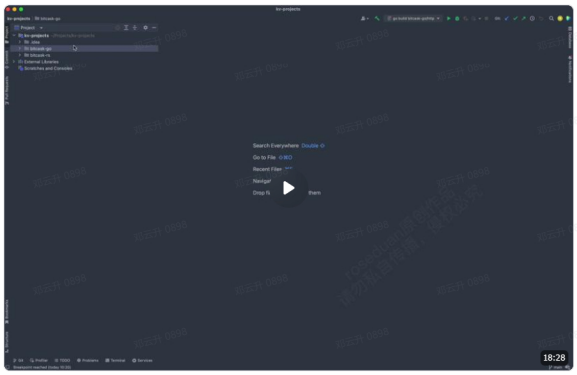
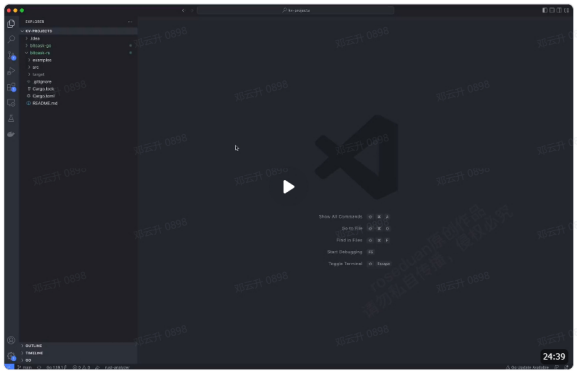
17 从零实现 KV 存储—HTTP 接口

理论讲解

Go 编码



Rust 编码



目前我们实现的 KV 存储引擎，其提供的接口都是内嵌的，也就是说我们可以将其作为一个嵌入式的库在 Go 或者 Rust 的其他应用中进行使用。

在实际的应用场景中，KV 存储引擎往往需要与其他系统进行交互，以便于数据的共享和使用。而 HTTP 协议是一种广泛应用于互联网上的应用层网络协议，通过将数据进行封装和传输，可以实现不同系统之间的数据交换和通信。因此，将 KV 存储引擎与 HTTP 协议进行集成，可以为应用开发和数据管理带来诸多好处。

最大的好处便是，将 KV 存储引擎接入 HTTP 协议可以方便地实现数据的访问和共享。通过使用 HTTP 协议的 GET 和 POST 等请求方式，可以快速地从存储引擎中获取数据、更新数据以及删除数据。这样，不仅可以方便地与其他系统进行数据交换，而且可以在不同的平台和语言之间实现数据的互通。

所以我们可以提供一个 web 服务，让使用者可以进行远程调用，这节课程主要实现的是 HTTP 的接口，大家感兴趣的话也可以去实现其他的接口，例如 RPC。

对于 Go 语言，有很多著名的 web 框架可以来实现这一点，比如 go-zero、echo、gin 等等，为了实现上的简洁，我们课程当中采用了 Go 自带的 HTTP 框架，当然后续如果有其他需求的话，你也可以基于其他的框架来实现，大致的逻辑都是相通的。

对于 Rust 语言，也有很多知名的 web 框架，例如 rocket、actix-web、axum 等，这里有个项目详细比较了 Rust 中的各个 web 框架：https://github.com/flosse/rust-web-framework-comparison，感兴趣的同学可以参考。在课程演示中，我会使用 actix-web 来进行编码，其他的都类似，大家可以根据需要去实现。

我们之前实现了好几个用户层可以调用的接口，这里我以下面的这几个接口作为演示，在外层封装一个 HTTP 访问的接口，对于后续其他的方法也是类似的，大家可以按照同样的方式进行添加。

- **Put**
- **Get**
- **Delete**
- **ListKeys**
- **Stat**

添加 HTTP 接口的大致逻辑，一般是定义一个处理的方法，接受并解析参数，然后调用我们的存储引擎操作数据的接口，最后将返回值按照 HTTP 的格式返回。

然后在启动 HTTP 服务的时候，可以注册我们实现的处理方法。

Go

```go
func main() {
    // 注册 http 的接口
    http.HandleFunc("/bitcask/put", handlePut)
    http.HandleFunc("/bitcask/get", handleGet)
    http.HandleFunc("/bitcask/delete", handleDelete)
    http.HandleFunc("/bitcask/stat", handleStat)
    http.HandleFunc("/bitcask/listkeys", handleListKeys)

    // 启动 HTTP 服务
    http.ListenAndServe("localhost:8080", nil)
}
```

Rust

```rust
#[actix_web::main]
async fn main() -> std::io::Result<()> {
    // 启动 Engine
    let mut opts = Options::default();
    opts.dir_path = PathBuf::from("/tmp/bitcask-rs-http");
    let eng = Arc::new(Engine::open(opts).unwrap());

    // 启动 http 服务
    HttpServer::new(move || {
        App::new().app_data(web::Data::new(eng.clone())).service(
            Scope::new("/bitcask")
                .service(put_handler)
                .service(get_handler)
                .service(delete_handler)
                .service(listkeys_handler)
                .service(stat_handler),
        )
    })
    .bind(("127.0.0.1", 8080))?
    .run()
    .await
}
```