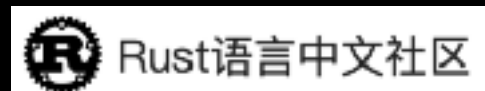


# 初探Rust微服务架构

## 让Rust能尽快落地到生产环境

苏林



## 分享内容

- 回顾上次分享内容
- 为什么需要微服务架构
- gRPC是什么
- Rust中如何使用gRPC

## 期望公开课达到的目的

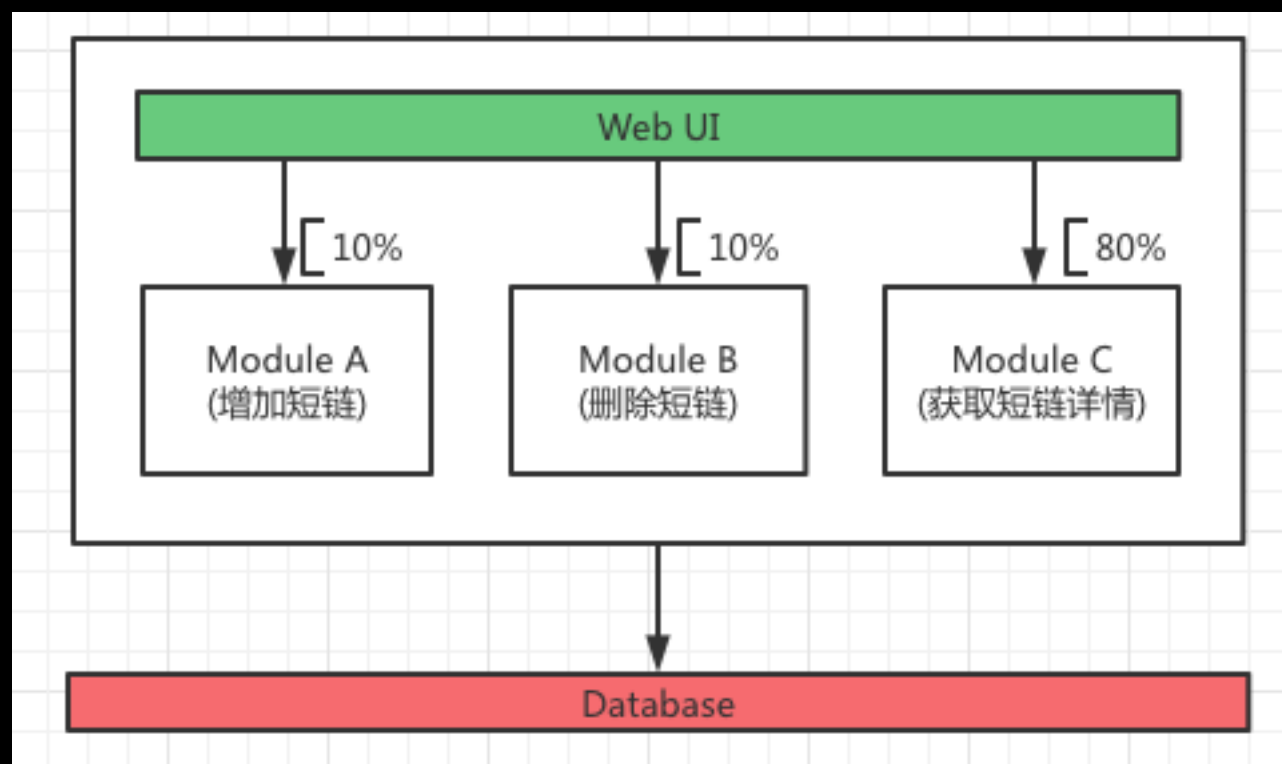
- 这个实战也是连续做了3次公开课了, 核心目的希望大家将Rust落地生产环境, 提升大家学习Rust的兴趣, 告别学习Rust始终在一个main文件里面写一些demo
- 认识微服务架构
- 掌握Rust在微服务架构中的使用.

