



# ICP区块链开发入门课程

4. 用 Motoko 做后端

主讲: Paul Liu - DFINITY 工程师

# 课程大纲

1. 使用 SDK 搭建一个简易网站
2. Motoko 语言简介
3. Canister 智能合约
4. 用 Motoko 做后端
5. 用 Javascript 做前端



# 值、类型、类型推断、类型检查

- 值域 vs. 类型域
- 类型代表了静态语义
- 类型检查让代码更安全
- 类型标注可以帮助类型推断

```
import Blob "mo:base/Blob";  
type Blob = Blob.Blob;
```

duplicate definition for Blob in block Motoko

mo:base/Blob

[View Problem](#) No quick fixes available

```
let Blob = "Blob";
```

```
var seed : [var Nat8] = [var 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
```

# 基础类型

- 布尔型 Bool
- 自然数 Nat, Nat8, Nat16, Nat32, Nat64
- 整数 Int, Int8, Int16, Int32, Int64
- 浮点数 Float
- 字符串 Text
- 字符 Char
- Principal
- Blob
- None
- Error

# Record (记录结构) vs. Variant (枚举)

```
let person = {  
  name = "Jacky Chan";  
  age = 67;  
};
```

```
func f() : {name: Text; age: Nat} {  
  person  
}
```

元组 (tuple) 是记录结构 (record) 的特殊形式

```
let x : (Int, Bool) = (10, false);  
let y : Bool = x.1;
```

```
type Gender = {  
  #male;  
  #female;  
};
```

```
let person = {  
  name = "Jacky Chan";  
  age = 67;  
  gender = #male;  
};
```

```
func f() : {name: Text; age: Nat} {  
  person  
};
```

field gender does not exist in type  
{age : Nat; name : Text} Motoko

[View Problem](#) No quick fixes available

```
let g = f().gender;
```

# 模式匹配 (Pattern match)

```
type Person = {  
  name: Text;  
  age: Nat;  
  gender: Gender;  
};
```

```
func retired(person: Person) : Bool {  
  switch (person.gender) {  
    case (#male) (person.age >= 60);  
    case (#female) (person.age >= 55);  
  }  
};
```

```
type Gender = {  
  #male;  
  #female;  
  #unspecified: {retire_age: Nat};  
};
```

```
func retired(person: Person) : Bool {  
  switch (person.gender) {  
    case (#male) (person.age >= 60);  
    case (#female) (person.age >= 55);  
    case (#unspecified({retire_age})) (person.age >= retire_age);  
  }  
};
```



# Option 和 Result

```
func retired(person: Person) : ?Bool {  
  switch (person.gender) {  
    case (#male) ?(person.age >= 60);  
    case (#female) ?(person.age >= 55);  
    case (#unspecified) null;  
  }  
};
```

- Option 类型: ?Bool, ?Nat, ...
- Option 值: null, ?true, ?12, ...
- Result 类型: Result<R, E>
- Result 值: #ok(true), #err("Unknown")

```
type Result<Ok, Err> = {  
  #ok : Ok;  
  #err : Err;  
};  
  
//import Result "mo:base/Result";  
//type Result<R, E> = Result.Result<R, E>;  
  
func retired(person: Person) : Result<Bool, Text> {  
  switch (person.gender) {  
    case (#male) #ok(person.age >= 60);  
    case (#female) #ok(person.age >= 55);  
    case (#unspecified) #err("Unknown");  
  }  
};
```

# 子类型关系 (subtype)

- $B \leq A$
- B 是 A 的子类型
- 所有接受 A 类型的地方都可以用 B 类型的值
- A 更宽泛 (general), B 更具体 (specific)

```
func retired(person: { age: Nat; gender: Gender }) : Bool {  
  switch (person.gender) {  
    case (#male) (person.age >= 60);  
    case (#female) (person.age >= 55);  
  }  
};
```

```
let jacky : Person = {  
  name = "Jacky Chan";  
  age = 67;  
  gender = #male;  
};
```

```
let _ = retired(jacky);
```

```
func dec(a: Int) : Int { a - 1 };  
func inc(a: Nat) : Nat { a + 1 };
```

```
let _ = dec(1 : Nat);
```

literal of type  
Int  
does not have expected type  
Nat Motoko

[View Problem](#) No quick fixes available

```
let _ = inc(-1);
```



# 函数

- 函数: 从定义域 (Domain) 到值域 (Range) 的映射关系
- 类型:  $() \rightarrow \text{Result}<\text{Bool}, \text{Text}>$ ,  $() \rightarrow ()$ , ...
- 函数定义

```
func dec(a: Int) : Int { a - 1 };  
func inc(a: Nat) : Nat { a + 1 };
```

