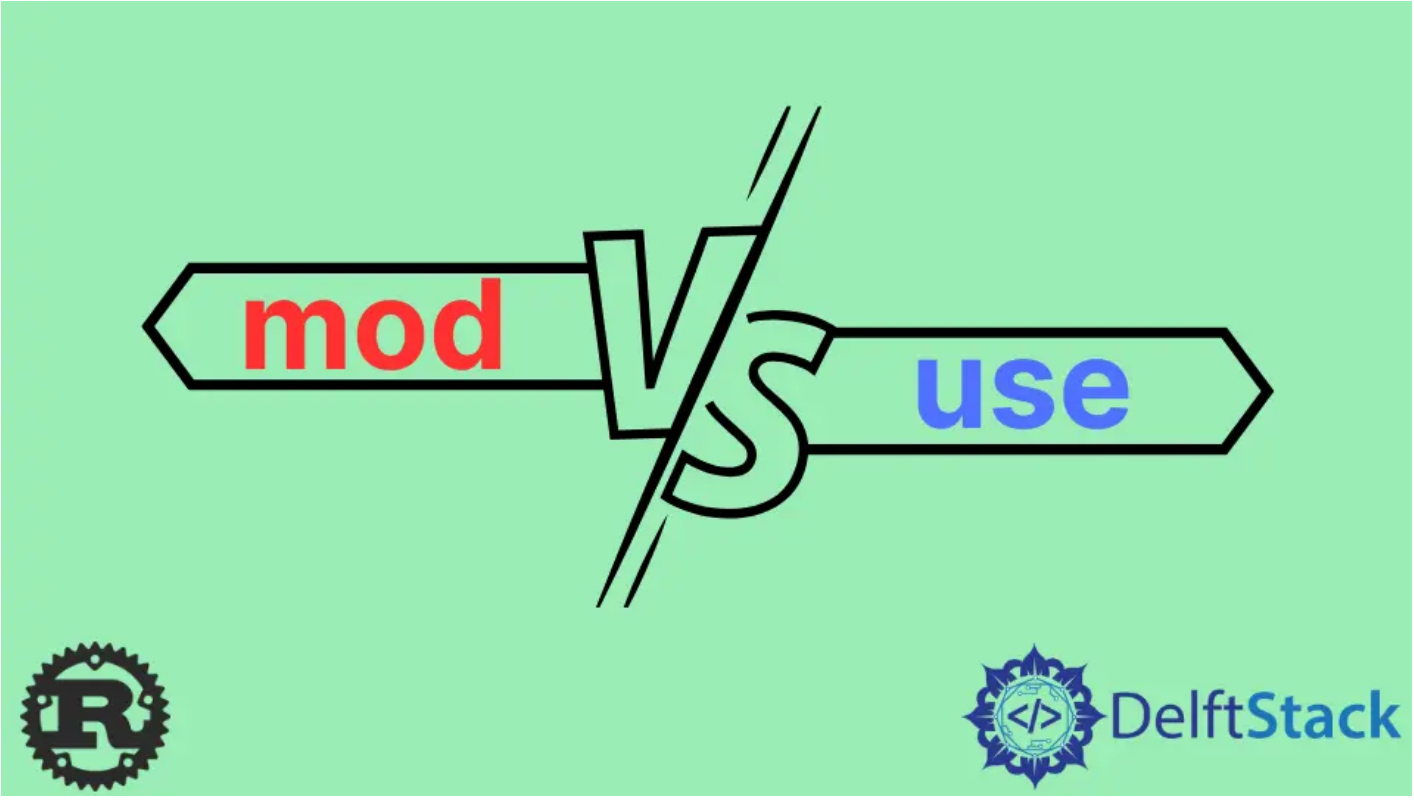
Diferencia entre mod y use en Rust

Muhammad Adil 30 enero 2023

Rust (/es/tags/rust/) Rust Use (/es/tags/rust-use/) (http://www.facebook.com/sharer.php?src=bm&u=https%3a%2f%2fwww.delftstack.com%2fes%2fhowto%2frust%2frust-mod-vs-use%2f&t=Diferencia%20entre%20mod%20y%20use%20en%20Rust) (https://twitter.com/intent/tweet/?text=Howtos%20de%20Rust-Diferencia%20entre%20mod%20y%20use%20en%20Rust&url=https%3a%2f%2fwww.delftstack.com%2fes%2fhowto%2mod-vs-use%2f&hashtags=web,development) (https://www.linkedin.com/shareArticle?mini=true&url=https%3a%2f%2fwww.delftstack.com%2fes%2fhowto%2frust%2frust-mod-vs-use%2f&title=Howtos%20de%20Rust-Diferencia%20entre%20mod%20y%20use%20en%20Rust&source=https%3a%2f%2fwww.delftstack.com%2fes%2fhowtmod-vs-use%2f&summary=Short%20summary)

TABLA DE CONTENIDO



Rust es un lenguaje de programación moderno y rápido que está diseñado para ser un lenguaje de programación de sistema seguro, concurrente y funcional. Fue creado por Mozilla en 2006.

