

第十六届 D2 前端技术论坛

Node.js Web 框架再进化 面向前端与未来标准

淘系技术部 - Node.js 架构组 - 刘子健（繁易）



刘子健 / 繁易 / Lxxyx



淘系技术部 - 前端技术 - Node.js 架构 - 前端技术专家



Node.js & Serverless & Web Framework



Midway.js Core Member



Node.js Core Collaborator



Alibaba Delegate

Contents

目录

01 Node.js & Web 框架简述

02 Midway - 面向前端的框架演进之路

03 未来 - 面向标准 & 规划

01

Node.js & Web 框架简述



[HOME](#) | [ABOUT](#) | [DOWNLOADS](#) | [DOCS](#) | [GET INVOLVED](#) | [SECURITY](#) | [CERTIFICATION](#) | [NEWS](#)

Node.js® is a JavaScript runtime built on **Chrome's V8 JavaScript engine**.

Node.js 可用于

编写 CLI

处理数据

Restful Api

页面渲染

.....

Web 框架功能

- 现代 Web 开发离不开 Web 框架
- Web 框架提供了高效开发 Web 应用的方式
- Web 框架存在**适用场景与规则约束**



Restful API



数据库 CRUD

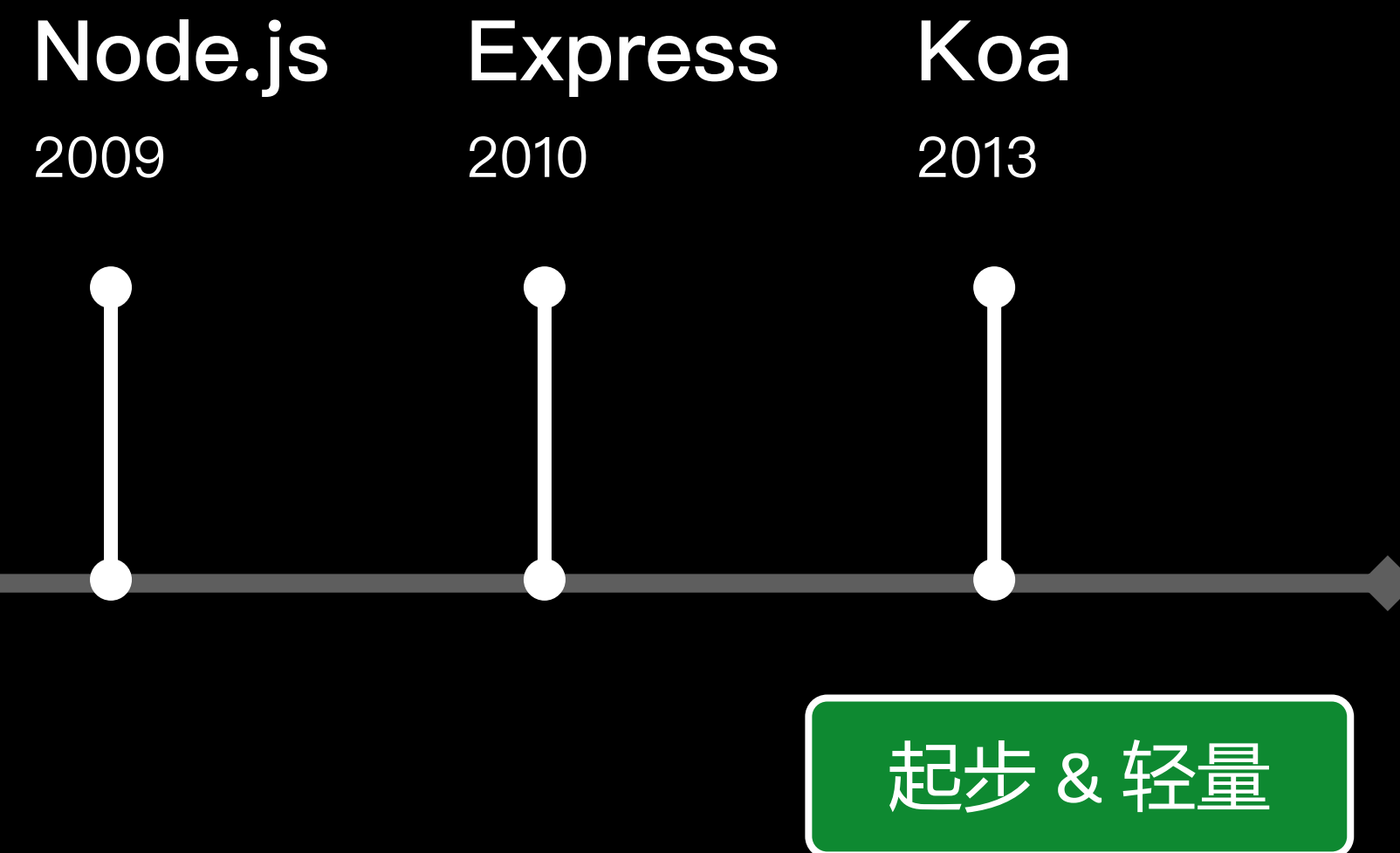


页面渲染



身份校验

Node.js Web 框架的三个阶段



- Node.js 刚起步
- 前端工程师尝鲜
- 验证 Node.js Web 场景可行性
- 主打：轻量 & 极简

Express



```
1 const app = require('express')()  
2  
3 app.get('/', (req, res) => {  
4   res.send('Hello World!')  
5 })  
6  
7 app.listen(3000)
```

Koa



```
1 const Koa = require('koa');  
2 const app = new Koa();  
3  
4 app.use(async ctx => {  
5   ctx.body = 'Hello World';  
6 });  
7  
8 app.listen(3000);
```

