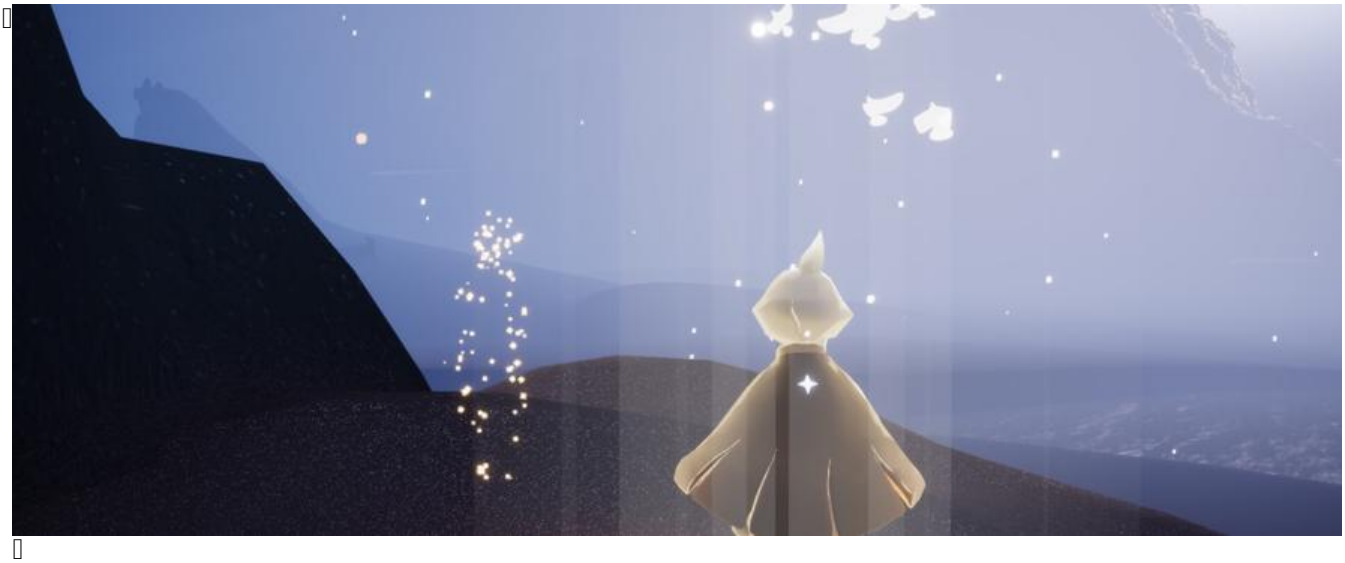


智能指针

与普通指针的对比

date 引用

智能指针

实现

常见的智能指针

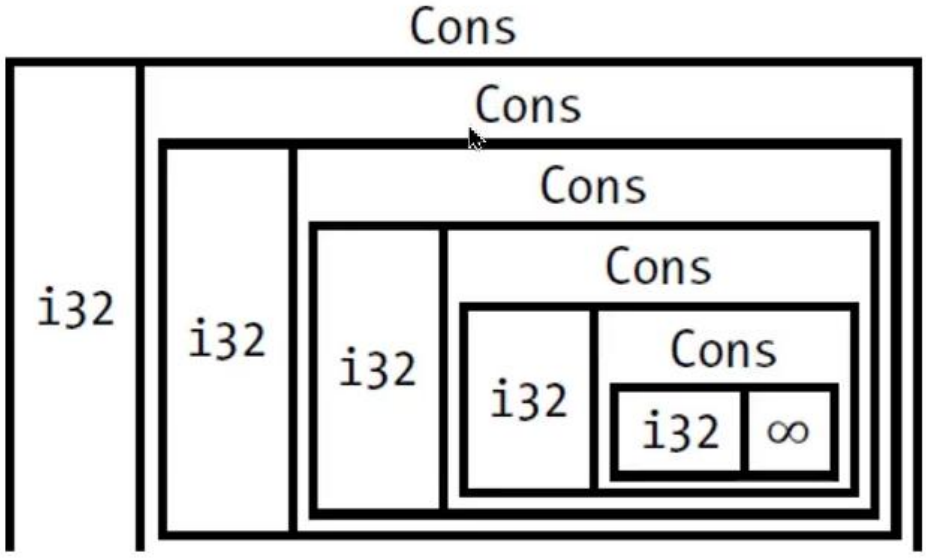
在堆内存上分配值

启用多重所有权的引用计数类型

通过RefCell<T>访
在运行时而不是编译时强制借用规则的类型

Box<T> 以及堆内存

使用场景



Deref Trait

!

