



# RUST CHINA CONF 2020

首届中国 Rust 开发者大会

2020.12.26-27 深圳



# **Rust China Conf 2020**

**Shenzhen, China**

[2020conf.rustcc.cn](http://2020conf.rustcc.cn)

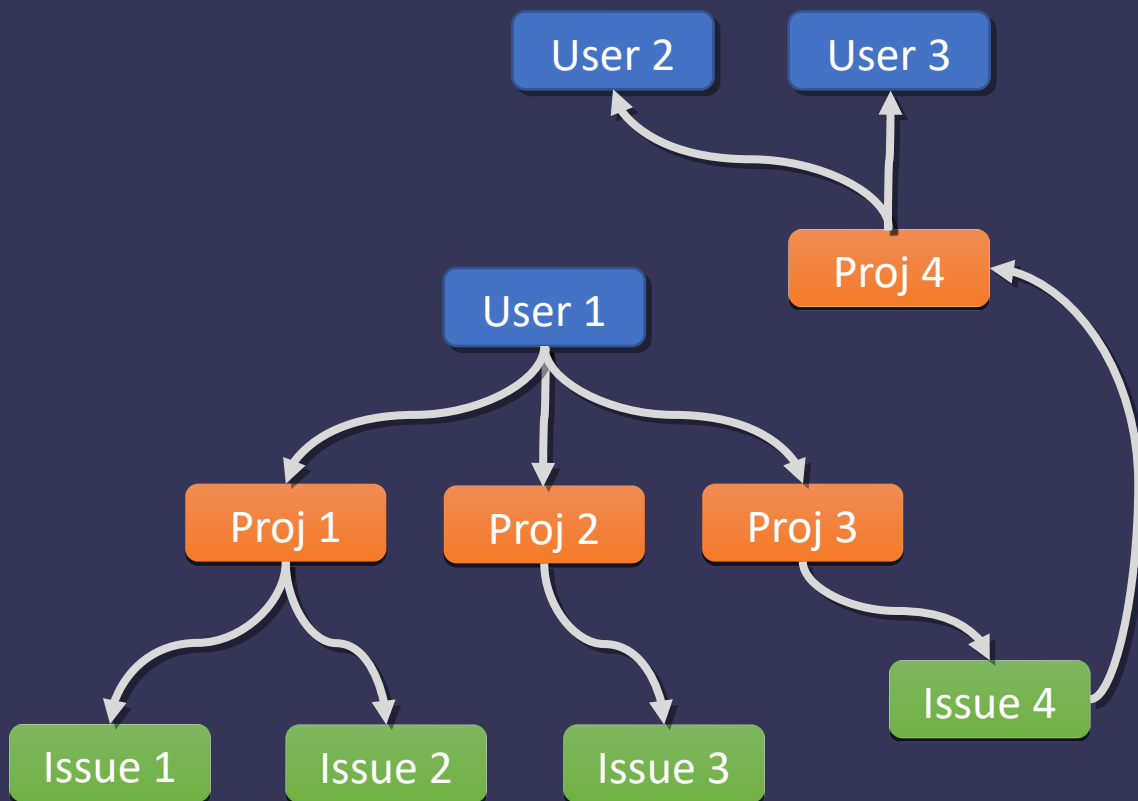


# Async-graphql Rust 语言实现的异步 GraphQL 服务端 库

孙黎 (老油条)



# GraphQL 是什么



```
user(id: 1) {  
  projects {  
    issues {  
      projects {  
        users {  
          ...  
        }  
      }  
    }  
  }  
}
```



# 发起开源项目 Async-graphql 的原因

- 完全支持 Async
- 完全符合 GraphQL 规范
- 类型安全, 高性能
- 无 DSL, 对 Rustfmt 友好
- 支持 Apollo Federation 实现基于 GraphQL 的微服务