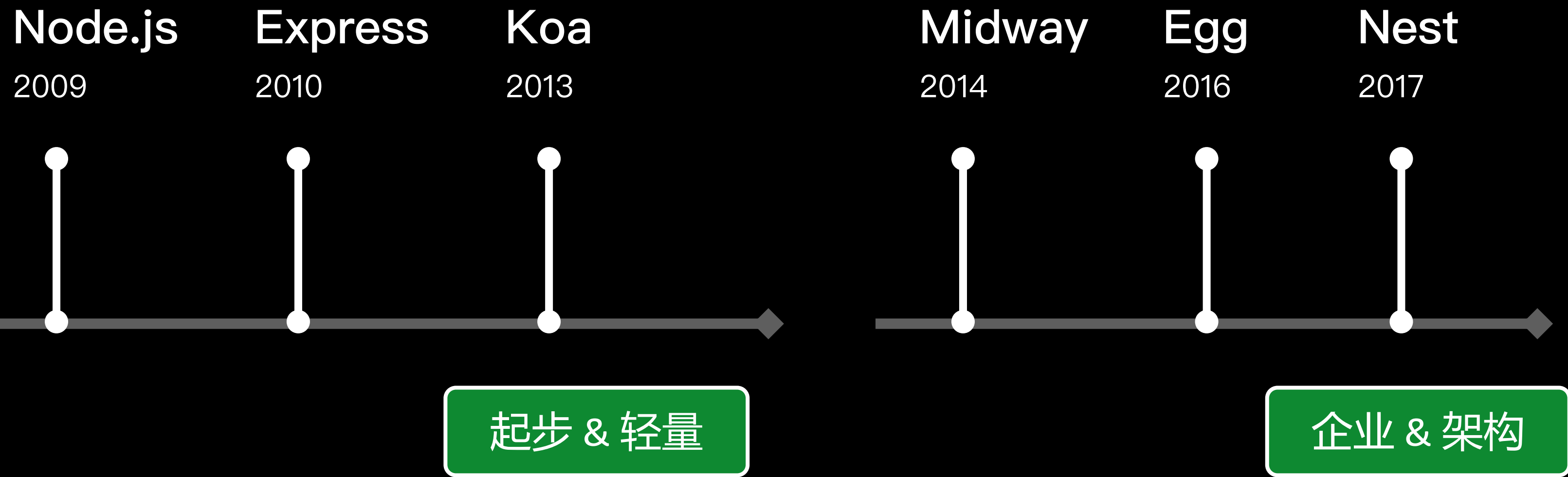
优点

- 极简 & 易学
- 易于集成
 - Express(Nest / Webpack)
 - Koa(Egg / Midway)
- 生态繁荣 & 久经考验

缺陷

- 缺乏规范 & 最佳实践
- 不利于团队协作 & 大规模开发
- Express 年久失修

Node.js Web 框架的三个阶段



- Node.js 刚起步
- 前端工程师尝鲜
- 验证 Node.js Web 场景可行性
- 主打：轻量 & 极简

- Node.js 规模化落地
- 专业 Node.js 工程师出现
- 主打
 - 企业级框架 & 架构
 - 规模化 & 团队协作

