Substrate_ 入门课第8期
Course 3【Rust基础 - 作业点评】- 助教腾达 林涛

课程回顾

Rust基础

数据类型
流程控制
异常处理
Cargo 项目管理工具

Node Template 代码导读

https://whisperd.tech/post/substrate_node_template_guide/

Rust基础

流程控制

```
fn match_Literal_variable(){
        //Matching Literals 匹配字面值
        let x =2;
        match x {
            1 => println!(  “one”  ),
            2 => println!(  “two”  ),
            _ =>println!("anything") //不在意其他值，使用 _ 表示不绑定变量
        }
}

        let y =4;
        let z=false;
        match y {
            2 |3 |4 | 5 if z => {println!(  “matched y”  )},
                //这里 2|3|4|5 是模式匹配，  if 后面不是模式匹配，
                // 这里相当于 (2|3|4|5) if z =>... 而不是  2|3|4| (5 if z)
            _ => println!("no matched y")
        }
```

课程回顾

Rust基础

流程控制

```
let x = Some(5);
let y = 10;

match x {
    Some(50) => println!("Got 50"),
    Some(y) => println!("Matched, y = {:?}", y),
    _ => println!("Default case, x = {:?}", x),
}

println!("at the end: x = {:?}, y = {:?}", x, y);
```

命名变量：
irrefutable patterns
不可反驳模式，即 一定匹配得上

```
let x = 5;
let y=0;
match x {
    y  => println!(  “alway match here, can  ‘t go to other arm：{}”  ,y),
        // 这里会有一个编译告警，无法触达下面的分支，这里100%匹配，
        // 这里的y遮盖了外部的y，这句话是个模式匹配，
        // 捕获到的变量放到y里，而不是说要匹配 x=0

    _ => println!("anything")
}
```

Rust基础

流程控制

```
enum MatchMessage {
    Hello {id : i32}
}

///Match 的 @ at 符号绑定变量
fn match_rang_at() {

        let msg = MatchMessage::Hello { id: 90 }; // 把id 改为 4，20，90 看不同的结果

        //新建变量id_variable，在模式匹配中新建变量
        match msg {
            MatchMessage::Hello { id: id_variable @ 1..=10 } => { println!("match:{}", id_variable) },
            // 这里能访问匹配到的值, @ 可以保存匹配到的值到一个变量中

            MatchMessage::Hello { id: 11..=50 } => { println!("Found an id in another range") },
            // 这里无法拿到匹配到的值，因为id 可能是 10~50里的任意一个

            MatchMessage::Hello { id } => { println!("Found some other id: {}", id) }
            // 因为这里没有指定范围，可以拿到id的值
        }
}
```

https://doc.rust-lang.org/book/ch18-03-pattern-syntax.html

作业点评

使用Rust标准库实现一个 tcp server，可以参考网上的代码修改，最终需要各位同学上传代码和运行结果截图

 # 要求

1）能正常运行

2）对 tcp client（比如可用telnet等）发过来的消息，打印，并做echo返回

3）对代码每一句做注释

4）做一次标准的错误处理（模式匹配）

作业点评

```rust
pub fn main() -> std::io::Result<()> {
    //在本地8080端口创建TCP连接，将字符串解析为一个 SocketAddr 类型，如果解析失败，抛出异常
    let addr = "127.0.0.1:8080".parse::<SocketAddr>().unwrap();
    let listener = TcpListener::bind(addr)?;

    //不断的接收连接进来，等同于循环的 listener.accept()
    for stream in listener.incoming() {
        //模式匹配，incoming得到的是一个Result,有可能是Ok，也有可能是Err。
        match stream {
            Ok(stream) =>{
                handle_client(stream)
                    .unwrap_or_else(|error| eprintln!("Server error got: {:?}", error));
                    //如果遇到错误，会打印错误，错误由 stream的read，write 发出
            },
            Err(e) => eprintln!("connection failed!") //连接错误
        }
    }
    Ok(())
}
```

