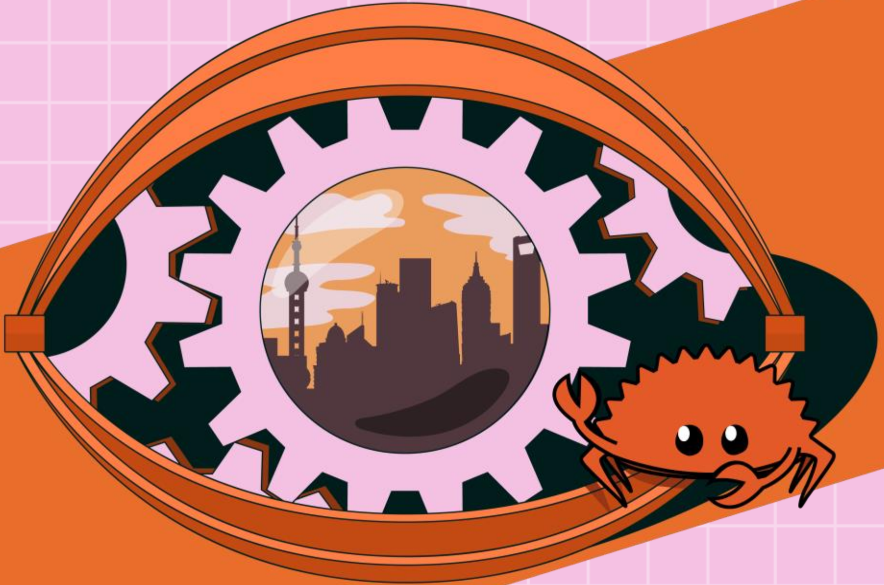
# Rust 异步 Runtime 的兼容层

施继成 @ DatenLord



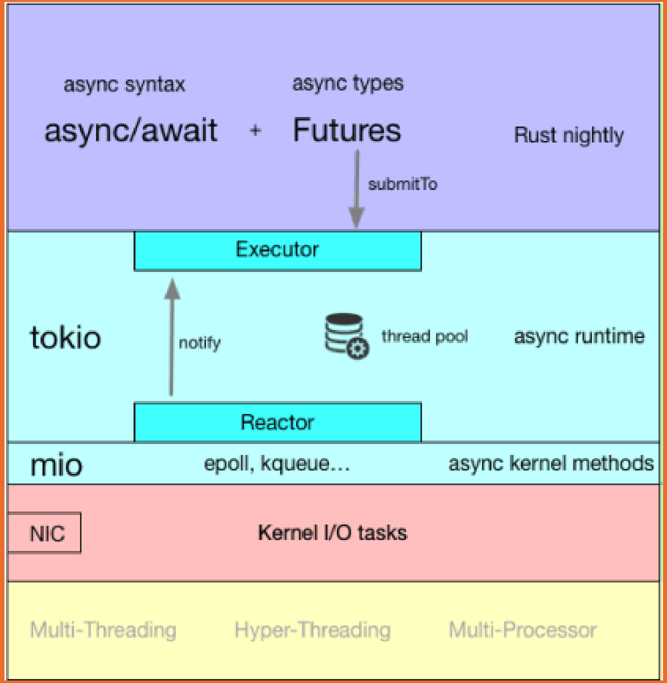
6.17-6.18 @Shanghai

## Table of Contents

1

## # Rust async runtime

## Rust async runtime



## Light-weight task

- Language and compiler define tasks
- How to run it?
- When to run it?
- How does it deal with the I/O?

Rust async runtime

## Runtime responsibilities

- Invoke waiting tasks and halt tasks
- Get notifications from the OS
- Schedule tasks across threads if it’s multi-thread model

Rust async runtime

## Available Runtimes

- Tokio
- Async-std
- Smol
- Monoio

2

## # Async runtime binding

Async runtime binding

## Which runtime to choose ?

- More adopters
- Rich eco-system
- Rich out-of-box features

- Maybe better performance
- Clean interface

Async runtime binding

## Eco system binding

- Panic "**not currently running on the Tokio runtime**"
- Hyper -- fast and safe HTTP for tokio
- Surf -- HTTP client framework for async-std

Async runtime binding

## Barriers on runtime switch

- Switch all I/O related data structures
- Switch all async macros
- Switch all functions
- Scan everywhere – We have to provide an abstraction to avoid that