基础框架



Express

Fastify



Koa



Koa

Express

Egg

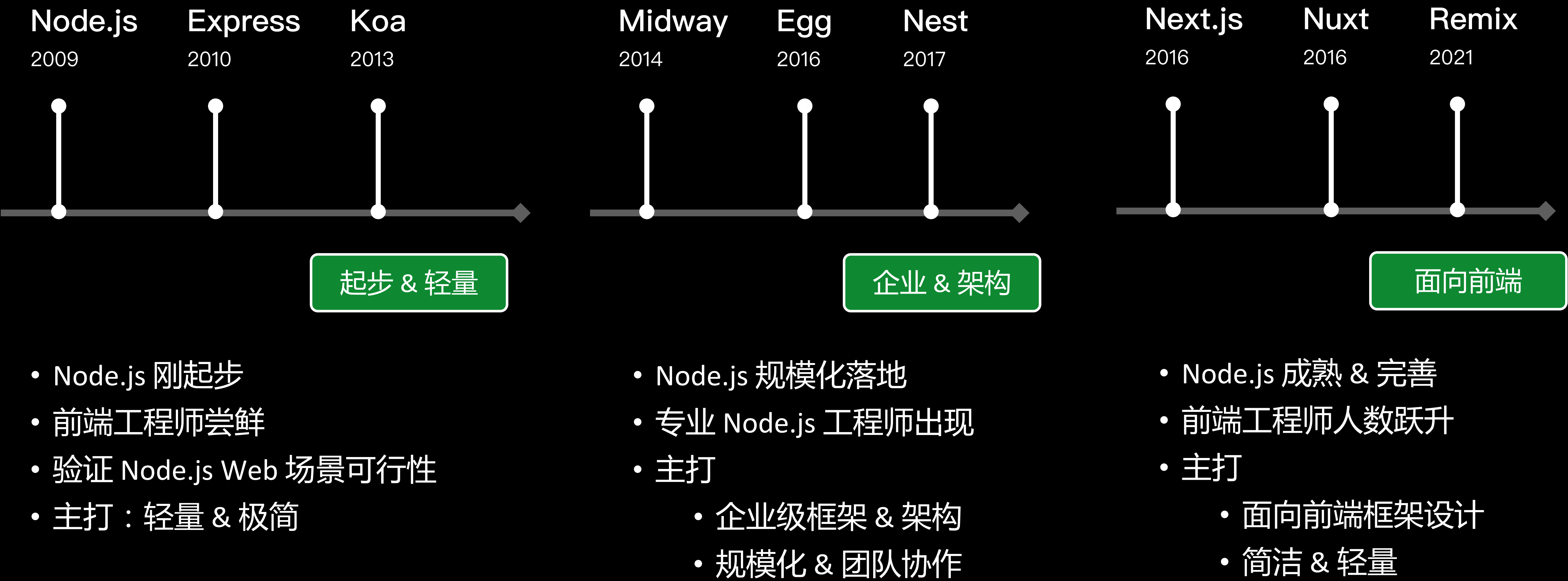
优点

- 大而全，功能完善
- 规范 & 最佳实践明确，易于团队协作
- 社区生态活跃

缺陷

- 大而全，上手成本高
- 限制多，较难拓展

Node.js Web 框架的三个阶段



NEXT.js



```
1 export default (req, res) => {  
2   res  
3     .status(200)  
4     .json({ name: 'John Doe' })  
5 }
```

