NobleProg

Rust 高级开发

培训计划 - 第一天上午

- Rust 介绍
- 安装 Rust
- Rust 语法基础
- Rust 基本数据结构
- Cargo 和 crates.io
- 练习：使用 clap 分析命令参数
- 所有代码均存于 Github

NobleProg

培训计划 - 第一天下午

- 内存所有权
- AsRef, AsMut, Iterator
- Trait 和 macro!
- 锁与线程
- 线程程间通讯
- 练习： Server-client TCP 通讯

NobleProg

培训计划 - 第二天上午

- 特殊数据类型：智能指针
  - Box
  - RC
  - RefCell
  - Arc
  - AsRef/AsRefMut Trait
- JSON/YAML 解析 (serde)
- 练习： Server-client YAML 通讯

NobleProg

培训计划 - 第二天下午

- 日志系统
  - logging
  - env_logger
  - rust-syslog
  - log4rs
- 使用和输出 C 库 (Foreign Function Interface)
- 练习：使用 Rust 编写 Server-Client YAML 客户端的 C 库

NobleProg

培训计划 - 第三天上午

- 异步编程：mio, tokio, impl Future
- Web 框架： yew + rocket
- 练习：
  - 使用 tokio 和 reqwest 访问网页
  - 创建 async crate AsyncSleep
  - 创建基于 Stream 的 RepeatTimer

NobleProg

培训计划 - 第三天下午

- 单元测试
- 集成测试
- 持续集成和持续部署
- 练习：
  - Github Aciton for CI
  - Quay.io for CD

NobleProg

Rust 介绍

- Mozilla 创造， Rust 基金会维护
- 强数据类型
- 保证内存和线程安全
- 学习曲线陡峭
- 极其严格的编译器
- 学习建议
  - https://doc.rust-lang.org/book/
  - 《深入浅出 Rust 》
  - Pull Request to Community

NobleProg

安装 Rust

- rustup
  - curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
- cargo
  - `cargo new`
  - `cargo build --release`
  - `cargo publish`
  - `cargo fmt`
  - `cargo clippy`
  - `cargo doc`
  - `cargo install`

Rust 语法基础 1

```rust
// SPDX-License-Identifier: Apache-2.0

fn count_odd(data: &[u16]) -> usize {
    data.iter().filter(|i| *i % 2 == 1).count()
}

fn main() {
    let a: [u16; 4] = [1u16, 2, 3, 4];

    let count = count_odd(&a);

    println!(
        "Got {} odd number{}",
        count,
        if count > 1 { "s" } else { "" }
    );
}
```


NobleProg

Rust 语法基础 2 – Error Handling

```rust
#[derive(Debug, Clone)]
struct AbcError {
    kind: ErrorKind,
    msg: String,
}
impl std::fmt::Display for AbcError {
    fn fmt(&self, f: &mut std::fmt::Formatter<'_>) -> std::fmt::Result {
        write!(f, "{:?}:{}", self.kind, self.msg)
    }
}
impl std::error::Error for AbcError {}
#[derive(Debug, Clone, Copy)]
enum ErrorKind {
    NotFound,
}
// Raise error when no odd found
fn count_odd(data: &[u16]) -> Result<usize, AbcError> {
    let count = data.iter().filter(|i| *i % 2 == 1).count();
```


NobleProg

Rust 基本数据结构

- Primitive Type: u8, u16, …
- Option<T>
- Result<T, E>
- Enum
- String and &str
- Vec and &[]
- std::collections::{HashMap, HashSet}
- std::collections::{BTreeMap, BTreeSet}

NobleProg

Cargo 和 crates.io

- Cargo -- Rust package manager
- crates.io -- Rust community’s package registry
- Cargo.lock
- cargo vendor

练习：使用 clap 分析命令参数

- `foo_cli subcommand_a`
- `foo_cli subcommand_b --arg1 value1`
  - The `--arg1` is argument of `subcommand_a`
- `foo_cli subcommand_c -arg2 value2`
  - The `--arg2` is argument of `subcommand_b`
- `foo_cli -v subcommand_b -arg1 value1`
  - The `-v` should be global argument available to all subcommands

内存所有权

- 初始化
- Move VS. Copy VS. Clone
- None Lexical Lifetimes
- 共享不可写， 可写不共享
- pub fn drop<T>(_x: T) {}
- Unsafe!→ safe: e.g. nix::unistd::sysconf

NobleProg

AsRef, AsMut, Iterator

- The `as_ref()` for in-place access
- The `as_mut()` for in-place modification
- std::collections::hash_map::Entry
- map.values_mut().map(|v| *v *=2)
- opt_vec.iter().filter_map(F).collect()

Trait 和 macro!

