

Rust ownership

1. 内存自动释放机制：在作用域结束的时候，自动释放

```
1 fn main() {
2     let a = String::from("hello");
3     foo(a);
4     let b = a; // error: 在将a传入foo之后, 在foo结束的时候内存被释放
5 }
6
7 fn foo(s: String) {
8     // todo
9 }
9
```

2. 转移 (move)：相当于两个变量共享同一个内存（浅复制），转移之后原变量不能继续用

```
1 fn main() {
2     let a = String::from("hello");
3     let b = a;
4     println!("{}", a); // error, a被转移
5 }
6
7 fn main() {
8     let a = String::from("hello");
9     let b = a.clone(); // deep clone
10    println!("{}", a); // a依然可用, 用clone方法深复制
11 }
12
13 fn main() {
14     let a = 1;
15     let b = a;
16     println!("{}", a); // a 是栈空间里的变量, 不会被转移
17 }
```

不会发生转移的类型（发生copy）

- All the integer types, such as `u32`.
- The Boolean type, `bool`, with values `true` and `false`.
- All the floating point types, such as `f64`.
- The character type, `char`.
- Tuples, if they only contain types that are also `Copy`. For example, `(i32, i32)` is `Copy`, but `(i32, String)` is not.

3. 不可变借用

```
1 // example 1
2 fn main() {
3     let a = String::from("hello");
4     foo(&a);
5     let b = a; // 借用之后可以进行转移
6 }
7
8 fn foo(s: &String) {
9     // todo
10 }
11
12 // example 2
13 fn main() {
14     // 允许有多个借用者
15     let a = String::from("hello");
16     foo(&a); // success
17     koo(&a); // success
18 }
19 fn foo(s: &String) {}
20 fn koo(s: &String) {}
21
22 // example 3
23 fn main() {
24     let a = String::from("hello");
25     foo(&a);
26 }
```

```
27 fn foo(s: &String) {
28     s.push_str(" world"); // error: immutable borrow
29 }
```

4. 可变借用

```
1 fn main() {
2     let mut a = String::from("abc");
3     foo(&mut a); // 相当于c++的引用，a的值已经被改变
4     println!("{}", a);
5 }
6
7 fn foo(s: &mut String) {
8     s.push_str("def");
9 }
10
11 fn main() {
12     let mut a = String::from("hello");
13     foo(&mut a);
14     let b = a; // 可变借用用在借用之后可以转移
15     println!("{}", b);
16 }
17
18 fn foo(s: &mut String) {
19     s.push_str(" world");
20 }
21
22 fn main() {
23     let mut a = String::from("hello");
24     foo(&mut a); // 多个借用者
25     koo(&mut a);
26 }
27
28 fn foo(s: &mut String) {
29     println!("{}", s);
30 }
```

```
31
32 fn koo(s: &mut String) {
33     println!("{}", s);
34 }
35
36 fn main() {
37     let mut s = String::from("hello");
38
39     let r1 = &mut s; // error: 在同一个scope里面, 最多允许存在一个可变借用者
40     let r2 = &mut s;
41
42     println!("{}", r1, r2);
43 }
44
45 let mut s = String::from("hello");
46
47 let r1 = &s; // no problem
48 let r2 = &s; // no problem, 不可变借用可以有多个, 但是在存在不可变借用的时候, 不允许在同一个
    scope有可变借用
49 let r3 = &mut s; // BIG PROBLEM
50 println!("{}", r1, r2, r3);
51
52 // 例外的情况
53 let mut s = String::from("hello");
54
55 let r1 = &s; // no problem
56 let r2 = &s; // no problem
57 println!("{}", r1, r2);
58 // r1 and r2 are no longer used after this point
59 let r3 = &mut s; // no problem
60 println!("{}", r3); // r1和r2在第一个println之后没有再使用, 这时候r3是可以安全的改变数据的, 因此没有问题
```