUST’s basic syntax.
Box/RawVec are the two basic structures of heap memory allocation and freeing.
Each smart pointer is actually an elegant routine of RUST’s implementation of classical data structures.
RUST’s adaptation of different operating systems saves programmers from the repetitive effort and complacency of C.
Supporting only async/await for asynchronous programming, Future also embodies RUST’s attitude of doing the most basic work.
… …)