Async runtime binding

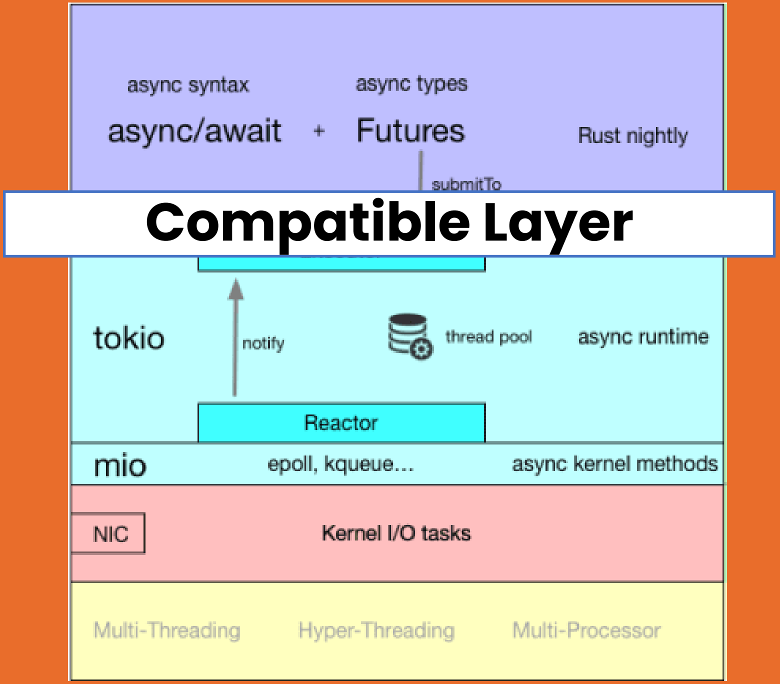
## Build libs for all runtimes

- Impossible – You don’t know how many runtimes there
- Provide a wrapper for runtimes
- Easy switch with rust features and conditional compilation

3

## # Compatible layer

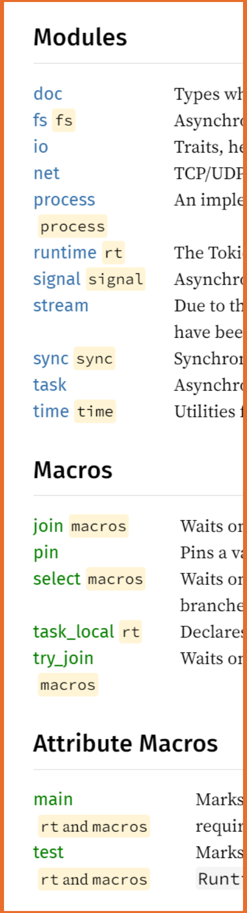
## Compatible layer



# Insight of compatible layer

- Rust lang and compiler → syntax and task type
- Async compatible layer → I/O and task management

## Compatible layer



**Modules**

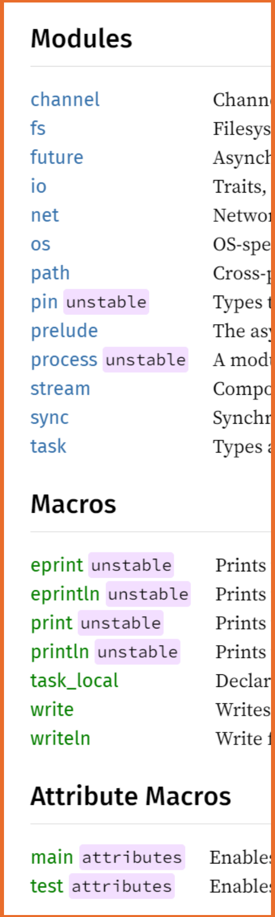
| | |
|---|---|
| doc | Types wh |
| fs fs | Asynchro |
| io | Traits, he |
| net | TCP/UDP |
| process process | An imple |
| runtime rt | The Toki |
| signal signal | Asynchro |
| stream | Due to th have bee |
| sync sync | Synchron |
| task | Asynchro |
| time time | Utilities f |

**Macros**

| | |
|---|---|
| join macros | Waits or |
| pin | Pins a va |
| select macros | Waits or branche |
| task_local rt | Declares |
| try_join macros | Waits or |

**Attribute Macros**

| | |
|---|---|
| main rt and macros | Marks requir |
| test rt and macros | Marks Runt |



**Modules**

| | |
|---|---|
| channel | Chann |
| fs | Filesys |
| future | Asynch |
| io | Traits, |
| net | Networ |
| os | OS-spe |
| path | Cross-p |
| pin unstable | Types t |
| prelude | The asy |
| process unstable | A modu |
| stream | Compo |
| sync | Synchr |
| task | Types a |