# Async-graphql 现状

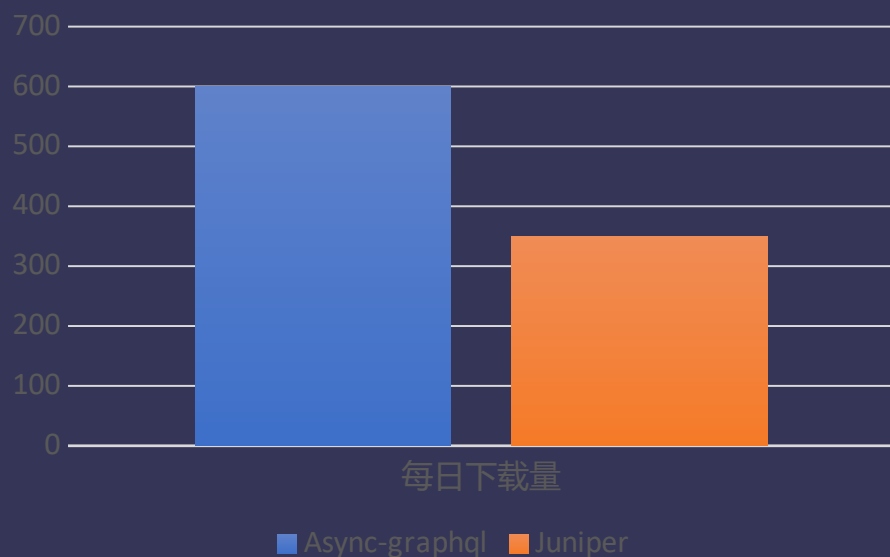
Stargazers: 1.1k

Contributors: 44

Issues: 229 closed

PR: 112

每日下载量 ( crates.io )



**Vector.dev** @vectordotdev · Nov 7

Thanks to upstream maintainer sunli829 for helping us with this ``async-graphql`` problem.

A powerful library, and a kind maintainer.

We recommend it for your Rust GraphQL projects. It's been a good experience.



**Vector.dev** @vectordotdev · Aug 27

What a pleasure to use the **async-graphql** library by sunli829! 😊



# Async-graphql 的实现



# 执行一个查询的步骤

解析



校验



执行



输出





# 核心 Trait

## Type

- 表示一个 GraphQL 类型，提供 `create_type_info` 函数在类型注册表中创建该类型的信息。

## InputType

- 表示一个输入值类型，提供 `parse` 函数解析输入值。

## OutputType

- 表示一个输出值类型，提供 `resolve` 函数根据客户端的查询输出结果。



# 定义 Schema

```
type User {  
    name: String,  
    email: String,  
}  
  
type Query {  
    user(id: Int): User,  
}  
  
schema {  
    query: Query  
}
```

```
#[derive(SimpleObject)]  
struct User {  
    name: String,  
    email: String,  
}  
  
struct Query;  
  
#[Object]  
impl Query {  
    async fn user(&self, id: u64) -> Result<User> {  
        // 从数据库中加载指定 id 的 User  
    }  
}
```



# 执行阶段

```
{  
  user(id: 10) {  
    name  
    email  
  }  
}
```

User::name()

String::resolve()

Query::resolve()

Query::user(id: 10)

User::resolve()

User::email()

String::resolve()



# Type 类型

```
trait Type {  
    fn type_name() -> String;  
  
    fn create_type_info(registry: &mut registry::Registry) -> String;  
}
```

## Rust 类型

## GraphQL 类型

i32

Int!

Vec<i32>

[Int!]!

Vec<Option<i32>>

[Int]!

## Rust 类型

Option<i32>

Option<Vec<i32>>

Option<Vec<Option<i32>>>

## GraphQL 类型

Int

[Int!]

