

Artículo

[Eduardo Anglada](#) · 24 mayo, 2021 · Lectura de 4 min

[Open Exchange](#)

Cómo acceder a IRIS desde Rust

Qué te parece si te digo que muy pronto te podrás conectar a IRIS desde la aplicación escrita en Rust...

¿Qué es Rust?

Rust es un lenguaje de programación multiparadigma diseñado teniendo en cuenta el rendimiento, la seguridad y especialmente que la concurrencia sea segura. Rust es sintácticamente similar a C++, pero puede garantizar la seguridad de la memoria mediante el uso de un verificador de préstamos para validar las referencias. Rust logra la seguridad de la memoria sin emplear un recolector de basura, y el conteo de referencias es opcional.

(c) [Wikipedia](#).

Es el lenguaje más valorado durante los últimos cinco años en la [encuesta de Stack Overflow 2020](#).



¿Qué es posible ahora mismo?

Puede trabajar con globals y hacer consultas SQL sencillas. Observa el ejemplo de trabajo.

```
use irisnative;
use irisnative::{connection::*, global, global::Sub, Global};

fn main() {
    let host = "127.0.0.1";
    let port = 1972;
    let namespace = "USER";
    let username = "_SYSTEM";
    let password = "SYS";
    match irisnative::connect(host, port, namespace, username, password) {
        Ok(mut connection) => {
            println!("Connection established");

            println!("Server: {}", connection.server_version());

            connection.kill(&global!(A));
            connection.set(&global!(A(1)), "1");
            connection.set(&global!(A(1, 2)), "test");
            connection.set(&global!(A(1, "2", 3)), "123");
            connection.set(&global!(A(2, 1)), "21test");
            connection.set(&global!(A(3, 1)), "test31");

            let mut global = global!(A(""));
            while let Some(key) = connection.next(&mut global) {
                println!("^A({:?}) = {:?}", key, {
                    if connection.is_defined(&global).0 {
```

```

        let value: String = connection.get(&global).unwrap();
        value
    } else {
        String::from("&lt;UNDEFINED>")
    }
});
let mut global1 = global!(A(key, ""));
while let Some(key1) = connection.next(&mut global1) {
    let value: String;
    if connection.is_defined(&global1).0 {
        value = connection.get(&global1).unwrap();
    } else {
        value = String::from("&lt;UNDEFINED>");
    }
    println!("^A({:?}, {:?}) = {:?}", key, key1, value);
}
}

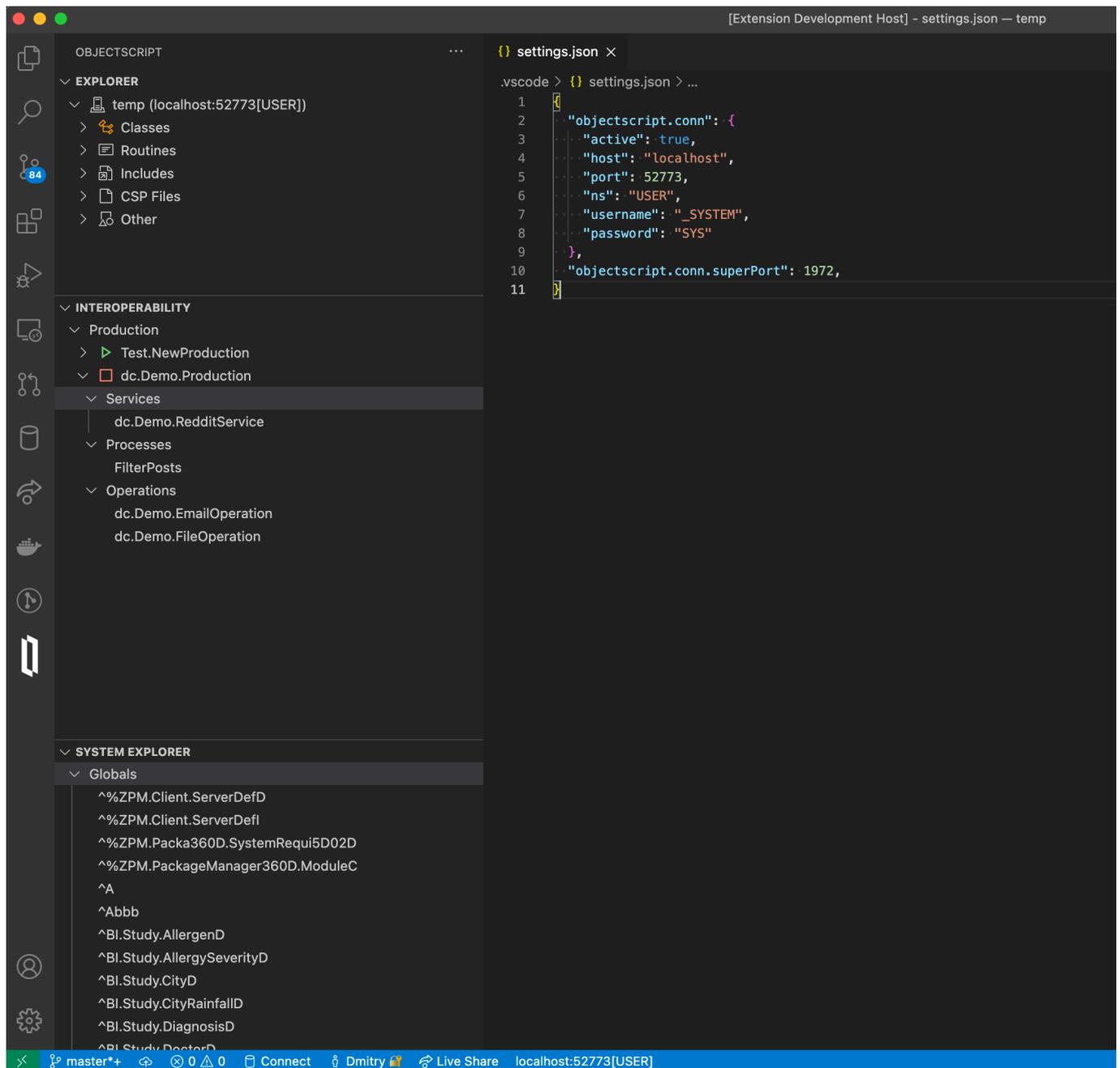
let mut rs = connection.query(String::from(
    "SELECT Name from %Dictionary.ClassDefinition WHERE Super = 'Ens.Production'
and Abstract&lt;>1"));
while rs.next() {
    let name: String = rs.get(0).unwrap();
    println!("{}", name);
}
}
Err(err) => {
    println!("Error: {}", err.message);
}
}
}
}