Nuxt.js



```
1 export default async (req, res) => {  
2   await someAsyncFunction()  
3  
4   return {  
5     someData: true  
6   }  
7 }
```

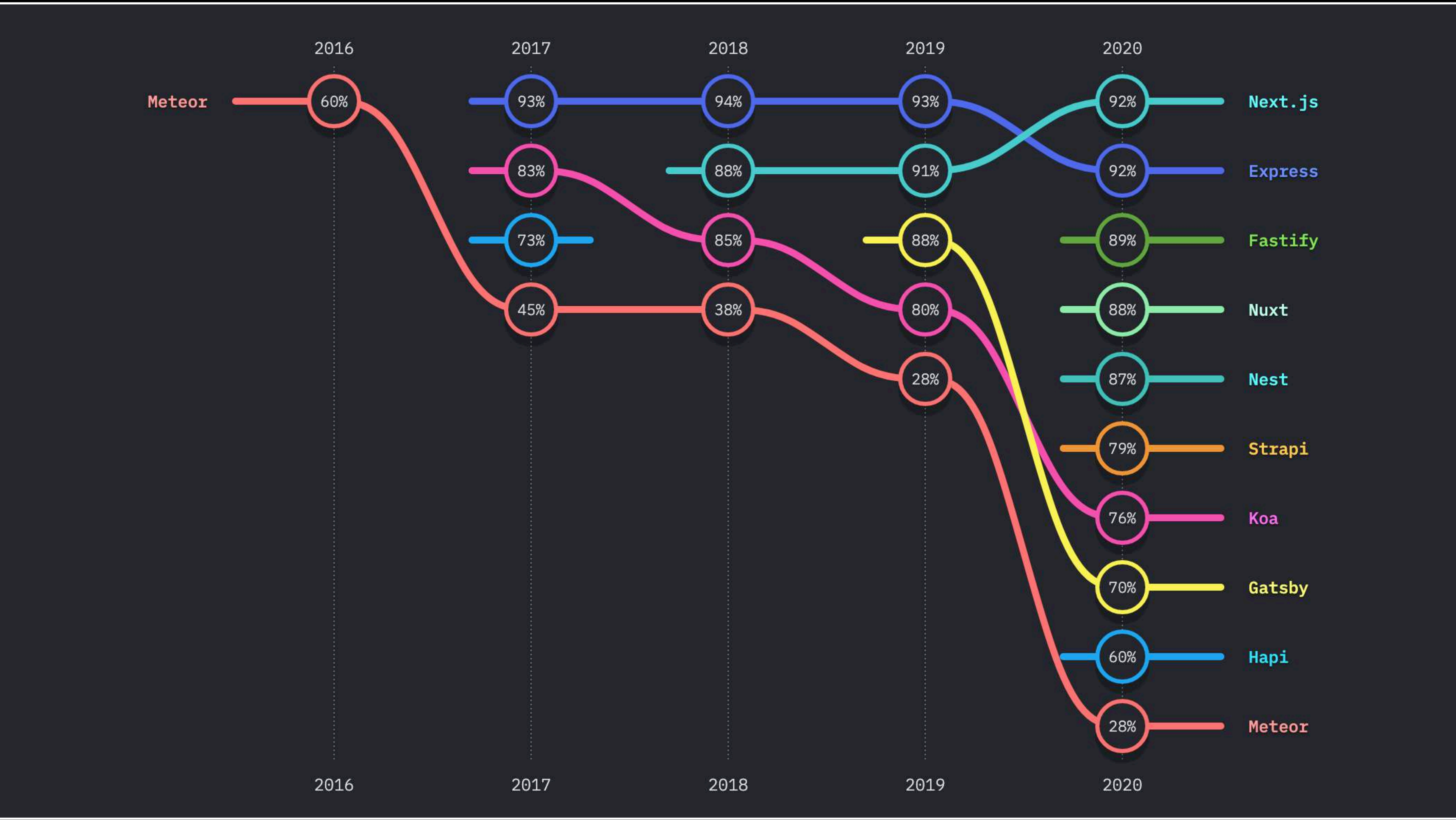
优点

- For 前端，全栈开发
- 简单易学
- 支持 Serverless 部署

缺陷

- 后端功能弱
- 自定义拓展困难
- 强依赖于平台支持

Node.js Web 框架满意度调研 (stateofjs 2020)