- Trait provided method: e.g. std::error::Error
- Trait required method. e.g. std::convert::TryFrom
- Macro 去除重复代码的最后手段
- 常用 Trait: PartialEq(NaN), Hash

锁与线程

- `Arc<Mutex<ShareState>>`

```
pub fn spawn<F, T>(f: F) -> JoinHandle<T>
where
    F: FnOnce() -> T + Send + 'static,
    T: Send + 'static,
```

线程间通讯

- Go lang:
  - Do not communicate by sharing memory;
  - instead, share memory by communicating.
- mpsc(Multi-producer, single-consumer)
- Socket: e.g. TCP/UDP/Unix

练习：Server-client TCP 通讯

- Server 和 Client 都是独立 CLI 程序
- Server 监听 TCP 127.0.0.1:8766
- 每个 Client 对应一个独立的 Server 线程
- Client 向 8766 端口发送 chrono::Utc::now()
- Server 使用单一线程在终端打印出收到的当前时区时间
- 提示
  - TcpListener, TcpStream::read_exact()
  - Utc::now().timestamp(), timestamp_subsec_nanos()
  - DateTime::<Utc>::from_timestamp()
  - DateTime::to_rfc3339_opts()

NobleProg

智能指针

- Box<T>: Point to Data on the Heap
- Rc<T>: Reference Counted Read Only Heap Data
- RefCell<T>: Dynamic(runtime) borrowing (unsafe!)
- Arc<T>: Atomically Reference Counted
- Automatic referencing and dereferencing
  - Only for method
  - Only for `&`, `&mut`, `*`
- `Deref Trait` for custom smart pointer

NobleProg

JSON/YAML 解析 (serde)

- Framework: serde
  - `Trait serde::Serializer`
  - `Trait serde::Deserializer`
- Implementation: serde_json, serde_yaml, serde_ini, etc
- 自定义解析
  - #[serde(deserialize_with = "crate::deserializer::u16_or_string")]
  - impl<'de> Deserialize<'de> for MyStruct
  - #[serde(try_from = "FromType")]

练习： Server-client YAML 通讯

- 基于 Server-client TCP 练习
- Client 发送 YAML 包含版本号，主机名和时间
- Server 收集并打印在终端
- 版本号需要支持数字 1 和字符串”1”，可选，默认 1
- 时间格式需要支持 RFC 3339 和 RFC 2822
- 主机名为可选项

日志系统

- Framework: `rust-lang/log`
  - `Trait log::Log`
- `Implementation:`
  - `env_logger`
  - `rust-syslog`
  - `log4rs`
- Runtime static:
  - `lazy_static`
  - `std::cell::LazyCell`
  - `once_cell`

使用和输出 C 库 (FFI)

- `crate-type = ["cdylib","staticlib"]`
- Input pointer for a string: `char *`
- Output pointer for a string: `char **`
- Output pointer for an opaque struct: `struct abc **`
- 修复 SONAME
- 内存泄漏测试
- 使用 C 库创建 Python 模块

NobleProg

练习：Client C library

- 基于 Server-Client YAML
- 输出 libcs_yaml.h（内容如下） 和 libcs_yaml.so.0
- 保存 log 到 char * (较难，可选)
- 提供
  - int report_time(char **err_msg, char **log);
  - void log_free(char * log);
- 提供 make check 检查内存泄漏

NobleProg

Rust 异步编程 1/2

- Event driven SYNC way: mio (based libc::epoll)
- Framework:
  - async/await 关键字
  - Trait std::future::Future
- Executor Implementations:
  - Tokio
  - Async-std
  - smol

Rust 异步编程 2/2

- 使用 async crates
  - 选择 Executor
  - 兼容性检查
  - 使用 async/await 关键字
- 提供 async crates
  - 兼容性高：自定义 Reactor 来调用 waker
  - 易用性高：使用 Executor 的 Reactor

NobleProg

Web 框架

- 前端：
  - yew(with WebAssembly)
  - iced
- 后端：
  - Rocket
  - hyper
- Demo: Yew, Rocket

练习 异步编程

- 基于 Server-client YAML 练习
- 提供客户端 async crate
  - async fn report_time()
- 创建 async fn AsyncSleep::new(Duration)，使用 tokio Async reactor
- 创建 async fn AsyncSleep::new(Duration)，基于线程 waker
- 使用 futures::stream::Stream 创建 async fn RepeatTimer::new(interval)

编写测试

- 单元测试
  - `cargo test --test-threads=1 --show-output`
  - 访问 `pub(crate) fn`
  - 测试间共享 `utils`
- 集成测试
  - Fixture: setup/cleanup
  - pytest

持续集成 CI 和持续部署 CD

- CI
  - 环境隔离：CI host, OS, Network
  - Merge blocker
  - Tiering
  - Demo: Github Action
- CD
  - 自动化并防止人工错误
  - Demo: container 自动更新

NobleProg

练习：构建 CI 和 CD

- 基于 Server Client YAML
- 使用 Github Action 执行 CI
  - cargo test
  - cargo clippy
- 使用 Quay 自动更新包含 server 和 client CLI 的 container

NobleProg

Thanks!

The World's Local Training Provider