**Macros**

| | |
|---|---|
| eprint unstable | Prints |
| eprintln unstable | Prints |
| print unstable | Prints |
| println unstable | Prints |
| task_local | Declar |
| write | Writes |
| writeln | Write f |

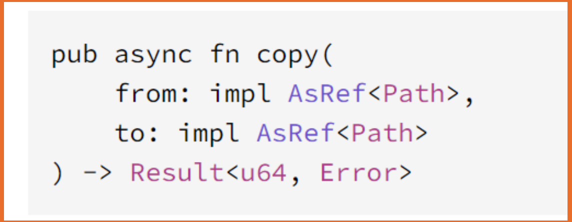
**Attribute Macros**

| | |
|---|---|
| main attributes | Enables |
| test attributes | Enables |

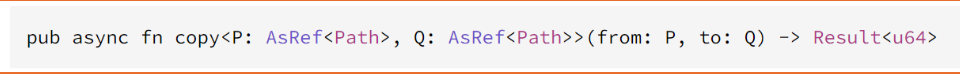
# Compare runtimes

- Tokio
- Async-std
- Similar component structures

## Compatible layer



```rust
pub async fn copy(
    from: impl AsRef<Path>,
    to: impl AsRef<Path>
) -> Result<u64, Error>
```



```rust
pub async fn copy<P: AsRef<Path>, Q: AsRef<Path>>(from: P, to: Q) -> Result<u64>
```

## Compare runtimes

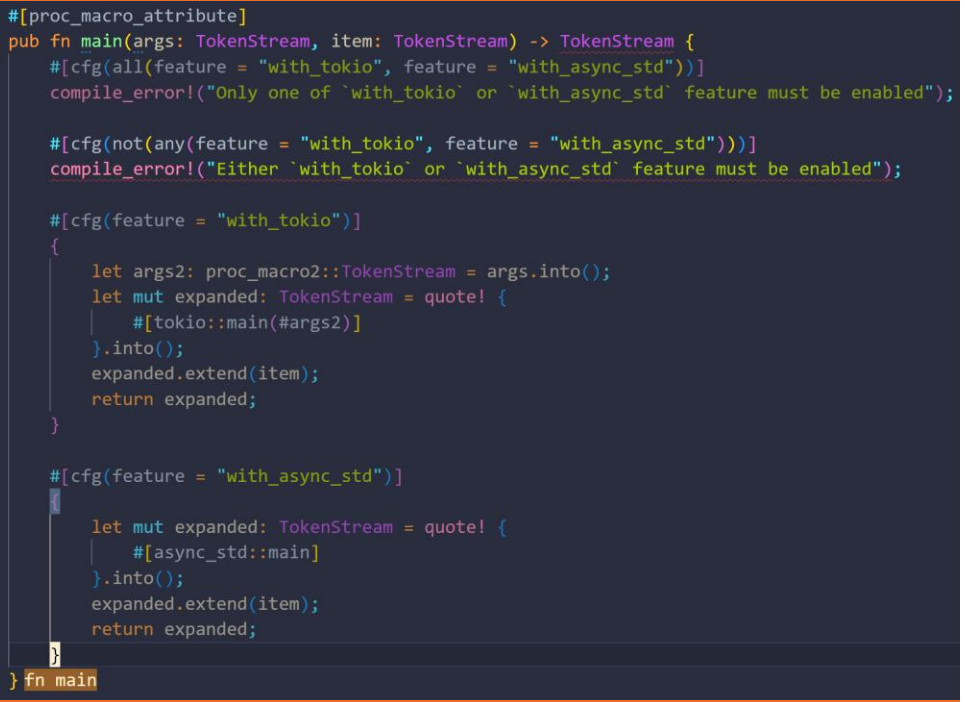
- Tokio
- Async-std
- Almost the same APIs

