String

- 我们一般说Rust中的String的时候，是指`String`或者`&str`. 关于二者的区别可以看看这一篇：https://zhuanlan.zhihu.com/p/123278299
- 创建String的很多种方法：

```
let mut s = String::new();

let data = "init contents";
let s = data.to_string();

let s = String::from("init contents");
```

- 更新String的方法：

```
let mut s = String::from("init contents");
s += " Hi!";
println!("after adding first: {}", s);
```

- 我们也可以用push的方法：

```
let mut s1 = String::from("foo");
let s2 = "bar";
s1.push_str(s2);
println!("s2 is {}", s1);
```

  - 这里push_str( )用的是string slice，因为我们并不想拿走s2的ownership
- 将两个String合并到一起的方法：

```
let s1 = String::from("Hello, ");
let s2 = String::from("world!");
let s3 = s1 + &s2;
```

  - 注意的是，这里s1被拿走了ownership，所有接下来就不会valid了。但s2因为是引用，所以就没关系。
  - 但其实`+`只能将一个`&str`加到String后面，所以下面的会报错：

```
let s1 = String::from("Hello, ");
let s2 = String::from("world!");
let s3 = s1 + s2;
```

  - 报错信息：

```
error[E0308]: mismatched types
  --> src/main.rs:20:19
   |
20 |     let s3 = s1 + s2;
   |                   ^^
   |                   |
   |                   expected `&str`, found struct `String`
   |                   help: consider borrowing here: `&s2`
```

- 但有趣的是，我们在之前的`let s3 = s1 + &s2;`中`&s2`实际上是个`&String`，理论上是不符合的。这里之所以可以编译，是因为rust compiler用了deref coercion, 将`&s2`转成了`&s2[..]`
- 用`+`来concentrate strings的话，我们发现它一定会拿走s1的ownership。为了不让任何输入的string被取走ownership，我们可以用`format!`：

```
let s1 = String::from("Hello, ");
let s2 = String::from("world");
let s3 = String::from("!");

let s = format!("{}{}{}", s1, s2, s3);
```

Index into String

- 实际上，Rust是不允许index into string的，比如下面的代码会报错：

```
let s1 = String::from("hello");
let h = s1[0];
```

遍历String

- 直接上例子：

```
for c in "你好啊".chars(){
    println!("{}", c);
}

for b in "你好啊".bytes(){
    println!("{}", b);
}
```

输出：

```
你
好
啊
228
189
160
229
165
189
229
149
138
```

- 这里我们发现，rust实际上是用3个u8来存一个汉字的。这也解释了为什么rust不允许index string。因为比如如果index"你好啊"的第一位，难道返回1/3个“你”吗？