Next.js 登顶 & Express.js 仍然备受关注


- Node.js Web 框架迭代与前端行业发展密切相关
- 前端应用场景多于纯 Node.js 后端场景
- 面向前端设计的全栈框架兴起，Node.js 用法回归简洁 & 轻量

02

Midway - 面向前端的框架演进之路

Midway

Node.js Framework For "Fullstack"

[Documentation](#)[Github](#) Star

4,917



始于 2014



7 个大版本



2018 年正式开源

Midway @ 2018 - 从企业级起步



TypeScript

静态类型

多人协作



IoC

复杂架构

面向接口编程



Egg

统一框架

复用生态

```
1 import {
2   Inject,
3   Controller,
4   Get,
5   Provide,
6   Query,
7 } from '@midwayjs/decorator';
8 import { UserService } from '../service/user';
9
10 @Provide()
11 @Controller('/api/user')
12 export class APIController {
13   @Inject()
14   userService: UserService;
15
16   @Get('/')
17   async getUser(@Query('id') uid) {
18     const user = await this.userService.getUser(
19       uid
20     );
21     return {
22       success: true,
23       message: 'OK',
24       data: user,
25     };
26   }
27 }
```

Midway Demo



集团 1600+ Node.js 应用

常年 cpu 利用率 < 10% , 乃至 5%

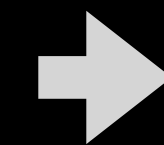
服务器利用率低



前端维护乏力

Docker、进程，限流，日志、跨语言

DevOps 成本高



传统应用的缺点限制了
Node.js 在阿里的进一步发展

前端业务诉求：



“ 后端往大后台下沉，前端往小前台发力，提升生产力。 ”



“ 前端同学希望将中台服务快速组合为各类业务接口，和端侧同步快速交付前台，以更快的响应业务需求变化来帮助业务试错。 ”