```

Por lo tanto, será posible conectarse a IRIS por medio de la red, tan pronto como tenga acceso al puerto del súper servidor (1972).

Caso de uso real

Para utilizarlo en producción, tiene que ser compilado en un archivo ejecutable o librería. Yo tengo un proyecto donde lo uso ahora. Es la extensión VSCode para el control avanzado de InterSystems IRIS. El proyecto participó en el [Gran Premio para Desarrolladores de InterSystems](#). Rust ayuda a obtener acceso directo a IRIS, y será posible comprobar el estado o detener/iniciar la producción, observar los globals y muchas otras cosas en el futuro.



Rust tiene que ser compilado en un formato binario para la plataforma deseada, y por el momento esta extensión se desarrolló para macOS x64 y Windows x64. Pero Rust se puede compilar para una amplia gama de plataformas, incluyendo Arm64.

Veamos qué más se puede hacer

Vamos a intentar ejecutar la aplicación Rust (del ejemplo anterior) con el conector IRIS en el Docker. En Dockerfile simple se puede construir una imagen con la aplicación.

```
FROM ekidd/rust-musl-builder
```

```
ADD --chown=rust:rust . ./
```

```
RUN cargo build --release --example main
```

```
FROM scratch
```

```
COPY --from=0 /home/rust/src/target/x86_64-unknown-linux-musl/release/examples/main /
```

```
CMD [ "/main" ]
```

Vamos a crearla

```
$ docker build -t rust-irisnative .
```

Y la ejecutamos

```
$ docker run -it rust-irisnative
```

```
Connection established
```

```
Server: IRIS for UNIX (Ubuntu Server LTS for x86-64 Containers) 2020.4 (Build 524U) T  
hu Oct 22 2020 13:04:25 EDT
```

```
^A("1") = "1"
```

```
^A("1", "2") = "test"
```

```
^A("2") = "&lt;UNDEFINED>"
```

```
^A("2", "1") = "2test"
```

```
^A("3") = "&lt;UNDEFINED>"
```

```
^A("3", "1") = "test31"
```

```
Test.NewProduction
```

```
dc.Demo.Production
```

Si te preguntas qué tamaño tiene esa imagen:

```
$ docker images rust-irisnative
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
rust-irisnative	latest	0ble54e7aa6f	2 minutes ago	3.92MB

Solo unos pocos megabytes de la imagen pueden conectarse a IRIS, que se ejecuta en otro lugar. Parece que es muy bueno para los microservicios y las aplicaciones sin servidor. Y como ventaja para el IoT, las aplicaciones de Rust pueden ejecutarse en pequeños PCs, por ejemplo, RaspberryPi Pico. ¿Qué te parece y cómo utilizarías Rust?

[#Despliegue](#) [#Lenguajes](#) [#VSCode](#) [#InterSystems IRIS](#) [#InterSystems IRIS for Health](#)
[Ir a la aplicación en InterSystems Open Exchange](#)

URL de fuente: <https://es.community.intersystems.com/post/c%C3%B3mo-acceder-iris-desde-rust>