







# rust

- rust
- 
- 
- 
- 
- 
- 

# rust



# windows:

[https://static.rust-lang.org/rustup/dist/x86\\_64-pc-windows-msvc/rustup-init.exe](https://static.rust-lang.org/rustup/dist/x86_64-pc-windows-msvc/rustup-init.exe)

msvcwindows sdk

# linux:

`curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`

# 

`source "$HOME/.cargo/env"`

**rustc --version **

# 

`rustc --version`

# 

`rustup show`

# 

`rustup update`

# 

`rustup doc`

# 

```
rustup self uninstall
```




[vscode, rust-analyzer](#) [rust-lang.org](#) 


# hello, world


```
# vi main.rs



fn main() {
    println!("Hello, world!");
}



# rustc 
rustc main.rs
```


# cargo


```
# 
cargo --version

# 
cargo new hello_cargo

# debug 
cargo build

# release 
cargo build --release

# 
cargo run

# 
```

cargo check



```
/
/Cargo.toml          # [ ][ ][ ][ ][ ][ ]

/src                 # [ ][ ][ ][ ]
|--/main.rs          # main
L--

/target              # [ ][ ][ ][ ][ ]
|--/debug             # debug[ ][ ]
L--/release           # release[ ][ ]
```

```
[package]
name = "project_name"
version = "0.1.0"
edition = "2023"

[dependencies]
```



```
# [ ][ ][ ][ ][ ][ ][ ]
let x = 5;

# [ ][ ][ ][ ][ ]
let mut x = 5;
x=10;

# [ ][ ][ ]
let x = 9;
```

```
fn function_name(){
}

// 测试
// 测试 ->测试
fn fun_name() -> u32 {
    // 测试测试
    // 55
    // 测试 return
    // return 55;
}
```

测试

```
struct Employ {
    name: String,
    age: u32,
    sex: String,
}

fn main() {
    let mut em: Employ = Employ {
        name: String::from("测试"),
        age: 30,
        sex: String::from("a"),
    };
    em.name = String::from("donglietao");
    println!("{}", em.name);
}
```

测试测试测试测试

```
fn main() {
    let em = get_employ();
    // 测试
    let em2 = Employ { ..em };
}
```

```

println!("{}", em.name);
println!("{}", em2.name);

// 打印
println!("{}", em2);
}

fn get_employ() -> Employ {
    let mut em: Employ = Employ {
        name: String::from("张三"),
        age: 30,
        sex: String::from("a"),
    };
    em.name = String::from("donglietao");
    return em;
}

// 打印
#[derive(Debug)]
struct Employ {
    name: String,
    age: u32,
    sex: String,
}

```

打印

```

fn main() {
    let em = Employ::new(String::from("张三"), 33, String::from("1"));
    println!("{}", em.to_string());
}

// 打印
#[derive(Debug)]
struct Employ {
    name: String,
    age: u32,
    sex: String,
}

```

```
// impl
impl Employ {
    // 
    fn new(name: String, age: u32, sex: String) -> Employ {
        return Employ {
            name: name,
            age: age,
            sex: sex,
        };
    }

    // 
    // self
    fn to_string(self) -> String {
        return self.name;
    }
}
```

```
enum Sex {
    Main = 0,
    Woman = 1,
}
```





rust 4

- package:
- crate
- modules use
- path

package crate

cargo new project crate src/main.rs src/lib.rs (library) package

module

```
/Cargo.toml
/src/
/src/main.rs
/src/ffmpeg_encoder/mod.rs
/src/ffmpeg_encoder/book.rs
/src/ffmpeg_encoder/people.rs
```

