さうすの Rust 勉強会

lesson03

内容

- データベースも使ったサーバーを作る

sqlx

- マクロでデータベースを叩く
- コンパイル時型チェックが入っている
- いろんなデータベースをサポートしているが、今回は PostgreSQL を使う

環境構築

- `direnv` を入れよう

direnv

cargo-expand

`sqlx::query!`

コンパイル時型チェック付きのクエリ用マクロ。

```rust
#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let pool =
        sqlx::PgPool::connect(
            &std::env::var("DATABASE_URL").expect("DATABASE_URL must be set"))
            .await?;

    let items = sqlx::query!("SELECT * FROM items").fetch_all(&pool).await?;
    println!("{:?}", &items);

    Ok(())
}
```

`sqlx::query!` で生成されたコード

```
let items = {
    {
        #[allow(clippy::all)]
        {
            use ::sqlx::Arguments as _;
            let query_args = <sqlx::postgres::Postgres as ::sqlx::database::HasArguments>::Arguments::default();
            struct Record { name: String, value: String }
            #[automatically_derived]
            impl ::core::fmt::Debug for Record {
                fn fmt(&self, f: &mut ::core::fmt::Formatter) -> ::core::fmt::Result {
                    ::core::fmt::Formatter::debug_struct_field2_finish(f, "Record", "name", &self.name, "value", &&self.value)
                }
            }
            ::sqlx::query_with::<sqlx::postgres::Postgres, _>("SELECT * FROM items", query_args)
                .try_map(|row: sqlx::postgres::PgRow| {
                    use ::sqlx::Row as _;
                    let sqlx_query_as_name = row.try_get_unchecked::<String, _>(0usize)?;
                    let sqlx_query_as_value = row.try_get_unchecked::<String, _>(1usize)?;
                    Ok(Record { name: sqlx_query_as_name, value: sqlx_query_as_value, })
                })
        }
    }
}
    .fetch_all(&pool)
    .await?;
```

sqlx::query_as!

Row ではなく、クエリの結果を構造体に格納する

```rust
#[derive(Debug)]
struct Item { name: String, value: String }

#[tokio::main]
async fn main() -> anyhow::Result<()> {
    let pool =
        sqlx::PgPool::connect(
            &std::env::var("DATABASE_URL").expect("DATABASE_URL must be set"))
            .await?;

    let items = sqlx::query_as!(Item, "SELECT * FROM items")
        .fetch_all(&pool)
        .await?;
    println!("{:?}", &items);

    Ok(())
}
```

`sqlx::query_as!` で生成されたコード

```
let items = {
    {
        #[allow(clippy::all)]
        {
            use ::sqlx::Arguments as _;
            let query_args = <sqlx::postgres::Postgres as ::sqlx::database::HasArguments>::Arguments::default();
            ::sqlx::query_with::<sqlx::postgres::Postgres, _>("SELECT * FROM items", query_args)
                .try_map(|row: sqlx::postgres::PgRow| {
                    use ::sqlx::Row as _;
                    let sqlx_query_as_name = row.try_get_unchecked::<String, _>(0usize)?;
                    let sqlx_query_as_value = row.try_get_unchecked::<String, _>(1usize)?;
                    Ok(Item { name: sqlx_query_as_name, value: sqlx_query_as_value })
                })
        }
    }
}
    .fetch_all(&pool)
    .await?;
```

`sqlx::query`

たまに `sqlx::query!` で書けないものがあって、そのときは `sqlx::query`（`!` がない）を使う。例えば

```
enum OrderBy { DESC, ASC }
fn get_all_entries(conn: &mut PgConnection, order_by: OrderBy) -> Vec<Entry> {
    let order_by_str = if order_by == DESC { "DESC" } else { "ASC" };
    sqlx::query(&format!("
        SELECT * FROM entries
        ORDER BY {order_by_str}
    "))
    .fetch_all(&mut *conn)
    .await
    .unwrap()
}
```

axum と sqlx の使い方

テストの書き方

https://docs.rs/sqlx/latest/sqlx/attr.test.html

`#[sqlx::test]` を使うと、`sqlx::PgPool` をテスト関数の引数から取れる。しかもテストごとに独立なので、並列にテストを実行することができる。`sqlx` がデータベースの作成・マイグレーション・削除までしてくれて、必要なテーブルが作成した状態でテストを作成することができる。