库文档参考地址
https://docs.rs/
https://github.com/launchbadge/sqlx
https://docs.rs/sqlx/latest/sqlx/

1 Rust 数据库框架SQLx使用

1.1 简介

SQLx是一个rust异步**数据库框架**，目前最新版本为v0.5.13，文档不完善。不同于**diesel**这类orm框架，没有DSL，用户自己编写sql语句，将查询结果按列取出或映射到struct上。

- 支持 async-std 和 **tokio**
- 支持 postgresql 、**mysql**/maridb、sqlite
- 纯rust实现mysql和postgresql 访问驱动程序（sqlite使用了 libsqlite3 C 库）
- 支持TLS
- 嵌套事务

1.2 添加依赖

我们这里选择的异步依赖框架为Tokio，数据库为mysql

tokio

```
sqlx = { version = "0.5", features = [ "runtime-tokio-native-tls" , "mysql" ] }
tokio = { version = "1", features = ["full"] }
```

1.3 连接mysql

sqlx 自带连接池

```
let db_pool    =
MySqlPool::connect("mysql://root:123456@127.0.0.1/actix_user_crud").await?;
```

或者

```
let db_pool = MySqlPoolOptions::new()
      .connect_timeout(Duration::from_secs(10))
      .min_connections(50)
      .max_connections(100)
      .idle_timeout(Duration::from_secs(600))
      .connect(db_url)
      .await.unwrap();
```

1.4 范例代码

见：ex1_mysql_curd

2 Rust - redis

https://docs.rs/redis/0.21.5/redis/index.html
https://github.com/redis-rs/redis-rs

2.1 引入依赖

```
[dependencies.redis]
version = "*"
```

或者git版本

```
# [dependencies]

[dependencies.redis]
git = "https://github.com/mitsuhiko/redis-rs.git"
```

2.2 连接Redis

格式：
URL格式 redis://[][:<passwd>@][:port][/<db>]

连接Redis代码如下：

```
    let client = redis::Client::open("redis://:password@ip:port/")?;
    let mut con = client.get_connection()?;
```

```
extern crate redis;
use crate::redis::ConnectionLike;
use redis::Connection;

fn main() {
    let con = connection_redis().ok().unwrap();
    // 测试是否成功连接Reids
    let is_open = con.is_open();
    println!("isOk: {}", is_open);
}

/**
 * 连接connection_redis
 */
fn connection_redis() -> redis::RedisResult<Connection> {
    let client = redis::Client::open("redis://:password@ip:port/")?;
    let con = client.get_connection()?;
    Ok(con)
}
```

2.3 低级命令

低级命令是指直接使用Reids命令进行操作。

```rust
/**
 * 低级命令
 */
fn low_level_commands(con: &mut redis::Connection) -> redis::RedisResult<()> {
    // 在Redis上新增一条key为my_key, value为42的数据
    let _: () = redis::cmd("SET").arg("my_key").arg(42).query(con)?;

    // 获取key为my_key的value
    let my_key: i32 = redis::cmd("GET").arg("my_key").query(con)?;
    println!("my_key: {}", my_key);

    // hash 结构操作
    let _: () = redis::cmd("HSET")
        .arg("books")
        .arg("java")
        .arg(1)
        .arg("python")
        .arg(2)
        .query(con)?;
    let java_value: i32 =
redis::cmd("HGET").arg("books").arg("java").query(con)?;
    println!("java_value: {}", java_value);

    Ok(())
}
```

