学会 Substrate 区块链应用开发

9. Off-chain Worker 教程 II

Jimmy Chu
jimmy.chu@parity.io

获取帮助: **https://substrate.dev**

内容

- ocw: 发送 HTTP 请求及 JSON 解析
- ocw: 存储数据库
- Substrate im-online pallet 导读
- Substrate session pallet 导读

事前准备

- 继续参看 recipe offchain-demo pallet: https://github.com/substrate-developer-hub/recipes/tree/master/pallets/offchain-demo
- 可一边开着代码，一边听以下内容

发送 HTTP 请求及 JSON 解析

- ocw 的特质是它不需要所有节点都去跑同一堆逻辑，或不需要去同步，另外也不影响出块时间
- 可以用来作 http 访问及解析返回的 JSON 对象

代码

- 在 offchain-demo/src/lib.rs 裡

- `response.body().collect::<Vec<u8>>()` 最后一句返回的，就是以 bytes 格式呈现的 JSON 对象 (因为 no_std 没有 String 用)

```rust
fn fetch_from_remote() -> Result<Vec<u8>, Error<T>> {
  // 定義遠端 http 訪問地址
  let remote_url_bytes =
b"https://api.github.com/orgs/substrate-developer-hub".to_vec();
  let remote_url = str::from_utf8(&remote_url_bytes)
    .map_err(|_| <Error<T>>::HttpFetchingError)?;

  // 定義 timeout 時間
  let timeout =
sp_io::offchain::timestamp().add(rt_offchain::Duration::from_millis(
3000));

  // 創建請求
  let request = rt_offchain::http::Request::get(remote_url);

  // 發送請求
  let pending = request
    .deadline(timeout) // Setting the timeout time
    .send() // Sending the request out by the host
    .map_err(|_| <Error<T>>::HttpFetchingError)?;

  // http 請求是異步操作，所以我們在這裡 try_wait 等它回來
  let response = pending
    .try_wait(timeout)
    .map_err(|_| <Error<T>>::HttpFetchingError)?
    .map_err(|_| <Error<T>>::HttpFetchingError)?;

  // 撿查返回碼是否 ok (http code == 200?)
  if response.code != 200 {
    debug::error!("Unexpected http request status code: {}",
response.code);
    return Err(<Error<T>>::HttpFetchingError);
  }

  // 以 byte (u8) 型式返回內容
  Ok(response.body().collect::<Vec<u8>>())
}
```

ocw: JSON 序列化 / 反序列化

- 在 rust 里，JSON 解析一般用到 serde 及 serde_json 库

- 但因为在编释 runtime 已用了 serde 的 std 版。所以再用 serde 時得改用另一名字 (alt_serde)。而且我们对 serde_json 作了小修改

Cargo.toml

```
[dependencies]

alt_serde = { version = "1",
default-features = false, features =
["derive"] }

# updated to `alt_serde_json` when latest
version supporting feature `alloc` is
released
serde_json = { version = "1",
default-features = false, git =
"https://github.com/Xanewok/json", branch =
"no-std", features = ["alloc"] }
```

ocw: JSON 序列化

```
use alt_serde::{Deserialize, Deserializer};

#[serde(crate = "alt_serde")]
#[derive(Deserialize, Encode, Decode, Default)]
struct GithubInfo {
  // Specify our own deserializing function to
convert JSON string to vector of bytes
  #[serde(deserialize_with = "de_string_to_bytes")]
  login: Vec<u8>,
  #[serde(deserialize_with = "de_string_to_bytes")]
  blog: Vec<u8>,
  public_repos: u32,
}

pub fn de_string_to_bytes<'de, D>(de: D) ->
Result<Vec<u8>, D::Error>
where
  D: Deserializer<'de>,
{
  let s: &str = Deserialize::deserialize(de)?;
  Ok(s.as_bytes().to_vec())
}

// …
let serialized: Vec<u8> =
serde_json::to_vec(&gh_info)?;
```

parity 一块+ OneBlock+

ocw: JSON 反序列化

```
let gh_info: GithubInfo =

serde_json::from_str(&response_json).map_err(|
_| <Error<T>>::HttpFetchingError)?;
```

ocw: 存储数据库

特质:

- 专门给 ocw 读写的数据库
- 不会在区块链网络里同步
- 可在不同区块触发的 ocw 之间作沟通

代碼

引入相关的库

```
use sp_runtime::{
  offchain as rt_offchain,
  offchain::storage::StorageValueRef,
  ...
}
```

使用

```
// 用前，先創建一個 storage，傳入它的 key 作參數
// storage 的名稱最好以 pallet 名為前綴。因為 storage
key 是全局的。
let s_info =
StorageValueRef::persistent(b"offchain-demo::gh-inf
o");

if let Some(Some(gh_info)) =
s_info.get::<GithubInfo>() {
  debug::info!("cached gh-info: {:?}", gh_info);
  return Ok(());
}

// We are implementing a mutex lock here with
`s_lock`
let res: Result<Result<bool, bool>, Error<T>> =
s_lock.mutate(|s: Option<Option<bool>>| {
  ...
});
```

parity 一块+ OneBlock+

StorageValueRef 主要 API

与链上存储 StorageValue 类似

- pub fn set(&self, value: &impl Encode)
- pub fn get<T: Decode>(&self) -> Option<Option<T>>
- pub fn clear(&mut self)
- pub fn mutate<T, E, F>(&self, f: F) -> Result<Result<T, T>, E> where T: Codec, F: FnOnce(Option<Option<T>>) -> Result<T, E>

Substrate im-online Pallet

- 代码:https://github.com/paritytech/substrate/blob/master/frame/im-online/src/lib.rs
- 目的:
  在每个 session 内，验证人节点向其他节点发放脉冲信号证明自己在线。因为当验证人节点不能达到一定程度的在 线百分率，会受到惩罚，所持代币会被削减。
- 資源:
  https://substrate.dev/rustdocs/v2.0.0-rc3/pallet_im_online/index.html

- `fn heartbeat` - 外部可调用函数，加插一次脉冲信号到 ReceivedHeartbeats 存储
- `fn offchain_worker` - 捡查只有是 validators 才发出脉搏信号
- `fn send_heartbeats` - 发信号给所有 local_authority_keys 的节点(即验证节点)
- `fn prepare_heartbeat` - 准备数据并对数据签名
- `fn send_single_heartbeat` - 以不具签名交易 (但对数据包签名) 发出脉冲信号
- `fn validate_unsigned` - 签证这交易和附签名的数据包

Substrate Session Pallet

代码:https://github.com/paritytech/substrate/blob/master/frame/session/src/lib.rs

目的:
一个 session 内，验证人节点会维持不变。这模块就是决定 session 是否结束了，轮替 session 时读取新的验证人节点，并算出再下一轮的新验证人节点。里面也包含了管理 session 钥匙的工具性函数

参考:
https://substrate.dev/rustdocs/v2.0.0-rc3/pallet_session/index.html

- `fn set_keys, purge_keys, do_set_keys, do_purge_keys`, 用来管理 keys 的外部调用函数. purge_keys 把里面的所有 keys 移除。
- `fn load_keys, take_keys, put_keys, [put/clear]_key_owner` - 都是工具性函数。
- `fn on_initialize()` - 入口:檢测一个 session 是不是结束了，是的话就 rotate_session(), 然后返回这次的交易权重。
- `fn rotate-session` - 结束上一个 session，从 QueuedChanged 取得下一轮 session keys. 然后算出再下一轮的 session keys 放在 QueueChanged 内。新 session 开始

Questions?

官网文档：substrate.dev

知乎专栏：parity.link/zhihu

jimmy.chu@parity.io