Struct

定义与实例化一个Struct

- 首先，很直观的定义一个struct：

```
struct User{
    active: bool,
    username: String,
    email: String,
    sign_in_count: u64
}
```

- 在实例化一个struct的时候，我们可以设置它为mutable的，但这个mutable是对于整个struct的，而不能仅针对其中一个元素

```
fn main() {
    let mut user1 = User{
        email: String::from("119010249@link.cuhk.edu.cn"),
        username: String::from("qpr"),
        active: true,
        sign_in_count: 1,
    };

    user1.email = String::from("1905873179@qq.com");
}
```

- 我们也可以用一个函数，来简化实例化的输入：

```
fn build_user(email: String, username: String) -> User{
    User{
        email,
        username,
        active: true,
        sign_in_count: 1,
    }
}
```

- 我们也可以用另一个struct的值来实例化一个struct：

```
let user2 = User{
    email: String::from("11111111@qq.com"),
    ..user1
};
```

    - 这里很神奇地做到了，user2有自己地email，但其他field的值完全和user1相同。
    - 但！非常重要的是，这里相当与是copy了user1的除了email之外的值。对于username这个属性，它是一个&String, 这意味着ownship发生了转换。所以这样赋值以后，user1就不能invalid了。

    - 除非，user2的email和username这两个属性都用的自己的值。active和sign_in_count这两个不用担心ownship转换,因为他们都是u64或bool

---

Tuple Struct

- Tuple Struct与Tuple不同的是：tuple是一个变量，而Tuple Struct是一个Struct，其是可以声明+实例化的。

```
struct Color(i32, i32, i32);
struct Point(i32, i32, i32);

let black = Color(0, 0, 0);
let origin = Point(0, 0, 0);
```

Unit-Like Structs

- 甚至可以定义并实例化没有任何fields的struct（可能之后会有奇用）：

```
struct AlwaysEqual;

let subject = AlwaysEqual;
```
