

Table of Contents

1	Moonbit/Core Result
1.1	Overview
1.2	Usage
1.2.1	Constructing Result
1.2.2	Querying variant
1.2.3	Extracting values
1.2.4	Transforming values
1.2.5	Monadic operations

Moonbit/Core Result

Overview

`Result[T,E]` is a type used for handling computation results and errors in an explicit and declarative manner, similar to Rust (`Result<T,E>`) and OCaml (`(('a, 'e) result)`). It is an enum with two variants: `Ok(T)`, which represents success and contains a value of type `T`, and `Err(E)`, representing error and containing an error value of type `E`.

Usage

Constructing Result

You can create a `Result` value using the `Ok` and `Err` constructors, remember to give proper type annotations.

```
1
2  test {
3      let _result : Result[Int, String] = Ok(42)
4      let _error : Result[Int, String] = Err("Error message")
5
6  }
```

Or use the `ok` and `err` functions to create a `Result` value.

```
1
2  test {
3      let _result : Result[String, Unit] = Ok("yes")
4      let _error : Result[Int, String] = Err("error")
5
6  }
```

Querying variant

You can check the variant of a `Result` using the `is_ok` and `is_err` methods.

```
1
2  test {
3      let result : Result[Int, String] = Ok(42)
4      let is_ok = result is Ok(_)
5      assert_eq(is_ok, true)
6      let is_err = result is Err(_)
7      assert_eq(is_err, false)
8  }
```

Extracting values

You can extract the value from a `Result` using the `match` expression (Pattern Matching).

```
1
2  test {
3      let result : Result[Int, Unit] = Ok(33)
4      let val = match result {
5          Ok(value) => value
6          Err(_) => -1
7      }
8      assert_eq(val, 33)
9  }
```

Or using the `unwrap` method, which will panic if the result is `Err` and return the value if it is `Ok`.

```
1
2  test {
3      let result : Result[Int, String] = Ok(42)
4      let value = result unwrap()
5      assert_eq(value, 42)
6  }
```

A safe alternative is the `or` method, which returns the value if the result is `Ok` or a default value if it is `Err`.

```
1
2  test {
3      let result : Result[Int, String] = Err("error")
4      let value = result or(0)
5      assert_eq(value, 0)
6  }
```

There is a lazy version of `or` called `or_else`, which takes a function that returns a default value.

```
1
2  test {
3      let result : Result[Int, String] = Err("error")
4      let value = result or_else(() => 0)
5      assert_eq(value, 0)
6  }
```

Transforming values

To transform values inside a `Result`, you can use the `map` method, which applies a function to the value if the result is `Ok`, and remains unchanged if it is `Err`.

```
1
2  test {
3      let result : Result[Int, String] = Ok(42)
4      let new_result = result map(x => x + 1)
5      assert_eq(new_result, Ok(43))
6  }
```

A dual method to `map` is `map_err`, which applies a function to the error value if the result is `Err`, and remains unchanged if it is `Ok`.

```
1
2  test {
3      let result : Result[Int, String] = Err("error")
4      let new_result = result map_err(x => x + "!")
5      assert_eq(new_result, Err("error!"))
6  }
```

You can turn a `Result[T, E]` into a `Option[T]` by using the method `to_option`, which returns `Some(value)` if the result is `Ok`, and `None` if it is `Err`.

```

1
2  test {
3    let result : Result[Int, String] = Ok(42)
4    let option = result to_option()
5    assert_eq(option, Some(42))
6    let result1 : Result[Int, String] = Err("error")
7    let option1 = result1 to_option()
8    assert_eq(option1, None)
9  }

```

Monadic operations

Moonbit provides monadic operations for `Result`, such as `flatten` and `bind`, which allow chaining of computations that return `Result`.

```

1
2  test {
3    let result : Result[Result[Int, String], String] = Ok(Ok(42))
4    let flattened = result flatten()
5    assert_eq(flattened, Ok(42))
6  }

```

The `bind` method is similar to `map`, but the function passed to it should return a `Result` value.

```

1
2  test {
3    let result : Result[Int, String] = Ok(42)
4    let new_result = result bind(x => Ok(x + 1))
5    assert_eq(new_result, Ok(43))
6  }

```