[Int]



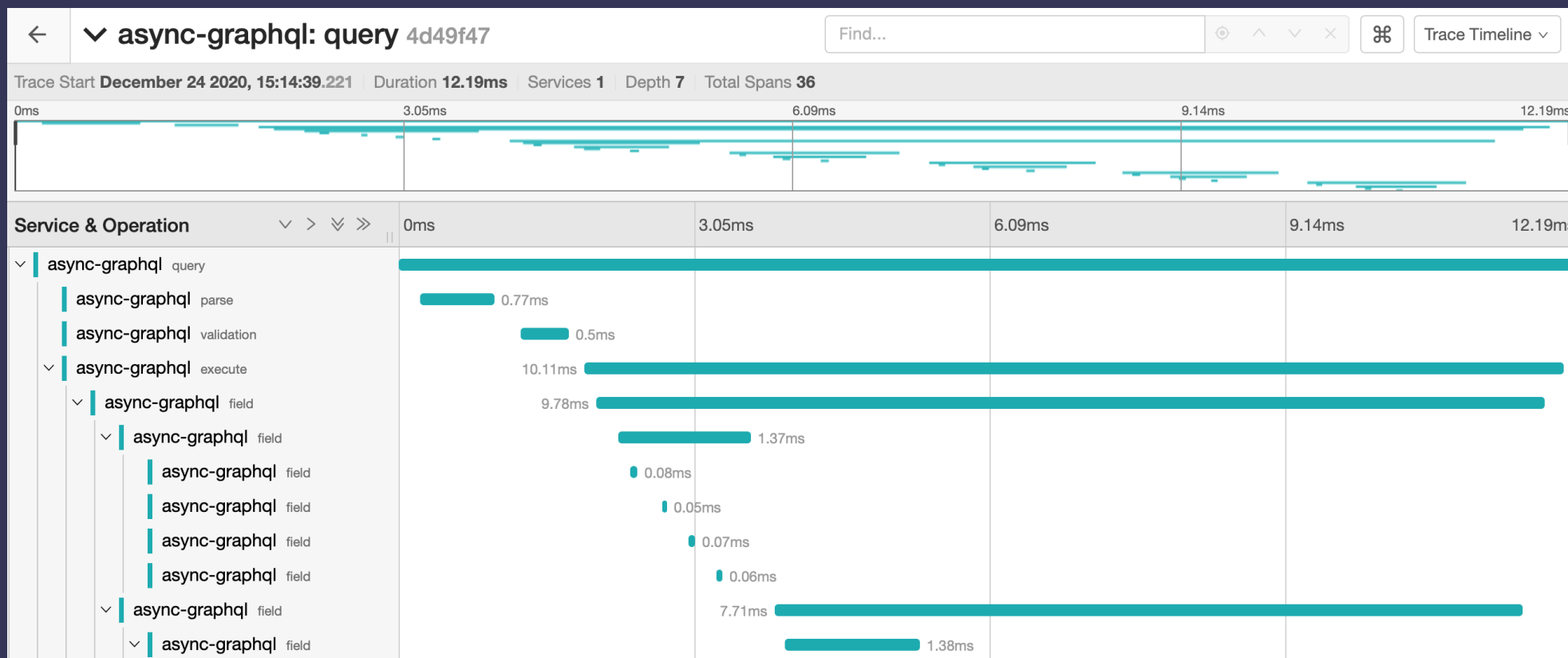
# InputType 和 OutputType 类型

```
trait InputType: Type + Sized {  
    fn parse(value: Option<Value>) -> InputValueResult<Self>;  
}  
  
trait OutputType: Type {  
    async fn resolve(&self, ctx: &Context) -> ServerResult<Value>;  
}
```



