Compilación y Ejecución en Rust

La ejecución de código en Rust puede realizarse mediante dos vías fundamentales:

- Compilador directo, rustc, para tareas sencillas.
- Cargo la herramienta estándar de gestión de proyectos, indispensable para el desarrollo moderno.

Usando rustc

Para comprender la esencia del proceso de compilación, es crucial familiarizarse con rustc, el compilador de Rust. Este método es ideal para archivos individuales o para entender cómo el código fuente se traduce en un ejecutable binario.

Fases del Proceso rustc

1. Paso 1: Creación del Módulo Fuente

   Todo comienza con el código fuente, que tradicionalmente lleva la extensión .rs.

   ```rust
   fn main() {
       println!("Compilador directo rustc."); //Archivo: main.rs
   }
   ```

2. Paso 2: Compilación

   Desde la terminal, se invoca a rustc, apuntando al archivo de entrada. El compilador lee el código y genera un archivo binario ejecutable en el mismo directorio.

   ```
   rustc main.rs
   ```

   En este proceso, rustc maneja internamente la verificación de tipos, el borrow checker y la generación del código máquina optimizado, utilizando LLVM.

3. Ejecución

   Esto genera un ejecutable.

   - Windows:

     ```
     .\main.exe
     ```

   - Linux/macOS:

     ```
     ./main
     ```

   Resultado:

   ```
   Compilador directo rustc.
   ```

Tu Primer Proyecto con Cargo

Crear un nuevo proyecto

```
# Crear un proyecto binario (aplicación)
cargo new a hola_mundo b
```

- a: Crear un nuevo proyecto Rust
- b: Nombre del proyecto

```
# Entrar al directorio                                    Terminal
cd hola_mundo
```

Estructura creada:

```
hello_world/ Raíz del proyecto                                Output
├── Cargo.lock Registra las versiones específicas
├── Cargo.toml Define las dependencias
├── src/
│   └── main.rs Código fuente principal
└── target/ Destino de la Compilación
    └── debug/
        ├── build/
        └── deps/
            └── hello_world El ejecutable de tu aplicación
```

Anatomía del Proyecto: Cargo.toml

Contenido inicial de Cargo.toml

```
[package]                                                     ⚙toml
name = "hola_mundo" Nombre del proyecto
version = "0.1.0" Versión siguiendo                    } (1) Metadatos
edition = "2021" Edición estable de Rust 2021

[dependencies] crates
# Ejmplos
# rand = "0.8.5" Permite generar números aleatorios    } (2) Librerías externas

# serde = "1.0.130" Permite serializar y deserializar datos
```

Punto de entrada: main.rs

```
fn main() A { C                                               Rust
    println! B("Hola, Rust!");
} C
```

1. A: Define la función principal del programa
2. B: Es una macro que imprime texto en la consola
3. C: Delimitan el bloque de código de la función

Compilar y Ejecutar

```
cargo run                                                     Terminal
```

```
Compiling hola_mundo v0.1.0 (/ruta/hola_mundo) A                    Terminal
 Finished `dev` profile [unoptimized + debuginfo] target(s) in 3.42s B
  Running `target/debug/hola_mundo`  // (3) C
Hola, Rust! D
```

1. A: Cargo compila el proyecto (solo la primera vez o si hay cambios)
2. B: Perfil de compilación: dev (desarrollo, sin optimizaciones)
3. C: Ruta del ejecutable que se está ejecutando
4. D: Output de tu programa

Si no modificaste el código, la segunda ejecución será instantánea:

```
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.01s    Terminal
 Running `target/debug/hola_mundo`
Hola, Rust!
```

Solo compilar sin ejecutar

```
cargo build
```

El binario se genera en target/debug/:

Sistema	**Ubicación del ejecutable**
Windows	`target\debug\hola_mundo.exe`
Linux/macOS	`target/debug/hola_mundo`