## 回顾一下上次公开课的内容

- 给大家展示一些之前的代码

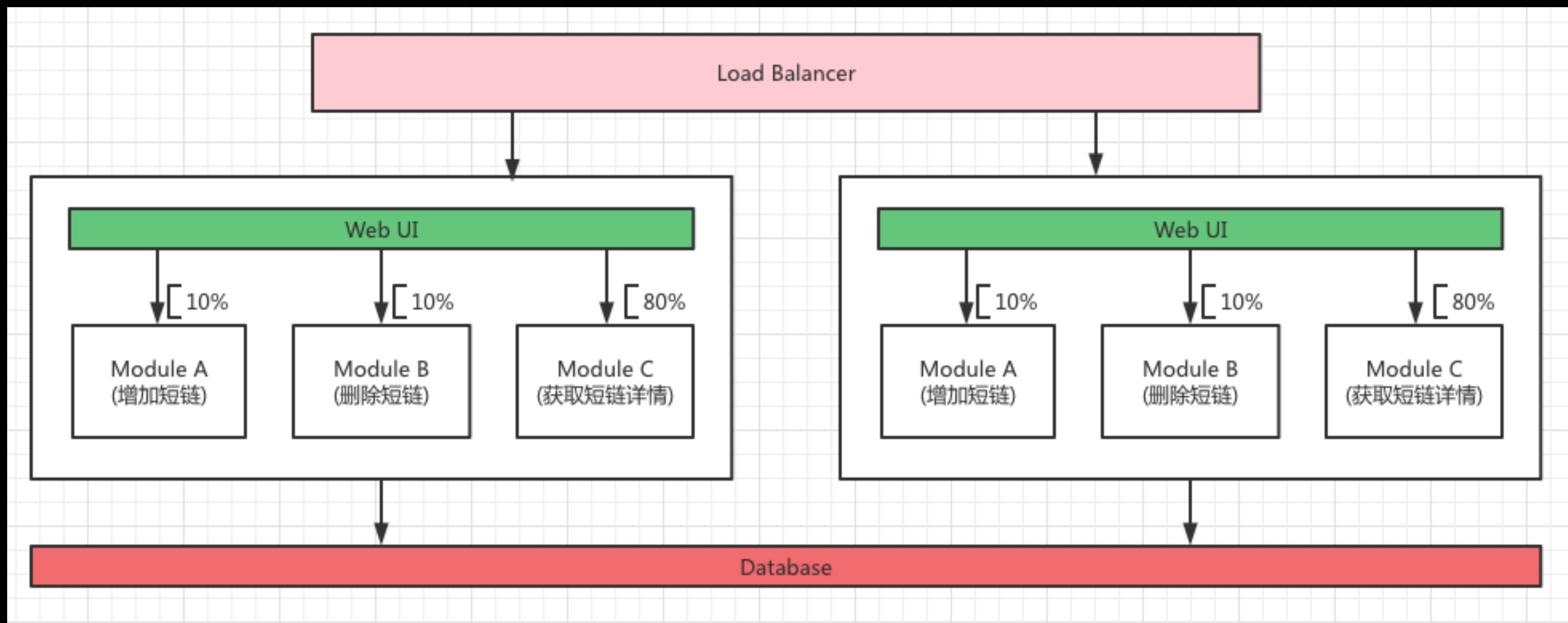
# 为什么需要微服务架构

## 传统应用架构的问题



# 为什么需要微服务架构

如何解决传统应用架构的问题



# 为什么需要微服务架构

传统应用架构还有哪些问题

- 1、系统资源浪费
- 2、部署效率太低
- 3、技术选型单一

# 为什么需要微服务架构

## 微服务架构概念

- 1、根据业务模块划分服务种类.
- 2、每个服务可独立部署且相互隔离.
- 3、通过轻量级API调用服务.
- 4、服务需保证良好的高可用性



# 为什么需要微服务架构

什么是RPC?

