宏 macro

宏在 Rust 里指的是一组相关特性的集合称谓:

- 使用`macro_rules!` 构建的声明宏 (declarative macro)
- 3 种过程宏
  - 自定义`#[derive]`宏, 用于struct 或enum, 可以为其指定随 `derive` 属性添加的代码
  - 类似属性的宏, 在任何条目上添加自定义属性
  - 类似函数的宏, 看起来像函数调用, 对其指定为参数的 token 进行操作

1. 宏与函数的差别

- 宏是用来编写可以生成其它代码的代码, 即所谓的 **元编程**(metaprogramming)
- 函数在定义签名时, 必须声明参数的个数和类型,
- 宏可处理可变的参数
- 编译器会在解释代码前展开宏
- 宏的定义比函数复杂得多, 难以阅读,理解, 维护
- 在某个文件调用宏时, 必须提前定义宏或将宏引入当前作用哉
- 函数可以在任何位置定义,并在任何位置使用

2. `macro_rules!`声明宏

```
#[macro_export]
macro_rules! vec {
    ( $( $x: expr ),* ) => {
        {
            let mut temp_vec = Vec::new();
            $(
                temp_vec.push($x);
            )*
            temp_vec
        }
    }
}
// 此处有一个单边模式 ( $( $x: expr ),* ), 后跟 => 和模式相关的代码块. 如果模式匹配, 代码将被执行. 更复杂的宏会有多个单边模式.
// $x: expr 匹配任何rust的表达式,并命名为x
// $( $x: expr ),* 表示匹配*号前0个或1个表达式
```

3. 过程宏

这种形式更像函数

- 接收并操作输入的Rust代码
- 生成另外一些Rust代码结果

三种过程宏:

- 自定义派生
- 属性宏
- 函数宏

Note: 创建过程宏时, 宏定义必须单独放在它们自己的包中, 并使用特殊的包类型