Substrate

River

Test-001

Setting Up Tests

https://substrate.dev/substrate-collectables-workshop/#/5/setting-up-tests

测试配置

创建测试模块

- 1 测试模块放在当前文件中，例如：substratekitties.rs，
- 2 创建测试文件test.rs。

substratekitties.rs

```
// Your substratekitties code

#[cfg(test)]
mod tests {
    // Your tests
}
```

属性#[cfg(test)]声明整个测试模块只是测试代码。

依赖

接下来，我们从外部模块导入一些测试依赖项。这些模块中的大部分用于替换我们想要在测试中实现的trait的配置类型。

```
use super::*;
use support::{impl_outer_origin, assert_ok};
use runtime_io::{with_externalities, TestExternalities};
use primitives::{H256, Blake2Hasher};
use runtime_primitives::{
    BuildStorage, traits::{BlakeTwo256, IdentityLookup},
    testing::{Digest, DigestItem, Header}
};
```

runtime_io

1- TestExternalities

内存中基于hashmap的外部性实现。换句话说，它模拟了运行时以最小方式执行所需的测试存储。TestExternalities接受通用类型Hasher，因此我们还导入Blake2Hasher，以便稍后在构建TestExternalities时使用它

2- with_externalities

1- 外部性类型的对象
2- 在给定第一个参数的情况下执行的闭包

```
with_externalities(some_externality, || {
    some_assertions!()
})
```

Origin

我们还要为测试运行时构造一个Origin类型。这个步骤通常由construct_runtime宏自动调用。但是在测试期间，我们必须手动操作。

```
impl_outer_origin! {
    pub enum Origin for KittiesTest {}
}
```

Construct a Mock Runtime: 构造一个模拟运行时

声明了一个主配置类型KittiesTest。KittiesTest将实现Kitties运行时使用的模块的每个配置特征，如system和balances。

```
#[derive(Clone, Eq, PartialEq)]
pub struct KittiesTest;

// Implement the system module traits
impl system::Trait for KittiesTest {}

// Implement the balances module traits
impl balances::Trait for KittiesTest {}

// Implement the trait for our own module, `super::Trait`
impl super::Trait for KittiesTest {}
```

system::Trait

```
impl system::Trait for KittiesTest {
    type Origin = Origin;
    type Index = u64;
    type BlockNumber = u64;
    type Hash = H256;
    type Hashing = BlakeTwo256;
    type Digest = Digest;
    type AccountId = u64;
    type Lookup = IdentityLookup<Self::AccountId>;
    type Header = Header;
    type Event = ();
    type Log = DigestItem;
}
```

Note：在test mock中，我们可以简化传递到某些trait的值类型。

1- AccountId可以用u64类型表示
2- Event可以用()来实现

Create Test Externalities: 创建测试外部性

我们已经准备好访问和构建我们刚刚实现的trait的模块。

让我们为Kitties模块分配一个类型别名，以便以后方便地访问它的方法。

```
type Kitties = super::Module<KittiesTest>;
```

我们使用一个包装器函数来创建前面提到的TestExternalities。

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

这个build_ext包装器函数随后将用于构建每个单元测试的模拟。在大多数情况下，它只是根据我们想要的模型构建一个genesis存储键/值存储。

You turn!

Substrate

River

Thanks