Rust tiene tres conceptos principales: propiedad, préstamo y vida útil. Rust usa las dos palabras clave: `use` y `mod`.

La palabra clave `use` se utiliza para importar el contenido del módulo al ámbito actual. Esto significa que hará que todas las funciones dentro del módulo estén disponibles para llamar desde este punto en adelante.

`mod`, por otro lado, importa solo un único elemento de otro módulo a su alcance actual para que pueda ser llamado o referenciado según sea necesario sin preocuparse de que nada más en ese módulo sea accesible a partir de ahora.

Concepto de `use` en Rust

Al seguir una ruta diferente, `use` agrega otro elemento al espacio de nombres actual. Un elemento es un objeto genérico al que necesita acceder, como una función, una estructura o un rasgo.

Una ruta es una jerarquía de módulos que debe seguir para llegar a ella. El espacio de nombres actual se refiere a traer un objeto al archivo actual para que pueda acceder a él como si fuera local.

Características de `use` en Rust

Rust presenta un mecanismo bastante dinámico para los usuarios que simplifica la introducción de muchos objetos en el espacio de nombres existente con un esfuerzo mínimo.

1. Puede usar la palabra clave `self` para introducir el módulo principal universal y cualquier otra cosa que desee aprovechar en el espacio de nombres.
2. Para evitar problemas de identidad, use la palabra clave `as` para cambiar las cosas.
3. Puede usar la sintaxis similar a glob para traer varios objetos al espacio de nombres actual:

```
use std::path::{self, Path, PathBuf};
```

Concepto de `mod` en Rust

Los módulos le permiten organizar su código en archivos separados. Dividen su código en piezas lógicas que pueden importarse y usarse en otros módulos o programas.

En resumen, `mod` se usa para especificar módulos y submódulos para que pueda usarlos en su archivo `.rs` actual, lo que puede ser complicado. ¿Por qué no simplemente usarlo?

Un módulo en Rust no es más que un contenedor para cero o más cosas. Es un método de organizar lógicamente los elementos para que su módulo pueda atravesarlos fácilmente.

Los módulos se pueden usar para crear la estructura de árbol de una caja, lo que le permite dividir su trabajo en varios archivos arbitrariamente profundos si es necesario. Un solo archivo `.rs` puede incluir varios módulos, o un solo archivo puede contener varios módulos.

Por último, puede usar `mod` para agrupar objetos de forma lógica. Puede construir bloques `mod` en un solo archivo `.rs`, o puede dividir su código fuente en varios archivos `.rs`, usar `mod` y dejar que Cargo genere la estructura de árbol de su caja.

Diferencia básica entre `mod` y `use` en Rust

La principal diferencia entre ellos es que `use` importa un módulo de una biblioteca externa, mientras que `mod` crea un módulo interno que solo se puede usar dentro del archivo actual.

Analicemos un ejemplo usando la palabra clave `usar`.

```
use hello::rota::function as my_function;
fn function() {
    println!("demo `function()`");
}
mod hello {
    pub mod rota {
        pub fn function() {
            println!("demo `hello::rota`");
        }
    }
}
fn main() {
    my_function();

    println!("Coming");
    {
        use crate::hello::rota::function;
        function();
        println!("Returning");
    }
    function();
}
```

Producción :

```
demo `hello::rota`
Coming
demo `hello::rota`
Returning
demo `function()`
```

Haga clic aquí (https://play.rust-lang.org/?version=stable&mode=debug&edition=2021&gist=e2d676291d74b5af0859b91d8be92326) para ver la demostración en vivo del código mencionado anteriormente.

Autor: **Muhammad Adil** (/es/author/muhammad-adil/)

(/es/author/muhammad-adil/)

(/es/author/muhammad-adil/)

Muhammad Adil is a seasoned programmer and writer who has experience in various fields. He has been programming for over 5 years and have always loved the thrill of solving complex problems. He has skilled in PHP, Python, C++, Java, JavaScript, Ruby on Rails, AngularJS, ReactJS, HTML5 and CSS3. He enjoys putting his experience and knowledge into words.

Facebook (https://www.facebook.com/profile.php?id=100011992218809/)