References & Borrowing

- 之前我们提到了，如果每次给函数传一个参都需要转移一次ownership，然后再转回来的话非常复杂。这里，rust给出的方案是`reference`。
    - reference: refer to the value without taking the ownership of the value.

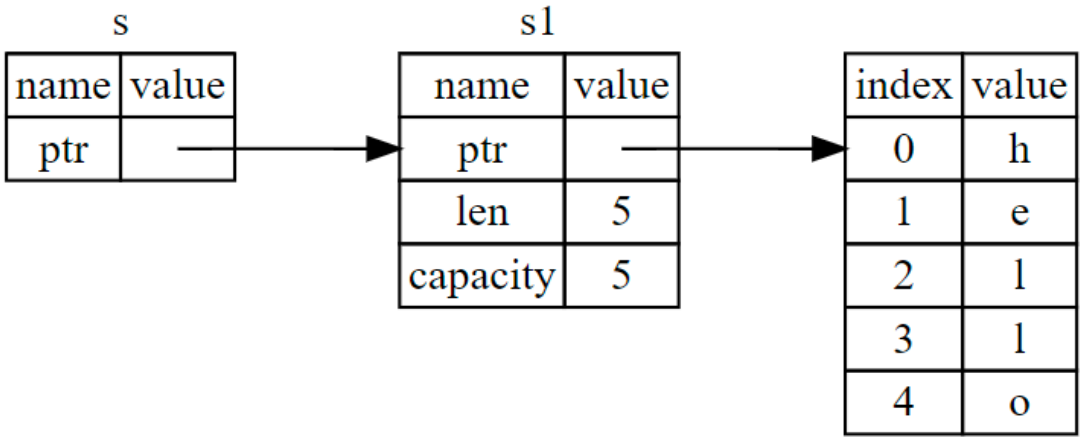
```
fn main() {
    let s1 = String::from("hello");

    let len = calculate_length(&s1);

    println!("The length of {} is {}", s1, len);
}

fn calculate_length(s: &String) -> usize{
    s.len()
}
```

    - `&s1` lets us create a reference that refers to the value of s1 but not own it.
    - 对于参数s, 它只是再函数内valid，但在out of scope的时候并不会被drop()，因为它并没有那个值的ownership。
    - 我们把创造一个reference的过程叫做`borrowing`： 其实贼形象，你借用了这个值，但并不拥有这个值。它的实际效果是：



- reference的值是不能改变的，比如：

```
fn main() {
    let mut s1 = String::from("hello");

    change(&s1);
}

fn change(some_string: &String){
    some_string.push_str(", world");
}
```

    它会报错：

```
ubuntu:~/Desktop/Rust_Learning/4_Ownership/reference$ cargo run
   Compiling reference v0.1.0
(/home/ubuntu/Desktop/Rust_Learning/4_Ownership/reference)
warning: variable does not need to be mutable
 --> src/main.rs:2:9
  |
2 |     let mut s1 = String::from("hello");
  |         ----^^
  |         |
  |         help: remove this `mut`
  |
  = note: `#[warn(unused_mut)]` on by default

error[E0596]: cannot borrow `*some_string` as mutable, as it is behind a `&`
reference
  --> src/main.rs:16:5
   |
15 | fn change(some_string: &String){
   |                        ------- help: consider changing this to be a
mutable reference: `&mut String`
16 |     some_string.push_str(", world");
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ `some_string` is a `&` reference,
so the data it refers to cannot be borrowed as mutable
```

- 当然，我们也可以让reference mutable。方法就是将pointer本身变成mut，并且让reference也用`&mut`：

```
fn main() {
    let mut s1 = String::from("hello");
    change(&mut s1);
}

fn change(some_string: &mut String){
    some_string.push_str(", world");
}
```

  - 注意，上述例子三个地方都要加`mut`：
    - 声明pointer时
    - 传参时
    - 声明函数的参数类型时

多个References同时存在的情况

- mutable的reference有一个很大的限制：同时只能有一个mutable reference：我认为是出于避免data race的情况，比如下面的代码就会报错：

```
fn main() {
    let mut s1 = String::from("hello");

    let r1 = &mut s1;
    let r2 = &mut s1;
    println!("{}, {}", r1, r2);
}
```

报错：

```
error[E0499]: cannot borrow `s1` as mutable more than once at a time
 --> src/main.rs:5:14
  |
4 |     let r1 = &mut s1;
  |              ------- first mutable borrow occurs here
5 |     let r2 = &mut s1;
  |              ^^^^^^^ second mutable borrow occurs here
6 |     println!("{}, {}", r1, r2);
  |                        -- first borrow later used here

For more information about this error, try `rustc --explain E0499`.
error: could not compile `reference` due to previous error
```

- 通常，我们的解决方法是，给它加个scope，限定它的生命周期：

```
{
    let r1 = &mut s1;
}

let r2 = &mut s1;
```

- 同样，rust也不允许immutable和mutable同时出现：

```
fn main() {
    let mut s1 = String::from("hello");

    let r1 = &s1;
    let r2 = &s1;
    let r3 = &mut s1;
    println!("{}, {}, {}", r1, r2, r3);

}
```

报错：

```
error[E0502]: cannot borrow `s1` as mutable because it is also borrowed as
immutable
 --> src/main.rs:6:14
  |
4 |     let r1 = &s1;
  |              --- immutable borrow occurs here
5 |     let r2 = &s1;
6 |     let r3 = &mut s1;
  |              ^^^^^^^ mutable borrow occurs here
7 |     println!("{}, {}, {}", r1, r2, r3);
  |                            -- immutable borrow later used here

For more information about this error, try `rustc --explain E0502`.
error: could not compile `reference` due to previous error
```

- Rust只允许同时多个immutable reference存在（因为只有多个"读"不会data race）

悬空reference

- Rust是禁止dangling reference的。为达到禁止dangling reference, Rust对这方面的限制是：确保data 不会比reference先out of scope。
- 比如，下面的代码就会报错：

```
fn main() {
    let reference_to_nothing = dangle();
}

fn dangle() -> &String{
    let s = String::from("hello");
    &s
}
```

报错：

```
error[E0106]: missing lifetime specifier
  --> src/main.rs:26:16
   |
26 | fn dangle() -> &String{
   |                ^ expected named lifetime parameter
   |
   = help: this function's return type contains a borrowed value, but there
is no value for it to be borrowed from
help: consider using the `'static` lifetime
   |
26 | fn dangle() -> &'static String{
   |                ~~~~~~~~

For more information about this error, try `rustc --explain E0106`.
error: could not compile `reference` due to previous error
```

  - 因为value比reference先out of scope, 所以报错了

- 我们可以总结以下两个关于References的规则:
  - At any time, you can have either one mutable reference or any number of mmutable references.
  - References must always be valid.