Hash Map

- Hash Map是以键值对的形式保存信息的，其中keys必须是同一种数据类型，values也必须是同一种数据类型。这里，我们不用深究HashMap在Rust中的实现原理。
- 创建HashMap的几种方式：

```
use std::collections::HashMap;

fn main() {

    let mut scores = HashMap::new();

    scores.insert(String::from("Blue"), 10);
    scores.insert(String::from("Yellow"), 50);

    let teams = vec![String::from("Blue"), String::from("Yellow")];
    let initial_scores = vec![10, 50];

    let mut scores: HashMap<_, _> =
        teams.into_iter().zip(initial_scores.into_iter()).collect();
}
```

  - 第一种是直接申明一个空的，然后往里面insert
  - 第二种是将两个Vec的iteration zip到一起，然后再collect成HashMap的形式。
- 另外，不管是上面的第一种方法还是第二种方法，这样的操作都会拿走原先数据结构的ownership，比如下面的代码：

```
let teams = vec![String::from("Blue"), String::from("Yellow")];
let initial_scores = vec![10, 50];

let mut scores: HashMap<_, _> =
  teams.into_iter().zip(initial_scores.into_iter()).collect();

println!("{}", teams[0]);
```

  会报这样的错，因为Vec并没有实现copy trait

```
10  |     let teams = vec![String::from("Blue"), String::from("Yellow")];
    |         ----- move occurs because `teams` has type `Vec<String>`,
which does not implement the `Copy` trait
...
14  |         teams.into_iter().zip(initial_scores.into_iter()).collect();
    |               ----------- `teams` moved due to this method call
15  |
16  |     println!("{}", teams[0]);
    |                    ^^^^^ value borrowed here after move
```

  - 正确的修改方法是用reference：

```
let mut scores: HashMap<_, _> =
        (&teams).into_iter().zip(initial_scores.into_iter()).collect();
```

读取HashMap的值的方法

- 用get的方法，返回的是Option类型：

```
let team_name = String::from("Blue");
let score = scores.get(&team_name);

match score {
    Some(s) => println!("{}", &s),
    None => (),
}
```

- 也可以用`for in`的方法：

```
for (key, value) in &scores{
    println!("{} : {}", key, value);
}
```

如何改变HashMap中的值

- Overwrite：

```
let mut scores = HashMap::new();

scores.insert(String::from("Blue"), 10);
scores.insert(String::from("Blue"), 25);

println!("{:?}", scores);
```

    - 打出来的结果是`{"Blue": 25}`，证明10这个值已经被覆盖了
- 我们还能用if的方法来插入值！

```
scores.entry(String::from("Blue")).or_insert(50);
```

    - 如果HashMap中已经有Blue这个key的值了，那么就不会插入50。如果没有则会插入。
- 我们甚至可以update based on the old values!

```
let text = "hello world wonderful world";
let mut map = HashMap::new();

for word in text.split_whitespace(){
    let count = map.entry(word).or_insert(0);
    *count += 1;
}
println!("{:?}", map);
```

    - 这里，输出的结果会是：

```
{"world": 2, "hello": 1, "wonderful": 1}
```

- `or_insert()`会返回一个mutable reference（&mut v），然后我们用`*`dereference后就可以修改。