RPC框架技术选型

Google 的gRPC

Facebook 的 Thrift

阿里的 Dubbo

# gRPC是什么

## gRPC HTTP2.0 Protobuf

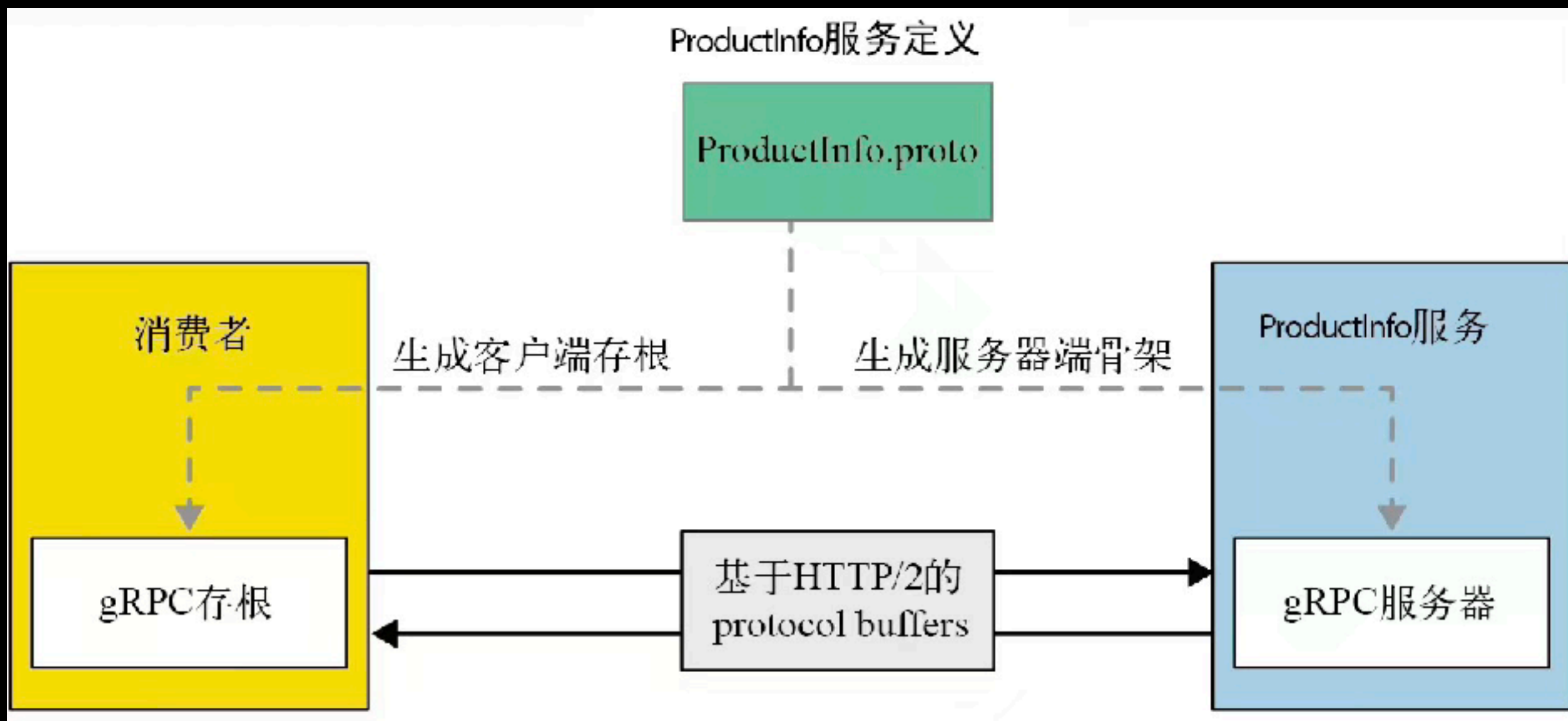
gRPC是一款RPC框架，也是本系列的主角，在性能和版本兼容上做了提升和让步：

Protobuf进行数据编码，提高数据压缩率

使用HTTP2.0弥补了HTTP1.1的不足

同样在调用方和服务方使用协议约定文件，提供参数可选，为版本兼容留下缓冲空间

## 使用gRPC实现微服务的实际场景.

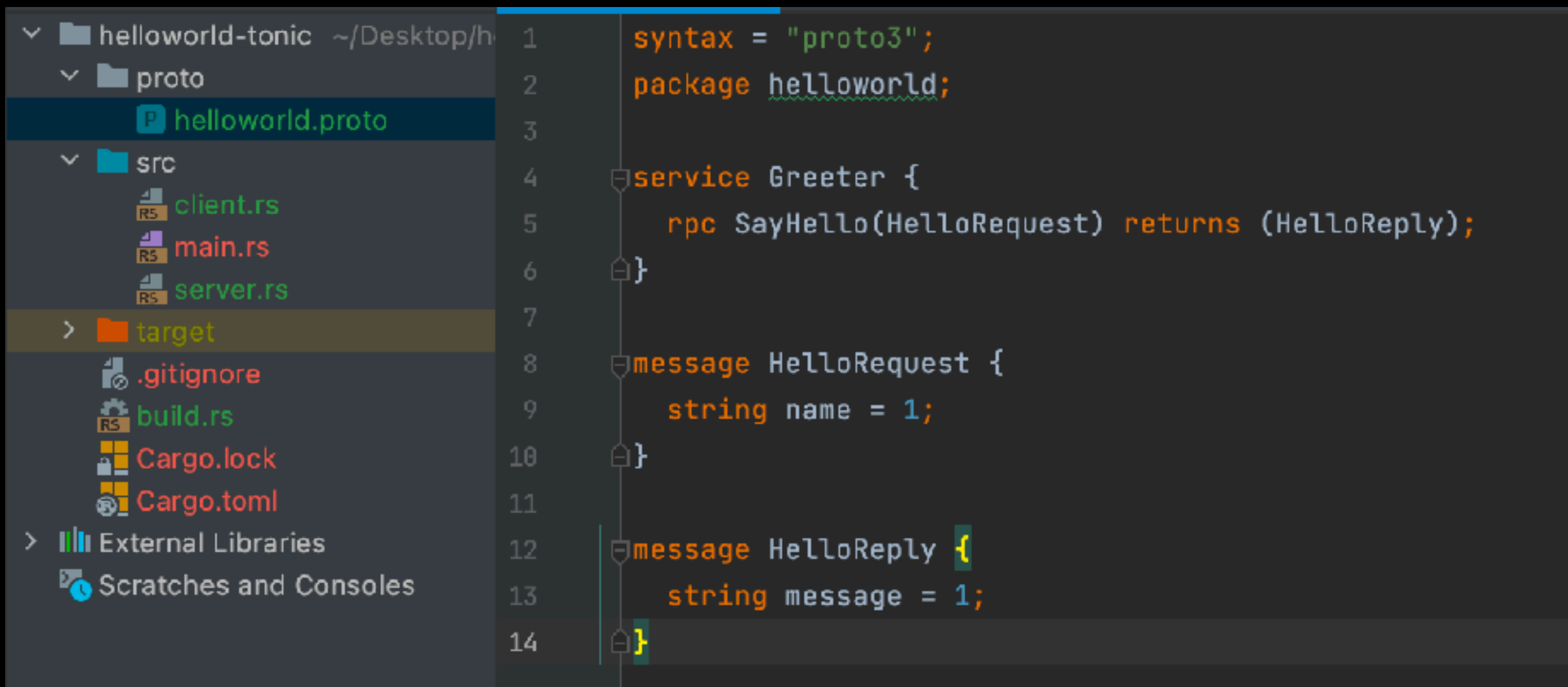


# gRPC通信的细节(以helloworld为例)

- 1、服务定义
- 2、gRPC服务器端
- 3、gRPC客户端

# Rust中如何使用gRPC(helloworld为例)

## 1、服务定义



The screenshot displays an IDE interface with a project explorer on the left and a code editor on the right. The project explorer shows the following structure:

- helloworld- tonic ~/Desktop/h...
  - proto
    - helloworld.proto
  - src
    - client.rs
    - main.rs
    - server.rs
  - target
  - .gitignore
  - build.rs
  - Cargo.lock
  - Cargo.toml
  - External Libraries
  - Scratches and Consoles

The code editor shows the content of `helloworld.proto`:

```
1 syntax = "proto3";
2 package helloworld;
3
4 service Greeter {
5     rpc SayHello(HelloRequest) returns (HelloReply);
6 }
7
8 message HelloRequest {
9     string name = 1;
10 }
11
12 message HelloReply {
13     string message = 1;
14 }
```

# Rust中如何使用gRPC(helloworld为例)

## 2、服务端实现

```
1 use tonic::{transport::Server, Request, Response, Status};
2
3 use helloworld::greeter_server::{Greeter, GreeterServer};
4 use helloworld::{HelloReply, HelloRequest};
5
6 pub mod helloworld {
7     tonic::include_proto!("helloworld");
8 }
9
10 #[derive(Debug, Default)]