- 匿名函数

```
let dec : Int -> Int = func (a) { a - 1 };  
let inc = func (a: Nat) : Nat { a + 1 };
```

# 高阶函数

From “mo:base/Array”:

```
/// Initialize a mutable array with `size` copies of the initial value.
public func init<A>(size : Nat, initVal : A) : [var A] {
  | Prim.Array_init<A>(size, initVal);
};

/// Initialize an immutable array of the given size, and use the `gen` function to produce the initial value for every index.
public func tabulate<A>(size : Nat, gen : Nat -> A) : [A] {
  | Prim.Array_tabulate<A>(size, gen);
};

// arr = [var 42, 42, 42, 42, 42] : [var Int]
let arr = Array.init<Int>(5, 42);

// brr = [0, 1, 2, ..., 99] : [Nat]
let brr = Array.tabulate<Nat>(100, func (i) { i });

// crr = [0, 2, 4, ..., 198] : [Int]
let crr = Array.tabulate<Int>(100, func (i) { i * 2 });
```

# Object (对象)

```
object counter {  
  var count = 0;  
  public func inc() { count += 1 };  
  public func read() : Nat { count };  
  public func bump() : Nat {  
    inc();  
    read()  
  };  
};  
  
let counter : Counter = do {  
  var count = 0;  
  let inc = func () { count += 1; };  
  let read = func () : Nat { count };  
  {  
    inc = inc;  
    read = read;  
    bump = func () : Nat { inc(); read() };  
  }  
};
```

```
type Counter = {  
  inc: () -> ();  
  read: () -> Nat;  
  bump: () -> Nat;  
};  
  
let counter : Counter = object {  
  var count = 0;  
  public func inc() { count += 1 };  
  public func read() : Nat { count };  
  public func bump() : Nat {  
    inc();  
    read()  
  };  
};
```

# Object 和 Class

```
class Counter() {  
  var c = 0;  
  public func inc() : Nat {  
    c += 1;  
    return c;  
  }  
};
```

```
let c1 = Counter();  
let c2 = Counter();  
let x = c1.inc();
```

```
class Counter(init : Nat) {  
  var c = init;  
  public func inc() : Nat { c += 1; c };  
};
```

```
let c1 = Counter(0);  
let c2 = Counter(10);
```

```
import Buffer "mo:base/Buffer";
```

```
class Counter<X>(init : Buffer.Buffer<X>) {  
  var buffer = init.clone();  
  public func add(x : X) : Nat { buffer.add(x); buffer.size() };  
  public func reset() { buffer := init.clone() };  
};
```

```
let c1 = Counter(Buffer.Buffer<Int>(10));  
let c2 = Counter<Nat>(Buffer.Buffer(20));
```

```
public class Buffer<X>(initCapacity : Nat) {  
  ...  
}
```

```
public class HashMap<K, V>(  
  initCapacity : Nat,  
  keyEq : (K, K) -> Bool,  
  keyHash : K -> Hash.Hash) {  
  ...  
}
```

# Actor

```
actor Counter {  
    var count = 0;  
  
    public shared func inc() : async () { count += 1 };  
  
    public shared func read() : async Nat { count };  
  
    public shared func bump() : async Nat {  
        count += 1;  
        count;  
    };  
};
```

```
type Counter = actor {  
    inc : shared () -> async ();  
    read : shared () -> async Nat;  
    bump : shared () -> async Nat;  
};
```

```
actor Counter {  
    var count = 0;  
  
    public shared func inc() : async () { count += 1 };  
  
    public shared query func read() : async Nat { count };  
  
    public shared func bump() : async Nat {  
        await inc();  
        await read();  
    };  
};
```

```
type Counter = actor {  
    inc : shared () -> async ();  
    read : shared query () -> async Nat;  
    bump : shared () -> async Nat;  
};
```