!

! " "

隐式解引用

解引用与可变性

- 在类型和 trait 在下列三种情况发生时，Rust 会执行 deref coercion：
  - 当 T: Deref<Target=U>，允许 &T 转换为 &U
  - 当 T: DerefMut<Target=U>，允许 &mut T 转换为 &mut U
  - 当 T: Deref<Target=U>，允许 &mut T 转换为 &U

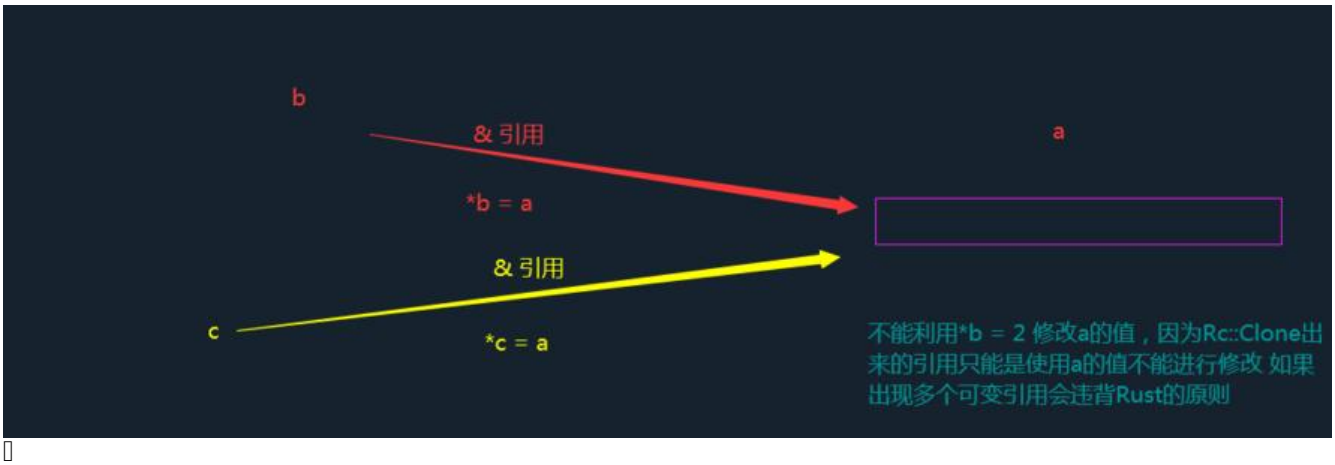
Drop Trait

RC

```
fn main() {
    let a = Rc::new(Cons(5, Rc::new(Cons(10, Rc::new(Nil)))));
    println!("count after creating a = {}", Rc::strong_count(&a));
                                                              1
    let b = Cons(3, Rc::clone(&a));
    println!("count after creating b = {}", Rc::strong_count(&a));
                                                              2
    {
        let c = Cons(4, Rc::clone(&a));
        println!("count after creating c = {}", Rc::strong_count(&a)
    }                                                         3
    println!("count after c goes out of scope = {}",
        Rc::strong_count(&a));
                                2
}
```

Rc::clone() vs 类型的 clone() 方法

- Rc::clone() ：增加引用，不会执行数据的深度拷贝操作
- 类型的 clone()：很多会执行数据的深度拷贝操作。



RefCell 和 内部可变性

不可变引用 代码执行错误

! " "

! " "

可变引用 代码执行正确

! " "

- 两个方法（安全接口）：
  - borrow 方法
    - 返回智能指针 Ref<T>，它实现了 Deref
  - borrow_mut 方法
    - 返回智能指针 RefMut<T>，它实现了 Deref

附录：循环引用导致内存泄漏