从C++到Rust，包含多个crate的package

原创 Ajonbin AJonbin的杂货铺 2024年01月02日 21:15 上海

之前讲过的是一个package里只包含一个crate，今天来看看在同一个pakcage里怎么包含多个crate。

先来看一个新的概念， target

> Cargo packages consist of targets which correspond to source files which can be compiled into a crate.
>
> cargo package 可以包含不同的target，这些target所指定的源代码可以编译成crate。

我们可以简单的理解成一个target对应个一个crate。

cargo 定义了5种不同的target

- library -- 库
- binary -- 可执行二进制文件
- example -- 示例
- test -- 功能测试
- benchmark -- 性能测试

有一条特殊的规则：一个pakcage里只能有一个library target，其他类型的target不限数量。

由于Cargo最基本的功能是包管理器，包管理最主要的就是库管理。如果每个package里只能有一个lib，那么我们只要指定库的名字，cargo就知道要使用哪个库。

如果一个package里可以有多个库，那么就需要额外的信息来指定库的名字，并且还有可能涉及到库之间的依赖关系，这就会带来额外的复杂性，所以Cargo在设计时就规定一个package里只能有一个lib。

现在主要来看看binary和library。

第一步，先创建个新的package，hello_pkg

```
$ cargo new hello_pkg
     Created binary (application) `hello_pkg` package

$ cd hello_pkg/

hello_pkg$ tree
.
├── Cargo.toml
└── src
    └── main.rs

2 directories, 2 files
```


第二步，增加一个lib库，simplemathlib。

在这之前，再强势插入一个“广告”，module。

module用来在一个crate里更好的组织你的代码。module可以把相关的代码组织成一个逻辑单元，并可以控制接口的可见性。这样更好实现模块化和可复用性。

module可以通过mod关键字来定义，也可以通过文件和目录结构来自动匹配。下面会有相应的演示，来帮助理解module的用法。

先创建库的目录和文件。

```
hello_pkg$ mkdir src/simplemathlib
hello_pkg$ vim src/simplemathlib/add.rs
hello_pkg$ mkdir src/simplemathlib/minus
hello_pkg$ vim src/simplemathlib/minus/mod.rs
hello_pkg$ vim src/simplemathlib/lib.rs
```


再来看看源文件的内容。

```
hello_pkg$ cat src/simplemathlib/add.rs

pub fn add(left: usize, right: usize) -> usize {
    left + right
}

hello_pkg$ cat src/simplemathlib/minus/mod.rs

pub fn minus(left: i32, right: i32) -> i32 {
    left - right
}

hello_pkg$ cat src/simplemathlib/lib.rs

pub mod add;
pub mod minus;

pub fn multiple(left:u32, right:u32) -> u32{
    left*right
}
```


首先，我们创建了一个目录simplemathlib，这就是我们要新建的lib crate。

在simplemathlib中包含了3个文件

simplemathlib/add.rs -- 定一个了一个add函数

simplemathlib/minus/mod.rs -- 定义了一个minus函数，注意文件名是mod.rs

simplemathlib/lib.rs -- 定一个了一个multiple函数。

注意lib.rs里用mod关键字申明了两个module

mod add 对应于 add.rs

mod minus 对应于 minus/mod.rs

这就是我们之前提到了“*module可以通过文件和目录结构来自动匹配*”。

如果是文件，那么module 名字和文件名相同。

如果是目录，那么module名字和目录名字相同，但是此目录下必须有一个名字为mod.rs的源文件来定义的module的实现。

代码层面实现完之后，需要在Cargo.toml文件中通过[lib]来申明libary段，如下图

```
[package]
name = "hello_pkg"
version = "0.1.0"
edition = "2021"

[dependencies]

[lib]
name = "simplemathlib"
path = "src/simplemathlib/lib.rs"
```


这样库crate就创建完了，我们可以用cargo build --lib 来编译一下。

```
$ cargo build --lib

   Compiling hello_pkg v0.1.0 (/private/tmp/hello_pkg)
    Finished dev [unoptimized + debuginfo] target(s) in 0.31s
```


在编译时， cargo build默认时编译可执行文件，所以这里我们要加上--lib参数。之前提过一个package里只有一个lib，所以不必指定库的名字。

第三步，增加可执行文件

由于package里可以有多个可执行文件，这次，我们就创建两个binary。

```
hello_pkg$ cat src/main.rs

fn main() {
    println!("Hello, world!");
  let result = simplemathlib::add::add(1,2);
    println!("1+2={}",result);
  let another_result = simplemathlib::multiple(1,2);
    println!("1x2={}",another_result);
}

hello_pkg$ cat src/minus.rs
fn main() {
    println!("Hello, minus!");
  let result = simplemathlib::minus::minus(2,1);
    println!("2-1={}",result);
}
```


首先，我们修改main.rs，其中调用时add函数，由于add函数在add module里，调用时需要加上module名字。再调用multiple函数。

接着，我们添加一个文件minus.rs，这是第二个可执行文件，所有也有main函数。这里我们调用minus函数。

添加完代码，就需要在Cargo.toml里增加binary段。

```
[package]
name = "hello_pkg"
version = "0.1.0"
edition = "2021"

[dependencies]

[lib]
name = "simplemathlib"
path = "src/simplemathlib/lib.rs"

[[bin]]
name = "minus"
path = "src/minus.rs"
```


这里有两点要注意

第一，增加binary段要使用[[bin]]，这里有两个[]。而[lib]只有一个[]。是因为，binary可以有多个，[[]]可以解析成一个表结构。而lib段只能有一个，所有lib只需要一个[]。

第二，我们只增加一个 binary，minus.rs。另外一个main.rs是默认的binary，所以不需要再添加了。

最后，我们在运行一下这两个binary。

```
hello_pkg$ cargo run

error: `cargo run` could not determine which binary to run.
Use the `--bin` option to specify a binary, or the `default-run` manifest key.
available binaries: hello_pkg, minus
```


可以看到，由于我们现在有两个binary，cargo不知道该运行哪一个，于是就报错。

这时需要通过--bin 参数来指定要运行的可执行target

```
hello_pkg$ cargo run --bin hello_pkg

   Compiling hello_pkg v0.1.0 (/private/tmp/hello_pkg)
    Finished dev [unoptimized + debuginfo] target(s) in 0.96s
     Running `target/debug/hello_pkg`
Hello, world!
1+2=3
1x2=2

hello_pkg$ cargo run --bin minus

   Compiling hello_pkg v0.1.0 (/private/tmp/hello_pkg)
    Finished dev [unoptimized + debuginfo] target(s) in 0.17s
     Running `target/debug/minus`
Hello, minus!
2-1=1
```


你也可以在Cargo.toml中用default-run在指定默认运行的程序，这样就不用每次都指定了。

```
[package]
name = "hello_pkg"
version = "0.1.0"
edition = "2021"
default-run = "minus"
```


这样，minus就变成了默认的可执行程序，运行minus就不需要再加--bin参数了。

```
hello_pkg$ cargo run

    Finished dev [unoptimized + debuginfo] target(s) in 0.07s
     Running `target/debug/minus`
Hello, minus!
2-1=1
```


收工。