



# Rust 编程语言入门



Microsoft®  
Most Valuable  
Professional

杨旭，微软MVP  
Rust、Go、C#开发者

## 1.4 Hello, Cargo

# Cargo


- Cargo 是 Rust 的构建系统和包管理工具
  - 构建代码、下载依赖的库、构建这些库...
- 安装 Rust 的时候会安装 Cargo
  - `cargo --version`

# 使用 Cargo 创建项目

- 创建项目：`cargo new hello_cargo`
  - 项目名称也是 `hello_cargo`
  - 会创建一个新的目录 `hello_cargo`
    - `Cargo.toml`
    - `src` 目录
      - `main.rs`
  - 初始化了一个新的 Git 仓库，`.gitignore`
    - 可以使用其它的 VCS 或不使用 VCS：`cargo new` 的时候使用 `--vcs` 这个 flag

# Cargo.toml

- TOML (Tom's Obvious, Minimal Language) 格式, 是 Cargo 的配置格式
- `[package]`, 是一个区域标题, 表示下方内容是用来配置包 (package) 的
  - name, 项目名
  - version, 项目版本
  - authors, 项目作者
  - edition, 使用的 Rust 版本
- `[dependencies]`, 另一个区域的开始, 它会列出项目的依赖项。
- 在 Rust 里面, 代码的包称作 crate。

```
hello_cargo >  Cargo.toml
1  [package]
2  name = "hello_cargo"
3  version = "0.1.0"
4  authors = ["dave <slash@qq.com>"]
5  edition = "2018"
6
7  [dependencies]
```

## src/main.rs

- cargo 生成的 main.rs 在 src 目录下
- 而 Cargo.toml 在项目顶层下
- 源代码都应该在 src 目录下
- 顶层目录可以放置：README、许可信息、配置文件和其它与程序源码无关的文件
- 如果创建项目时没有使用 cargo，也可以把项目转化为使用 cargo：
  - 把源代码文件移动到 src 下
  - 创建 Cargo.toml 并填写相应的配置

# 构建 Cargo 项目

## cargo build

- cargo build
  - 创建可执行文件: `target/debug/hello_cargo` 或 `target\debug\hello_cargo.exe` (Windows)
  - 运行可执行文件: `./target/debug/hello_cargo` 或 `.\target\debug\hello_cargo.exe` (Windows)
- 第一次运行 cargo build 会在顶层目录生成 cargo.lock 文件
  - 该文件负责追踪项目依赖的精确版本
  - 不需要手动修改该文件

# 构建和运行 cargo 项目

## cargo run

- cargo run, 编译代码 + 执行结果
  - 如果之前编译成功过, 并且源码没有改变, 那么就会直接运行二进制文件



## cargo check

- cargo check, 检查代码, 确保能通过编译, 但是不产生任何可执行文件
- cargo check 要比 cargo build 快得多
  - 编写代码的时候可以连续反复的使用 cargo check 检查代码, 提高效率

# 为发布构建

- `cargo build --release`
  - 编译时会进行优化
    - 代码会运行的更快，但是编译时间更长
  - 会在 *target/release* 而不是 *target/debug* 生成可执行文件
- 两种配置：
  - 一个开发
  - 一个正式发布

尽量用 Cargo

再见

