The Move Book

In programming languages, reflection is the ability of a program to examine and modify its own structure and behavior. Move has a limited form of reflection that allows you to inspect the type of a value at runtime. This is useful when you need to store type information in a homogeneous collection, or when you need to check if a type belongs to a package.

Type reflection is implemented in the Standard Library module std::type_name . Expressed very roughly, it gives a single function get() which returns the name of the type T .

The module is straightforward, and operations allowed on the result are limited to getting a string representation and extracting the module and address of the type.

Type reflection is an important part of the language, and it is a crucial part of some of the more advanced patterns.

In practice

The module is straightforward, and operations allowed on the result are limited to getting a string representation and extracting the module and address of the type.

```bash module book::type_reflection;

use std::ascii::String; use std::type_name::{Self, TypeName};

/// A function that returns the name of the type `T` and its module and address. public fun do_i_know_you(): (String, String, String) { let type_name: TypeName = type_name::get();

```
// there's a way to borrow
let str: &String = type_name.borrow_string();

let module_name: String = type_name.get_module();
let address_str: String = type_name.get_address();

// and a way to consume the value
let str = type_name.into_string();

(str, module_name, address_str)
```

}

# [test_only]

public struct MyType {}

# [test]

fun test_type_reflection() { let (type_name, module_name, _address_str) = do_i_know_you();

```
//
assert!(module_name == b"type_reflection".to_ascii_string());
```

} ```

Type reflection is an important part of the language, and it is a crucial part of some of the more advanced patterns.

Further reading

Type reflection is an important part of the language, and it is a crucial part of some of the more advanced patterns.
```