仅作参考

作业点评

```rust
fn handle_client( mut stream: TcpStream ) -> Result<()>{
    //创建一个缓冲区，用来保存数据
    let mut buf = [0; 500];
    //循环读取数据，如果读取数据异常直接返回
    while match stream.read( &mut buf ) {
            Ok(n) if n==0 => {
                    //如果读取到的长度是0，表示关闭连接，函数返回
                    false
            },
            Ok(n) => {
                    println!("received:{}", String::from_utf8_lossy(&buf[..n]));
                    //将读取到的内容返写到客户端去
                    stream.write( &buf[..n] )?;
                    true
            },
            Err(e) => {
                    false  //遇到错误，返回
            },
    }{ }
    Ok(()) //没有异常，返回 void
}
```

仅作参考

作业点评

优秀作业

https://github.com/simonjiao/learning-rust/tree/main/sahw2
https://github.com/taokeqin/rust-simple-tcp-echo-server/blob/main/src/main.rs
https://github.com/pencoa/substrate_hw3/blob/main/src/main.rs
https://github.com/TracySalander/RustEdu/tree/main/SubstrateBasic/Substrate_a3
https://github.com/Meta-One/MaLTCP
https://github.com/whtoo/Echo_Demo_RUST/blob/main/src/main.rs
https://github.com/lisiur/oneblock/blob/master/rust101/src/main.rs

扩展阅读

Substrate2.0宏定义

2.0		>=3.0
`decl_module! { }`	定义模块	`#[pallet::config]`
`module 内函数为 call`	包含可调用函数	`#[pallet:call]`
`decl_storage! { }`	定义存储单元	`#[pallet::storage]`
`decl_event! { }`	定义事件	`#[pallet::event]`
`decl_error! { }`	定义错误	`#[pallet::error]`

https://github.com/substrate-developer-hub/substrate-node-template/blob/v2.0.0-alpha.3/pallets/template/src/lib.rs

扩展阅读

Substrate2.0宏定义

```
decl_storage! {
  trait Store for Module<T: Trait> as TemplateModule {
          //.....
        Proofs get(fn proofs): map hasher(blake2_128_concat)   Vec<u8> => (T::AccountId, T::BlockNumber);
          //.....
        Something get(fn something): Option<u32>;
    }
}
```

```
#[pallet::storage]
#[pallet::getter(fn proofs)]
pub type Proofs<T:Config> = StorageMap<_, Blake2_128Concat, Vec<u8>, (T::AccountId, T::BlockNumber)>;

#[pallet::storage]
#[pallet::getter(fn something)]
pub type Something<T> = StorageValue<_, u32>;
```

https://zhuanlan.zhihu.com/p/99362959

扩展阅读

Substrate2.0宏定义

```
decl_module! {
  pub struct Module<T: Trait> for enum Call where origin: T::Origin {
    type Error = Error<T>;

    fn deposit_event() = default;

    ///……
    #[weight = 10_000 + T::DbWeight::get().writes(1)]
    pub fn do_something(origin, something: u32) -> dispatch::DispatchResult {
          let who = ensure_signed(origin)?;

          // Update storage.
          Something::put(something);

          // Emit an event.
          Self::deposit_event(RawEvent::SomethingStored(something, who));
          // Return a successful DispatchResult
          Ok(())
    }
```

3.0以后：

```
#[pallet::call]
impl<T: Config> Pallet<T> {



  #[pallet::weight(10_000 + T::DbWeight:
  pub fn do_something(origin: OriginFor
    let who = ensure_signed(origin)?;

    // Update storage.
    <Something<T>>::put(something);

    // Emit an event.
    Self::deposit_event(Event::Something
    // Return a successful DispatchResult
    Ok(())
  }
```

扩展阅读

Substrate2.0宏定义

```
decl_event! (
    pub enum Event<T> where AccountId = <T as frame_system::Trait>::AccountId {
        /// Event documentation should end with an array that provides descriptive names for event
        /// parameters. [something, who]
        SomethingStored(u32, AccountId),
    }
);
```

```
#[pallet::event]
#[pallet::generate_deposit(pub(super) fn deposit_event)]
pub enum Event<T: Config> {
    /// Event documentation should end with an array that provides descriptive names for event
    /// parameters. [something, who]
    SomethingStored(u32, T::AccountId),
}
```

扩展阅读

Substrate2.0宏定义

```
decl_error! {
    pub enum Error for Module<T: Trait> {
        /// Error names should be descriptive.
        NoneValue,
        /// Errors should have helpful documentation associated with them.
        StorageOverflow,
    }
}
```

```
#[pallet::error]
pub enum Error<T> {
    /// Error names should be descriptive.
    NoneValue,
    /// Errors should have helpful documentation associated with them.
    StorageOverflow,
}
```

