第一课: Rust入门基本原理(2)

所有权机制、借用规则

苏林

今天公开课内容

1、所有权机制
2、借用规则
3、生命周期及参数(大家看之前的公开课)

Copy Trait -> 区分 Copy语义和Move语义

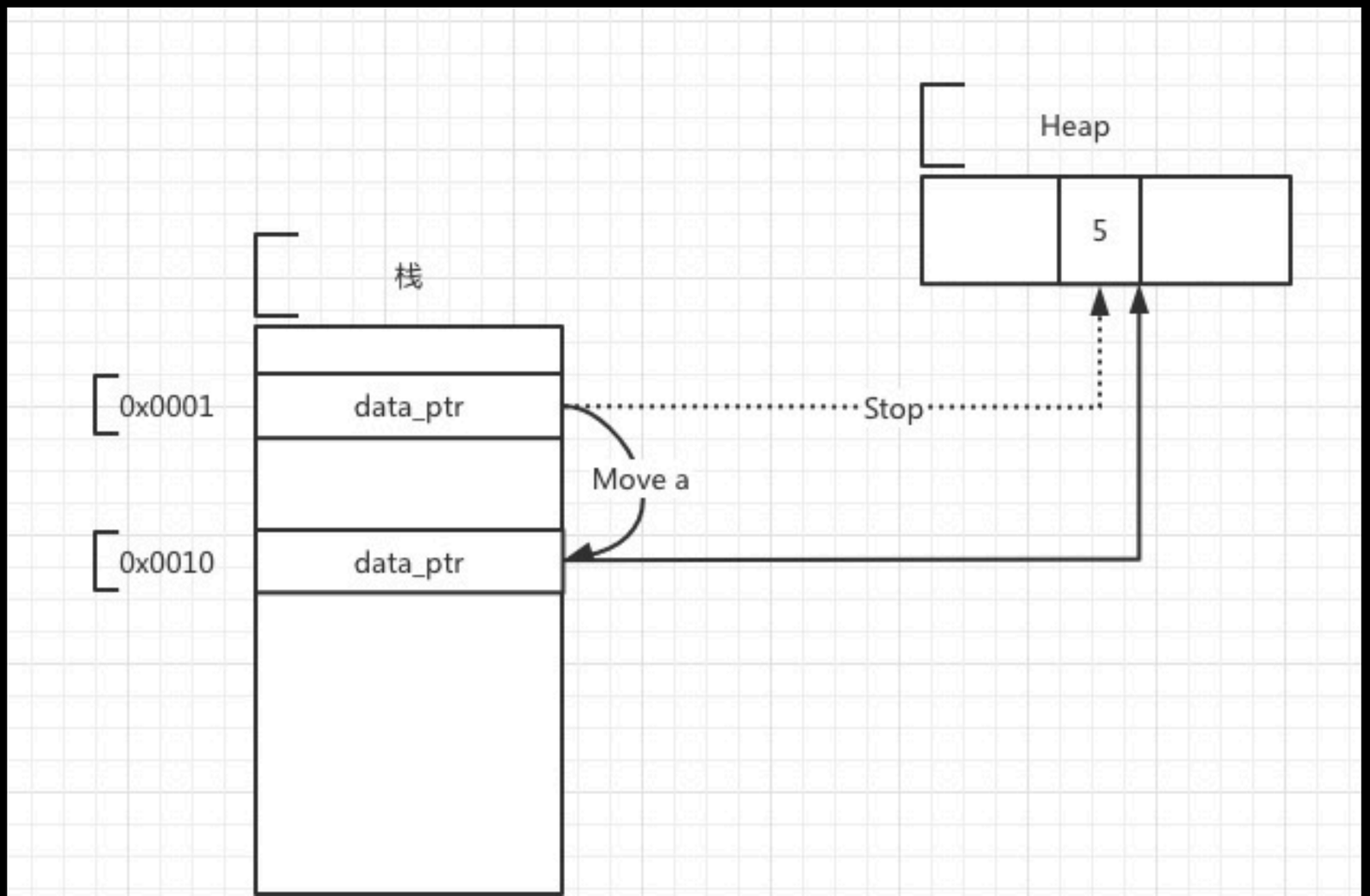
Copy语义 => 按位复制

Copy语义对应值类型, Move语义对应引用类型.

所有权机制: 保证内存安全和性能

所有权转移. 每个值都有一个所有者.

```
fn main() {
    let a = Box::new(5);
    let b = a;
    println!("{:?}", a);
}
```



```rust
#[derive(Debug)]
struct A {
    a: i32,
    b: i32
}


fn main() {
    let a = A {a: 1, b:2};
    let b :A = a;
    println!("{:?}", a);
}
```

```rust
#[derive(Debug)]
struct A {
    a: i32,
    b: Box<i32>
}

fn main() {
    let a = A {a: 1, b: Box::new( x: 2)};
    let b : A  = a;
    println!("{:?}", a);
}
```

```rust
fn main() {
    let a : (String, String) = (String::from( s: "a"), String::from( s: "b"));
    let b : (String, String) = a;
    println!("{:?}", a);

    let c : (...) = (1, 2, 3);
    let d : (...) = c;
    println!("{:?}", c);
}
```

```rust
fn main() {
    let v: [i32; 3] = [1, 2, 3];
    foo(v);
    assert_eq!([1,2,3], v);
}

fn foo(mut v: [i32; 3]) -> [i32; 3] {
    v[0] = 3;
    assert_eq!([3,2,3], v);
    v
}
```

```rust
fn main() {
    let mut v : [i32; 3] = [1, 2, 3];
    foo(&mut v);
    assert_eq!([3,2,3], v);
}

fn foo(v: &mut [i32; 3]) {
    v[0] = 3;
}
```

借用规则 => 保证内存安全

1、借用的生命周期不能长于出借方. -> 防止出现悬垂指针.

2、可变借用(&mut a) 不能有别名. -> 独占，可变借用不能共享, 只能独占, 只能有一个.

3、不可变借用不能再次出借为可变借用. -> 共享不可变

```rust
fn main() {
    let i :i32 = 20;
    let mut o :i32 = 5;
    compute( input: &i, output: &mut o);
}

fn compute(input: &i32, output: &mut i32) {
    if *input > 20 {
        *output = 1;
    }

    if *input > 5 {
        *output *= 2;
    }
}
```

QA环节

加群一起交流Rust & Databend

Databend

Rust语言中文社区

知数堂