# Tracing

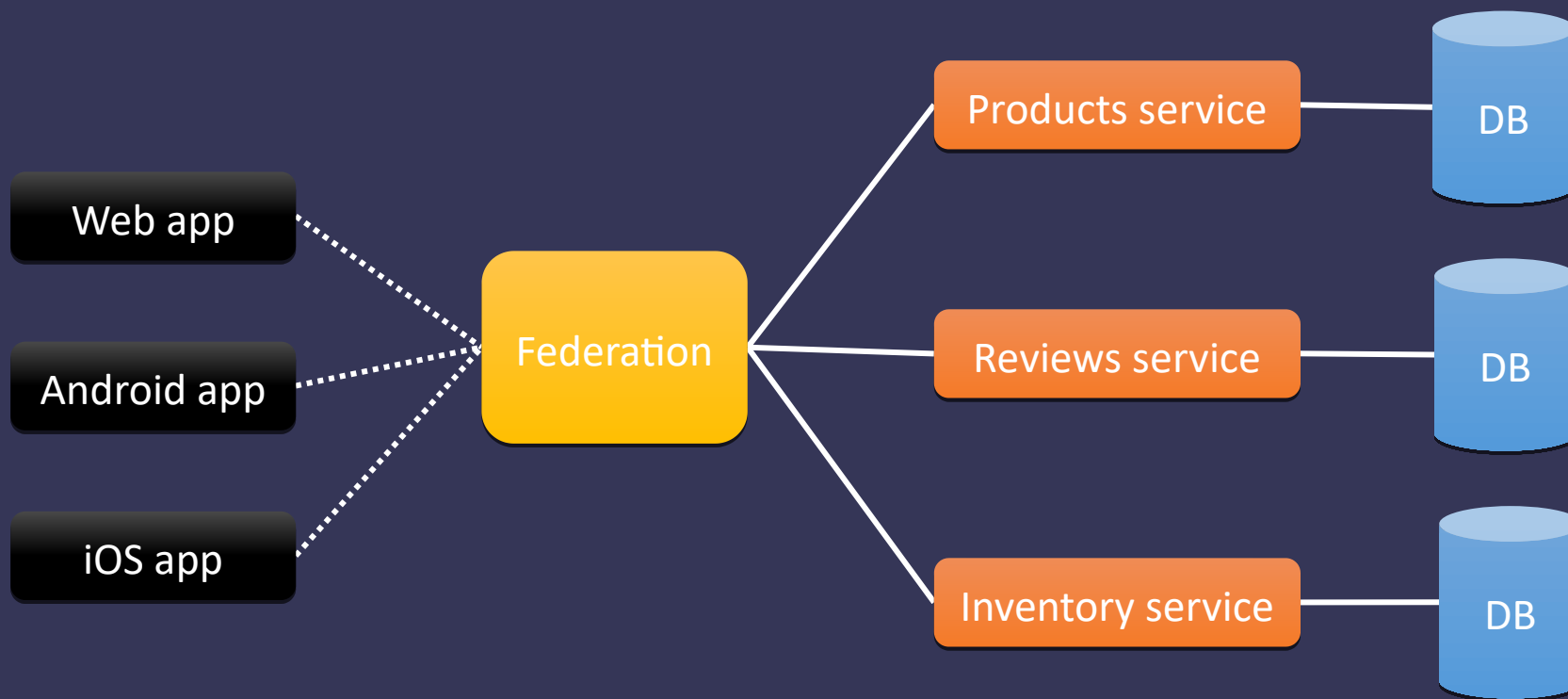
使用 Tracing 库来跟踪 GQL 查询各个阶段的执行情况





# Apollo Federation

使用 Federation 网关来实现基于 GraphQL 的微服务





# 用 Darling 简化过程宏的编写

```
#[derive(FromDeriveInput)]
#[darling(attributes(graphql), forward_attrs(doc))]
pub struct Enum {
    pub ident: Ident,
    pub generics: Generics,
    pub attrs: Vec<Attribute>,
    pub data: Data<EnumItem, Ignored>,

    #[darling(default)]
    pub internal: bool,
    #[darling(default)]
    pub name: Option<String>,
    #[darling(default)]
    pub rename_items: Option<RenameRule>,
    #[darling(default)]
    pub remote: Option<String>,
}
```





<https://github.com/async-graphql/async-graphql>

# 谢谢大家



# RUST CHINA CONF 2020

首届中国 Rust 开发者大会

2020.12.26-27 深圳