11 pub struct MyGreeter {}
12
13 #[tonic::async_trait]
14 impl Greeter for MyGreeter {
15     async fn say_hello(
16         &self,
17         request: Request<HelloRequest>,
18     ) -> Result<Response<HelloReply>, Status> {
19         println!("Got a request: {:?}", request);
20
21         let reply = helloworld::HelloReply {
22             message: format!("Hello {}!", request.into_inner().name).into(),
23         };
24
25         Ok(Response::new(reply))
26     }
27 }
28
29 #[tokio::main]
30 async fn main() -> Result<(), Box<dyn std::error::Error>> {
31     let addr = "[::1]:50051".parse()?;
32     let greeter: MyGreeter = MyGreeter::default();
33
34     Server::builder()
35         .add_service(GreeterServer::new(greeter))
36         .serve(addr)
37         .await?;
38
39     Ok(())
40 }
```

# Rust中如何使用gRPC(helloworld为例)

## 3、客户端实现

```
1 use hello_world::greeter_client::GreeterClient;
2 use hello_world::HelloRequest;
3 pub mod hello_world {
4     tonic::include_proto!("helloworld");
5 }
6
7 #[tokio::main]
8 async fn main() -> Result<(), Box<dyn std::error::Error>> {
9     let mut client = GreeterClient::connect("http://[::1]:50051").await?;
10
11     let request = tonic::Request::new(HelloRequest {
12         name: "Tonic".into(),
13     });
14
15     let response = client.say_hello(request).await?;
16
17     println!("RESPONSE={:?}", response);
18
19     Ok(())
20 }
```

# QA环节

加群一起交流Rust & Datafuse



Datafuse



Rust语言中文社区

