Substrate

River

Test-003

Testing Genesis

https://substrate.dev/substrate-collectables-workshop/#/5/testing-genesis

Testing Genesis初始化

Substrate允许您通过genesis块配置使用预配置的存储来部署链。

步骤如下：

- 扩展decl_storage以添加额外的genesis数据
- 在测试期间模拟genesis配置
- 测试genesis设定

1- Add Extra Genesis - **add_extra_genesis**

在decl_storage范围内，使用以下配置值创建一个名为add_extra_genesis的结构。

```
decl_storage! {
    trait Store for Module<T: Trait> as KittyStorage { ... }

    add_extra_genesis {
        config(kitties): Vec<(T::AccountId, T::Hash, T::Balance)>;
    }
}
```

add_extra_genesis

简单地说，add_extra_genesis允许您向genesis配置中添加新字段。这些新字段随后可用于构建其他存储或修改现有存储。

新字段在config()属性中命名。在这种情况下，我们创建了一个额外的小猫genesis字段。这个kitties字段需要一个包含以下值的元组向量:

- AccountId: the kitty owner
- Hash: the kitty id/dna
- Balance: the kitty's initial value

2- initialize the storage items: 初始化存储列表

接下来，我们需要使用现有的mint函数从这个kitties配置值初始化存储项，以构建存储值。

在add_extra_genesis中，您可以使用一个特殊的构建闭包来执行以下逻辑:

```
build(|storage: &mut StorageOverlay, _: &mut
ChildrenStorageOverlay, config: &GenesisConfig<T>| {
    with_storage(storage, || {
        for &(ref acct, hash, balance) in &config.kitties {

            let k = Kitty {
                    id: hash,
                    dna: hash,
                    price: balance,
                    gen: 0
                };

            let _ = <Module<T>>::mint(acct.clone(), hash, k);
        }
    });
});
```

3- Mock Genesis for Tests

回想一下，您使用TextExternalities为测试创建了初始模拟，其中仅使用初始链状态的默认值。

在同一个函数中，您现在可以指定初始小猫配置。为简单起见，让我们初始化2只小猫：

1 kitty with random DNA, belonging to account #0, worth 50 balance.
2 kitty with blank DNA, belonging to account #1, worth 100 balance.

```
fn build_ext() -> TestExternalities<Blake2Hasher> {
    let mut t =
system::GenesisConfig::<KittiesTest>::default().build_storage().unwrap().0;

t.extend(balances::GenesisConfig::<KittiesTest>::default().build_storage().unw
rap().0);
    t.extend(GenesisConfig::<KittiesTest> {

        // Your genesis kitties
        kitties: vec![    (0, H256::random(), 50), (1, H256::zero(), 100)],

    }.build_storage().unwrap().0);
    t.into()
}
```

Test Genesis

```
#[test]
fn should_build_genesis_kitties() {
    with_externalities(&mut build_ext(), || {
        // Check that 2nd kitty exists at genesis, with value 100
        let kitty0 = Kitties::kitty_by_index(0);
        let kitty1 = Kitties::kitty_by_index(1);

        // Check we have 2 kitties, as specified
        assert_eq!(Kitties::all_kitties_count(), 2);

        // Check that they are owned correctly
        assert_eq!(Kitties::owner_of(kitty0), Some(0));
        assert_eq!(Kitties::owner_of(kitty1), Some(1));

        // Check owners own the correct amount of kitties
        assert_eq!(Kitties::owned_kitty_count(0), 1);
        assert_eq!(Kitties::owned_kitty_count(2), 0);
    })
}
```

Your Turn!

Set up your genesis specs as specified above.

Write some new tests to ensure that kitties are correctly configured at genesis.

Refactor your previous tests to take advantage of this setup.

Genesis Deployment

https://github.com/paritytech/polkadot/blob/d102d8fbac950abf2a696097d65ec2edc64dc216/service/src/chain_spec.rs

Substrate

River

Thanks