Compatible layer

## Main components in async runtimes

- Macros
- Data structures and associate functions
- Raw functions

## Compatible layer

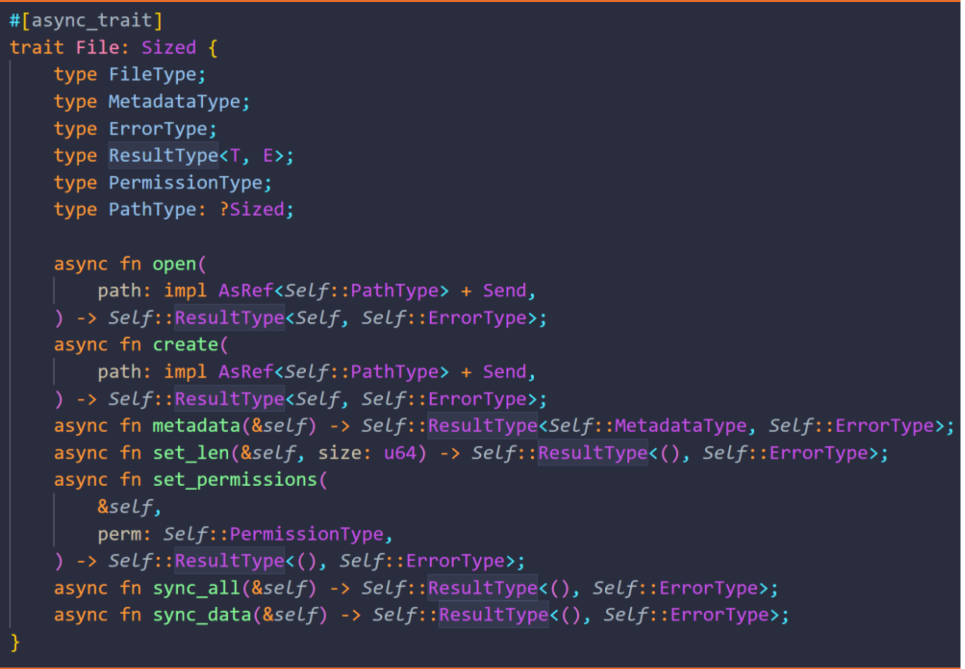


```
#[proc_macro_attribute]
pub fn main(args: TokenStream, item: TokenStream) -> TokenStream {
    #[cfg(all(feature = "with_tokio", feature = "with_async_std"))]
    compile_error!("Only one of `with_tokio` or `with_async_std` feature must be enabled");

    #[cfg(not(any(feature = "with_tokio", feature = "with_async_std")))]
    compile_error!("Either `with_tokio` or `with_async_std` feature must be enabled");

    #[cfg(feature = "with_tokio")]
    {
        let args2: proc_macro2::TokenStream = args.into();
        let mut expanded: TokenStream = quote! {
            #[tokio::main(#args2)]
        }.into();
        expanded.extend(item);
        return expanded;
    }

    #[cfg(feature = "with_async_std")]
    {
        let mut expanded: TokenStream = quote! {
            #[async_std::main]
        }.into();
        expanded.extend(item);
        return expanded;
    }
} fn main
```

# Macro wrapper

- Conditional compiling
- Attribute proc macro

## Compatible layer

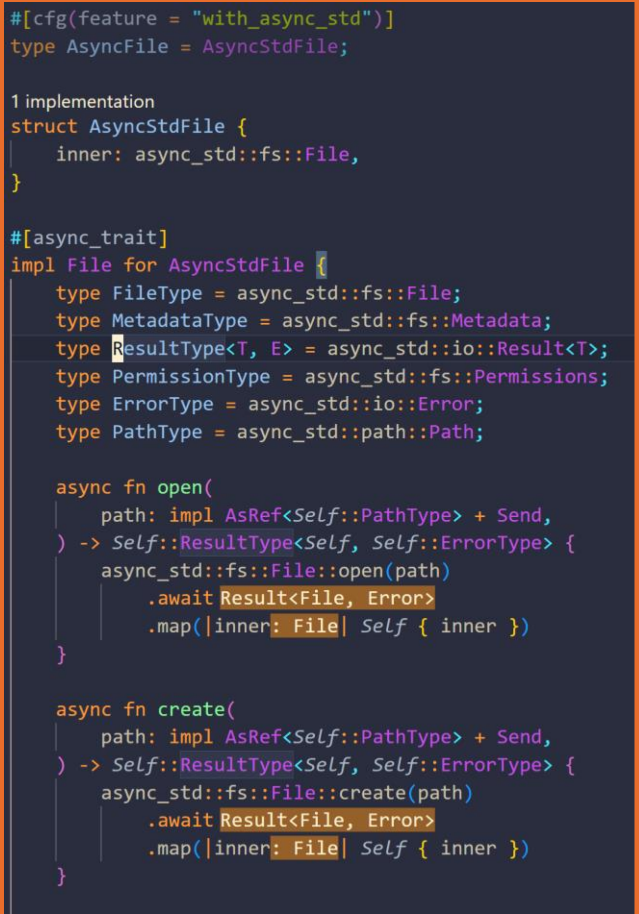


```rust
#[async_trait]
trait File: Sized {
    type FileType;
    type MetadataType;
    type ErrorType;
    type ResultType<T, E>;
    type PermissionType;
    type PathType: ?Sized;

    async fn open(
        path: impl AsRef<Self::PathType> + Send,
    ) -> Self::ResultType<Self, Self::ErrorType>;
    async fn create(
        path: impl AsRef<Self::PathType> + Send,
    ) -> Self::ResultType<Self, Self::ErrorType>;
    async fn metadata(&self) -> Self::ResultType<Self::MetadataType, Self::ErrorType>;
    async fn set_len(&self, size: u64) -> Self::ResultType<(), Self::ErrorType>;
    async fn set_permissions(
        &self,
        perm: Self::PermissionType,
    ) -> Self::ResultType<(), Self::ErrorType>;
    async fn sync_all(&self) -> Self::ResultType<(), Self::ErrorType>;
    async fn sync_data(&self) -> Self::ResultType<(), Self::ErrorType>;
}
```

