第七课

陈锡亮 Bryan
Polkadot Ambassador, Co-founder @ Laminar & Acala

bryan@laminar.one

内容

- 模块间解耦与功能复用
- 完成 Substrate Kitty
- Kitties UI 简介
- FRAME 治理相关模块介绍
  - sudo / democracy
  - membership / collective / elections-phragmen
  - treasury

模块间解耦与功能复用

- 软件开发中比较常见的最佳实践
  - 避免不必要的耦合
    - 保持代码可维护性，可读性，可测试性
    - 增加代码复用性
  - 避免重复造轮子
    - 避免代码臃肿，避免重复bug
    - 增加开发效率

模块间解耦与功能复用

- 依赖注入 Dependency Injection
  - 依赖是由模块的使用者传入，而非模块自己写死
  - 保证模块之间的抽象化和重用性
  - 去除模块之间不必要的耦合
  - 更加方便的测试
  - 依赖于接口 (trait) 而不是实现 (pallet)

模块间解耦与功能复用

- 使用 trait 定义接口

```rust
pub trait Time {
    type Moment: AtLeast32Bit + Parameter + Default + Copy;

    fn now() -> Self::Moment;
}

/// Trait to deal with unix time.
pub trait UnixTime {
    /// Return duration since `SystemTime::UNIX_EPOCH`.
    fn now() -> core::time::Duration;
}
```

```rust
/// Abstraction over a fungible assets system.
pub trait Currency<AccountId> {
    /// The balance of an account.
    type Balance: AtLeast32Bit + FullCodec + Copy + MaybeSerializeDeserialize + Debug + Default;

    /// The opaque token type for an imbalance. This is returned by unbalanced operations
    /// and must be dealt with. It may be dropped but cannot be cloned.
    type PositiveImbalance: Imbalance<Self::Balance, Opposite=Self::NegativeImbalance>;

    /// The opaque token type for an imbalance. This is returned by unbalanced operations
    /// and must be dealt with. It may be dropped but cannot be cloned.
    type NegativeImbalance: Imbalance<Self::Balance, Opposite=Self::PositiveImbalance>;
```

模块间解耦与功能复用

- 定义依赖

```
pub trait Trait: frame_system::Trait + pallet_transaction_payment::Trait {
    type Time: Time;
    type Randomness: Randomness<Self::Hash>;
```

- 注入依赖

```
impl pallet_contracts::Trait for Runtime {
    type Time = Timestamp;
    type Randomness = RandomnessCollectiveFlip;
```

模块间解耦与功能复用

- 使用依赖

```
fn random(&self, subject: &[u8]) -> SeedOf<T> {
    T::Randomness::random(subject)
}
```

```
pub fn top_level(origin: T::AccountId, cfg: &'a Config<T>, vm: &'a V, loader: &'a L) -> Self {
    ExecutionContext {
        caller: None,
        self_trie_id: None,
        self_account: origin,
        overlay: OverlayAccountDb::<T>::new(&DirectAccountDb),
        depth: 0,
        deferred: Vec::new(),
        config: &cfg,
        vm: &vm,
        loader: &loader,
        timestamp: T::Time::now(),
        block_number: <frame_system::Module<T>>::block_number(),
    }
}
```

FRAME 治理模块

- 权限控制
  - sudo
  - democracy
- 成员管理
  - membership
  - collective
  - elections-phragmen
- 财政管理
  - treasury

作业

1. 补完剩下的代码
   https://github.com/SubstrateCourse/substrate-kitties/blob/lesson7/pallets/kitties/src/linked_item.rs
2. 修复单元测试
3. 阅读 pallet-membership
   a. 分析 add_member 的计算复杂度
   b. 分析 pallet-membership 是否适合以下场景下使用，提供原因
      i. 储存预言机提供者
      ii. 储存游戏链中每个工会的成员
      iii. 储存 PoA 网络验证人