扩展阅读

Toml语法定义

Cargo.toml 文件中引入依赖 dependencies 有 2种等价写法

```
[dependencies.compiler_builtins]
version = "0.1.2"
optional = true

[dependencies.core]
version = "1.0.0"
optional = true
package = "rustc-std-workspace-core"
```

https://github.com/substrate-developer-hub/substrate-node-template/blob/v2.0.0-alpha.3/pallets/template/Cargo.toml

```
[dependencies]
compiler_builtins = {version = "0.1.2", optional = true }
core = {version = "1.0.0", optional = true, package = "rustc-std-workspace-
core"}
```

扩展阅读

Toml语法定义

键值对 key-value pair：

```
name = "Orange"
physical.color = "orange"
physical.shape = "round"
site."google.com" = true
```

点分隔键允许将相近属性放在一起：

等价于 JSON 的如下结构：

```
{
  "name": "Orange",
  "physical": {
      "color": "orange",
      "shape": "round"
   },
   "site": {
      "google.com": true
   }
}
```

扩展阅读

Toml语法定义

OneBlock+

```
# 数组
integers = [ 1, 2, 3 ]
colors = [ "红", "黄", "绿" ]
# 嵌套数组
nested_array_of_ints = [ [ 1, 2 ], [3, 4, 5] ]
# 嵌套混合数组
nested_mixed_array = [ [ 1, 2 ], ["a", "b", "c"] ]
```

扩展阅读

Toml语法定义

```
# 表 (table)
[table-1]
key1 = "some string"
key2 = 123

[dog."tater.man"]
type.name = "pug"
```

表（也被称为哈希表或字典）是键值对的集合。
它们由表头定义，连同方括号作为单独的行出现。
看得出表头不同于数组，因为数组只有值。

在它下方，直至下一个表头或文件结束，都是这个表的键值对。
表不保证保持键值对的指定顺序。

等价于 JSON 的如下结构：`{ “dog”: { “tater.man”: { “type”: { “name”: "pug" } } } }`

点分隔键为最后一个键名前的每个键名创建并定义一个表：

```
fruit.apple.color = "red"
# 定义一个名为 fruit 的表
# 定义一个名为 fruit.apple 的表

fruit.apple.taste.sweet = true
# 定义一个名为 fruit.apple.taste 的表
# fruit 和 fruit.apple 已经创建过了
```

扩展阅读

Toml语法定义

```toml
# 内联表（Inline Table）

name = { first = "Tom", last = "Preston-Werner" }
point = { x = 1, y = 2 }
animal = { type.name = "pug" }
```

内联表被完整地定义在花括号之中：`{` 和 `}` 。

括号中，可以出现零或更多个以逗号分隔的键值对。

等同于下面的标准表定义：

```toml
[name]
first = "Tom"
last = "Preston-Werner"

[point]
x = 1
y = 2

[animal]
type.name = "pug"
```

扩展阅读

Toml语法定义

```toml
# 表数组 (Array of Tables)

[[products]]
name = "Hammer"
sku = 738594937

[[products]] # 数组里的空表

[[products]]
name = "Nail"
sku = 284758393

color = "gray"
```

表名写在双方括号里的表头来表示

表头的第一例定义了这个数组及其首个表元素，而后续的每个则在该数组中创建并定义一个新的表元素。

等价于 JSON 的如下结构：

```json
{
    "products": [
      { "name": "Hammer", "sku": 738594937 },
      { },
      { "name": "Nail", "sku": 284758393, "color": "gray" }
    ]
}
```

参考： https://toml.io/cn/v1.0.0

Rust 参考资料

参考：
https://www.bilibili.com/video/BV1hp4y1k7SV
https://fasterthanli.me/articles/a-half-hour-to-learn-rust
https://time.geekbang.org/column/intro/100085301
https://gist.github.com/jimmychu0807/9a89355e642afad0d2aeda52e6ad2424
https://github.com/studyrs/Rustt
https://github.com/rust-lang/rustlings
https://github.com/studyrs/rusty-book

One Block+

Substrate_

Incubate **Developers** | Build **Communities** | Create **Impact** | Shape the **Ecosystem**