2.4 高级命令

高级命令是指封装之后的函数，使用这些函数可以很方便进行Redis操作。高级函数中封装了常用的命令方便操作Redis各种类型的数据。

```rust
/**
 * 高级命令
 */
fn high_level_commands(con: &mut redis::Connection) -> redis::RedisResult<()> {
    // String 类型操作
    con.set("count", 42)?;
    let count: i32 = con.get("count")?;
    println!("count: {}", count);

    con.incr("count", 100)?;
    let incr_count: i32 = con.get("count")?;
    println!("incr_count: {}", incr_count);

    // hash 类型操作
    con.hset("student", "name", "张三")?;
    con.hset("student", "age", 20)?;
    let name: String = con.hget("student", "name")?;
    println!("name: {}", name);

    // list操作
    con.lpush("students", "张三")?;
    con.lpush("students", "李四")?;
    let len: i32 = con.llen("students")?;
```

```
    println!("students lengtth: {}", len);

    // zset操作
    con.zadd("scores", "张三", 60)?;
    con.zadd("scores", "李四", 80)?;
    // 找到score 在70 - 100 之间的name
    let names: HashSet<String> = con.zrangebyscore("scores", 70, 100)?;

    for name in names {
        println!("name: {}", name);
    }

    Ok(())
}
```

2.5 事务

使用函数redis::transaction可以完成事务操作。这个函数会自动监视键，然后进入事务循环直到成功。一旦成功，结果就会返回。

```
fn transaction(con: &mut redis::Connection) -> redis::RedisResult<()> {
    let key = "transaction_test_key";
    con.set(key, 1)?;

    let (new_val,): (isize,) = redis::transaction(con, &[key], |con, pipe| {
        let old_val: isize = con.get(key)?;
        println!("old_val is: {}", old_val);
        pipe.incr(key, 2)
            .ignore()
            .incr(key, 100)
            .ignore()
            .get(key)
            .query(con)
    })?;
    println!("new_val is: {}", new_val);

    Ok(())
}
```

2.5 范例代码

见ex2_redis_curd

3 改写图床项目api

可以使用rust改下图床项目的api（除了文件上传的代码）
具体代码见：ex3_tc_api

3.1 注册

注册是一个简单的HTTP接口，根据用户输入的注册信息，创建一个新的用户。
请求URL

URL	http://42.194.128.13/api/reg
请求方式	POST
HTTP版本	1.1
Content-Type	application/json

请求参数

参数名	含义	规则说明	是否必须	缺省值
email	邮箱	必须符合email规范	可选	无
firstPwd	密码	md5加密后的值	必填	无
nickName	用户昵称	不能超过32个字符	必填	无
phone	手机号码	不能超过16个字符	可选	无
userName	用户名称	不能超过32个字符	必填	无

返回结果参数说明

名称	含义	规则说明
code	结果值	0：成功
1：失败		
2：用户存在		

服务示例
调用接口
http://42.194.128.13/api/reg
参数

{
"email": "472251823@qq.com",
"firstPwd": "e10adc3949ba59abbe56e057f20f883e",
"nickName": "lucky",
"phone": "18612345678",
"userName": "qingfu"
}

返回结果

{
"code": 0
}

3.2 登录

登录，根据用户输入的登录信息，登录进入到后台系统。
请求URL

URL	http://42.194.128.13/api/login
请求方式	POST
HTTP版本	1.1
Content-Type	application/json

请求参数

参数名	含义	规则说明	是否必须	缺省值
pwd	密码	md5加密后的值	必填	无
user	用户名称	不能超过32个字符	必填	无

返回结果参数说明

名称	含义	规则说明
code	结果值	0：成功
1：失败		
token	令牌	每次登录后，生成的token不一样，后续其他接口请求时，需要带上token。

服务示例
调用接口
http://42.194.128.13/api/login
参数
{
"pwd": "e10adc3949ba59abbe56e057f20f883e",
"user": "qingfu"
}
返回结果
{
"code": 0,
"token": "3a58ca22317e637797f8bcad5c047446"
}

参考

redis协议解析参考：https://zhuanlan.zhihu.com/p/139387293
json范例参考：https://dev.to/pintuch/rust-serde-json-by-example-2kkf

mysql:Rust-Sqlx极简教程 (shuzhiduo.com)