## Data structure wrapper

- The same type name but different type
- GAT
- Trait abstraction

## Compatible layer

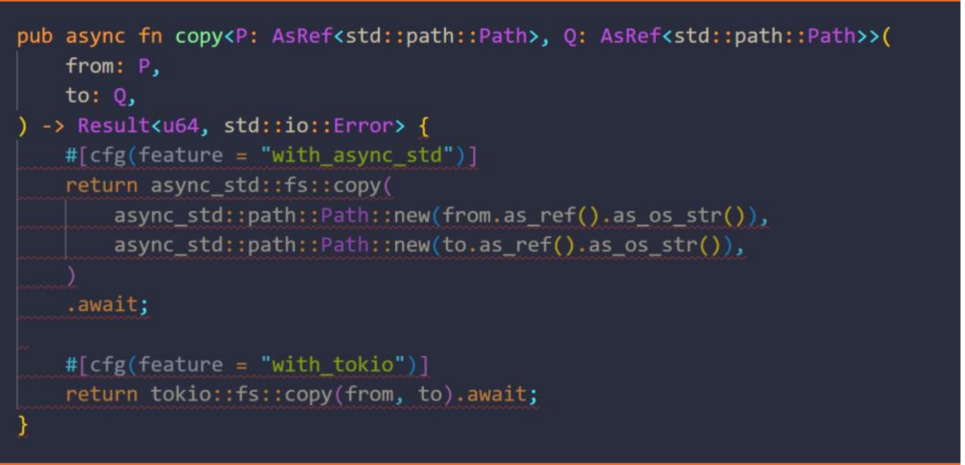


```
#[cfg(feature = "with_async_std")]
type AsyncFile = AsyncStdFile;

1 implementation
struct AsyncStdFile {
    inner: async_std::fs::File,
}

#[async_trait]
impl File for AsyncStdFile {
    type FileType = async_std::fs::File;
    type MetadataType = async_std::fs::Metadata;
    type ResultType<T, E> = async_std::io::Result<T>;
    type PermissionType = async_std::fs::Permissions;
    type ErrorType = async_std::io::Error;
    type PathType = async_std::path::Path;

    async fn open(
        path: impl AsRef<Self::PathType> + Send,
    ) -> Self::ResultType<Self, Self::ErrorType> {
        async_std::fs::File::open(path)
            .await Result<File, Error>
            .map(|inner: File| Self { inner })
    }

    async fn create(
        path: impl AsRef<Self::PathType> + Send,
    ) -> Self::ResultType<Self, Self::ErrorType> {
        async_std::fs::File::create(path)
            .await Result<File, Error>
            .map(|inner: File| Self { inner })
    }
```

# Data structure wrapper cont.

- The same type name but different type
- GAT
- Trait abstraction

## Compatible layer



```
pub async fn copy<P: AsRef<std::path::Path>, Q: AsRef<std::path::Path>>(
    from: P,
    to: Q,
) -> Result<u64, std::io::Error> {
    #[cfg(feature = "with_async_std")]
    return async_std::fs::copy(
        async_std::path::Path::new(from.as_ref().as_os_str()),
        async_std::path::Path::new(to.as_ref().as_os_str()),
    )
    .await;

    #[cfg(feature = "with_tokio")]
    return tokio::fs::copy(from, to).await;
}
```

## Function wrapper

- Conditional compiling
- Type conversion

Compatible layer

## Limitations

- Conditional compiling → global single runtime
- External Libs (Http, S3, etc.)
- Force the libs use this compatible layer
- We provide a layer to wrap the popular utils, such as HTTP compatible layer

# Thank you！

DatenLord小助手

达坦科技 DatenLord

扫一扫上面的二维码图案，加我为朋友。