# 实例 - Microblog

```
public type Message = Text;  
  
public type Microblog = actor {  
  follow: shared(Principal) -> async (); // 添加关注对象  
  follows: shared query () -> async [Principal]; // 返回关注列表  
  post: shared (Text) -> async (); // 发布新消息  
  posts : shared query () -> async [Message]; // 返回所有发布的消息  
  timeline : shared () -> async [Message]; // 返回所有关注对象发布的消息  
};
```

一个(极简的)去中心化的社交网络应用

- 每个 canister 代表一个用户
- Canister 可以通过 canister id 相互关注

# 通过 caller id 进行权限管理

每一个消息(远程函数调用)都有一个唯一确定的发送方 (caller)

- 由用户发出的消息
- Canister 相互之间发送的消息

可以在代码中直接获取 caller 的身份 (principal id)

```
public shared (msg) func post(text: Text): async () {  
    assert(Principal.toText(msg.caller) == "...");  
    ...  
};
```



# Actor Class

```
import Nat "mo:base/Nat";
import Map "mo:base/RBTree";

actor class Bucket(n : Nat, i : Nat) {

  type Key = Nat;
  type Value = Text;

  let map = Map.RBTree<Key, Value>(Nat.compare);

  public func get(k : Key) : async ?Value {
    assert((k % n) == i);
    map.get(k);
  };

  public func put(k : Key, v : Value) : async () {
    assert((k % n) == i);
    map.put(k,v);
  };
};
```

```
actor Map {

  let n = 8; // number of buckets
  type Key = Nat;
  type Value = Text;
  type Bucket = Buckets.Bucket;
  let buckets : [var ?Bucket] = Array.init(n, null);

  public func get(k : Key) : async ?Value {
    switch (buckets[k % n]) {
      case null null;
      case (?bucket) await bucket.get(k);
    };
  };

  public func put(k : Key, v : Value) : async () {
    let i = k % n;
    let bucket = switch (buckets[i]) {
      case null {
        let b = await Buckets.Bucket(n, i);
        buckets[i] := ?b;
        b;
      };
      case (?bucket) bucket;
    };
    await bucket.put(k, v);
  };
};
```

# Module import

- 库模块

```
import Array "mo:base/Array";  
import Result "mo:base/Result";
```

- 本地模块

```
import Types "types";  
import Utils "utils";
```

- Actor Class

```
import Counters "Counters";  
actor CountToTen {  
  let C : Counters.Counter = await Counters.Counter(1);  
  ...  
};
```

- Canister

```
import BigMap "canister:BigMap";  
import Connectd "canister:connectd";
```

# 课程作业 1

判断下述子类型关系是否为真

1.  $\{a: \text{Bool}\} \leq \{a: \text{Bool}; b: \text{Nat}\}$
2.  $\{a: \text{Bool}\} \leq \{\}$
3.  $\{\#red; \#blue\} \leq$   
 $\{\#red; \#yellow; \#blue\}$
4.  $\text{Nat} \leq \text{Int}$
5.  $\text{Int} \leq \text{Int32}$
6.  $() \rightarrow () \leq (\text{Text}) \rightarrow ()$
7.  $() \rightarrow (\text{Text}) \leq () \rightarrow ()$
8.  $() \rightarrow (\{\#male; \#female\}) \leq () \rightarrow ()$
9.  $(\text{Int}) \rightarrow (\text{Nat}) \leq (\text{Nat}) \rightarrow (\text{Int})$
10.  $(\text{Int16}, \text{Nat8}) \leq (\text{Int32}, \text{Nat32})$

# 课程作业

1. 把 Message 类型改为一个记录结构，并在里面添加 time 字段，记录发消息的时间。
2. 修改 posts 和 timeline 方法，仅返回指定时间之后的内容：

```
import Time "mo:base/Time";  
func posts(since: Time.Time): async [Message] {...};  
func timeline(since: Time.Time): async [Message] {...};
```

3. 思考题：如果关注对象很多，运行 timeline 就会比较慢，有什么办法可以提高效率？

## 下一节: Javascript 前端实例

- Agent-js 代理库
- 网络资料管理
- 异步调用后端方法
- 错误和异常处理
- 类型转换、编码与解码