crate src/main.rs src/lib.rs

```
# src/main.rs

// 
use crate::ffmpeg_encoder::book::Book;
use crate::ffmpeg_encoder::people::Employ;
use crate::ffmpeg_encoder::people::Sex;
use crate::ffmpeg_encoder::Shop;

// root module
pub mod ffmpeg_encoder;
```

```
fn main() {

    println!("{}", Shop::to_string());

    println!("{}", Book::new().to_string());

    let em = Employ::new(String::from("[]"), 33, Sex::Woman);
    println!("{}", em.to_string());
}
```

```
# [REDACTED]rust[REDACTED]
# src/ffmpeg_encoder/mod.rs
# src/ffmpeg_encoder.rs

pub mod book;
pub mod people;

pub struct Shop {}

impl Shop {
    pub fn to_string() -> String {
        return String::from("Shop to_string");
    }
}
```

```
# src/ffmpeg_encoder/book.rs

pub struct Book;

impl Book {
    pub(crate) fn new() -> Book {
        Book {}
    }

    pub(crate) fn to_string(&self) -> String {
        return String::from("Book to_string");
    }
}
```

```
}  
}
```

```
# 测试  
# src/ffmpeg_encoder/people.rs  
  
#[derive(Debug)]  
pub struct Employ {  
    name: String,  
    age: u32,  
    sex: Sex,  
}  
  
impl Employ {  
    pub fn new(name: String, age: u32, sex: Sex) -> Employ {  
        return Employ {  
            name: name,  
            age: age,  
            sex: sex,  
        };  
    }  
    // 测试self  
    pub fn to_string(&self) -> String {  
        return self.name.to_string();  
    }  
}  
  
#[derive(Debug)]  
pub enum Sex {  
    Main = 0,  
    Woman = 1,  
}
```



```
let args: Vec<String> = std::env::args().collect();

for arg in args {
    println!("{}", arg);
}
```



```
# Cargo.toml

# [REDACTED]

[dependencies]

log = "0.4.20"

simple_logger = "4.3.0"

# [REDACTED]rust[REDACTED]
```

```
# src/main.rs

use log::{debug, info, trace, warn};
use simple_logger::SimpleLogger;

fn main(){
    // 打印日志
    SimpleLogger::new().init().unwrap();
    info!("application start");
}
```

# JSON

```
# Cargo.toml

# [dependencies]

serde = { version = "1.0", features = ["derive"] }
serde_json = "1.0.68"
```

```
# src/main.rs

use log::{debug, info, trace, warn};
use serde::{Deserialize, Serialize};
use serde_json::json;
use simple_logger::SimpleLogger;

#[derive(Serialize, Deserialize)]
struct Person {
    name: String,
    age: u32,
}

async fn root() -> impl IntoResponse {
    info!("entry root");

    // 接收JSON数据
    // 解析JSON数据
    let json_str = r#"{"name": "John Doe", "age": 30}"#;

    // 解析JSON数据
    let json_obj: Person = serde_json::from_str(json_str).unwrap();

    // 生成JSON数据
    let json_str = serde_json::to_string(&json_obj).unwrap();
    log::info!("{}", json_str);

    // 生成JSON数据
    let json_obj = json!({"name": "John Doe", "age": 30});

    // 生成JSON数据
```

```
let json_str = json_obj.to_string();
log::info!("{}", json_str);

// 解析JSON字符串
let json_obj: Person = serde_json::from_value(json_str).unwrap();
log::info!("{}", json_obj.name);

return (StatusCode::OK, Json(json_obj));
}
```



- <https://lib.rs/>
- <https://crates.io/>



```
[dependencies]
# web
axum = "0.7.2"
#
tokio = { version = "1.35.1", features = ["full"] }
# http client
reqwest = "0.11.23"
#
log = "0.4.20"
#
simple_logger = "4.3.0"
#
serde = { version = "1.0", features = ["derive"] }
# json
serde_json = "1.0.68"
```

# RESTful API

## 

```
[dependencies]
# web
axum = "0.7.2"
# I/O
tokio = { version = "1.35.1", features = ["full"] }
# http client
reqwest = "0.11.23"
# 
log = "0.4.20"
# 
simple_logger = "4.3.0"
# 
serde = { version = "1.0", features = ["derive"] }
# json
serde_json = "1.0.68"
```

## http server

```
# src/main.rs

// 
use axum::{
    http::StatusCode,
    response::{Html, IntoResponse, Json},
    routing::{get, post},
    Router,
};
use log::{debug, info, trace, warn};
use serde::{Deserialize, Serialize};
use serde_json::json;
use simple_logger::SimpleLogger;
```



```

#[tokio::main]
async fn main() {
    // 初始化
    SimpleLogger::new().init().unwrap();
    debug!("power by iovhm.com");
    warn!("©copyRight");
    trace!("2023-12-27");
    info!("application start");

    let app = Router::new()
        // GET /
        .route("/", get(root));

    let listener = tokio::net::TcpListener::bind("0.0.0.0:3000").await.unwrap();
    info!("server listen at 0.0.0.0:3000");

    axum::serve(listener, app).await.unwrap();
}

#[derive(Serialize, Deserialize)]
struct Person {
    name: String,
    age: u32,
}

async fn root() -> impl IntoResponse {
    info!("entry root");

    // 返回JSON
    let json_obj = json!({"name": "John Doe", "age": 30});

    return Json(json_obj);
}

```