需要赋能前端，让云原生给前端降本增效



搭建服务



Serverless



智能化



IDE

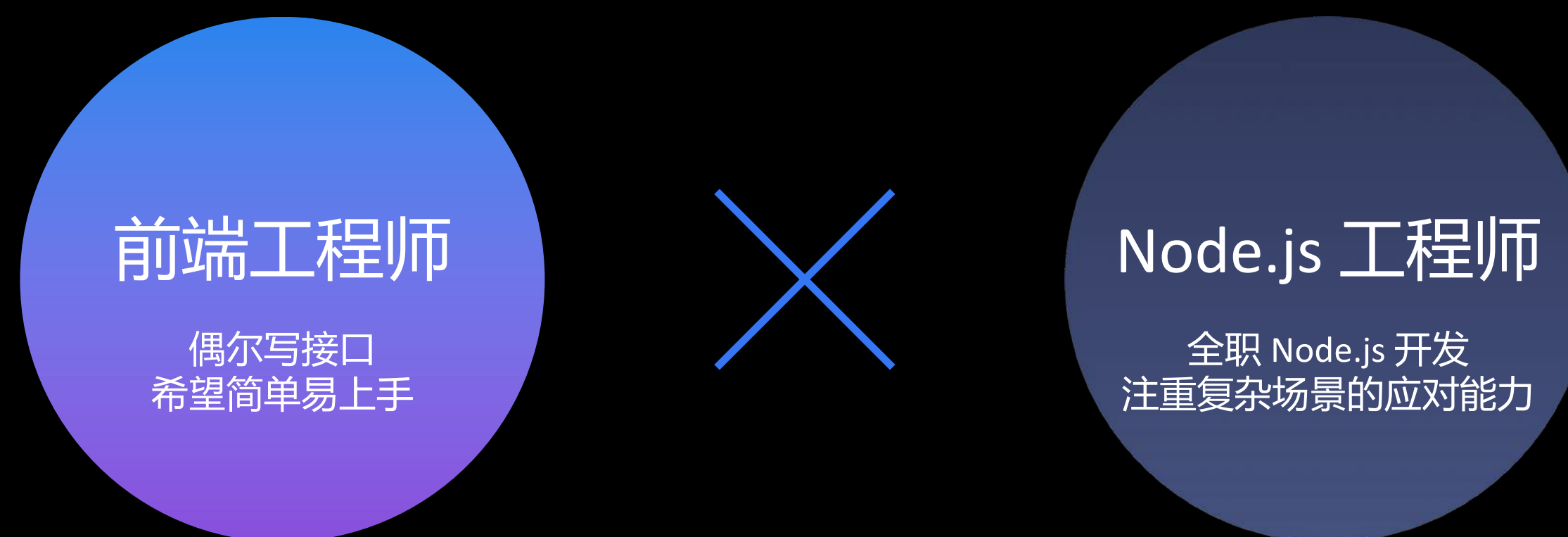
2019-2020 阿里经济体前端委员会四大技术方向

挑战：用户群体割裂



D2 前端技术论坛
D2 FRONTEND TECHNOLOGY FORUM

精心



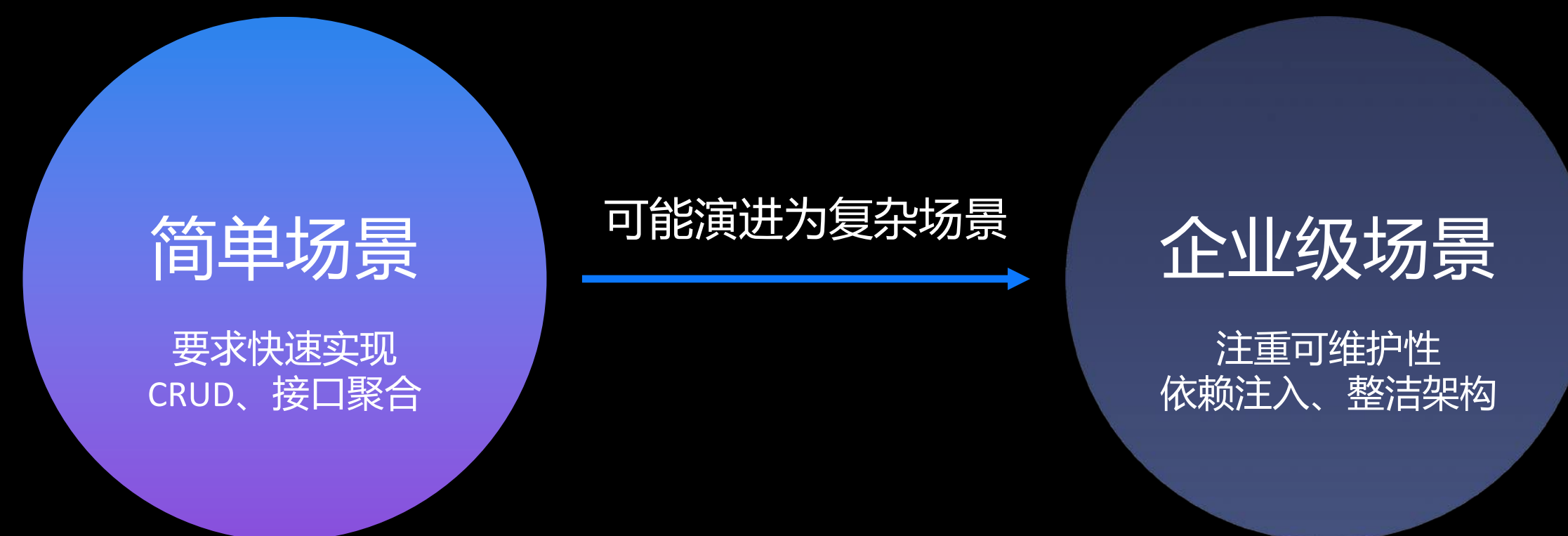
如何在一个框架下服务好两种用户？

挑战：使用场景不同



D2 前端技术论坛
D2 FRONTEND TECHNOLOGY FORUM

精心



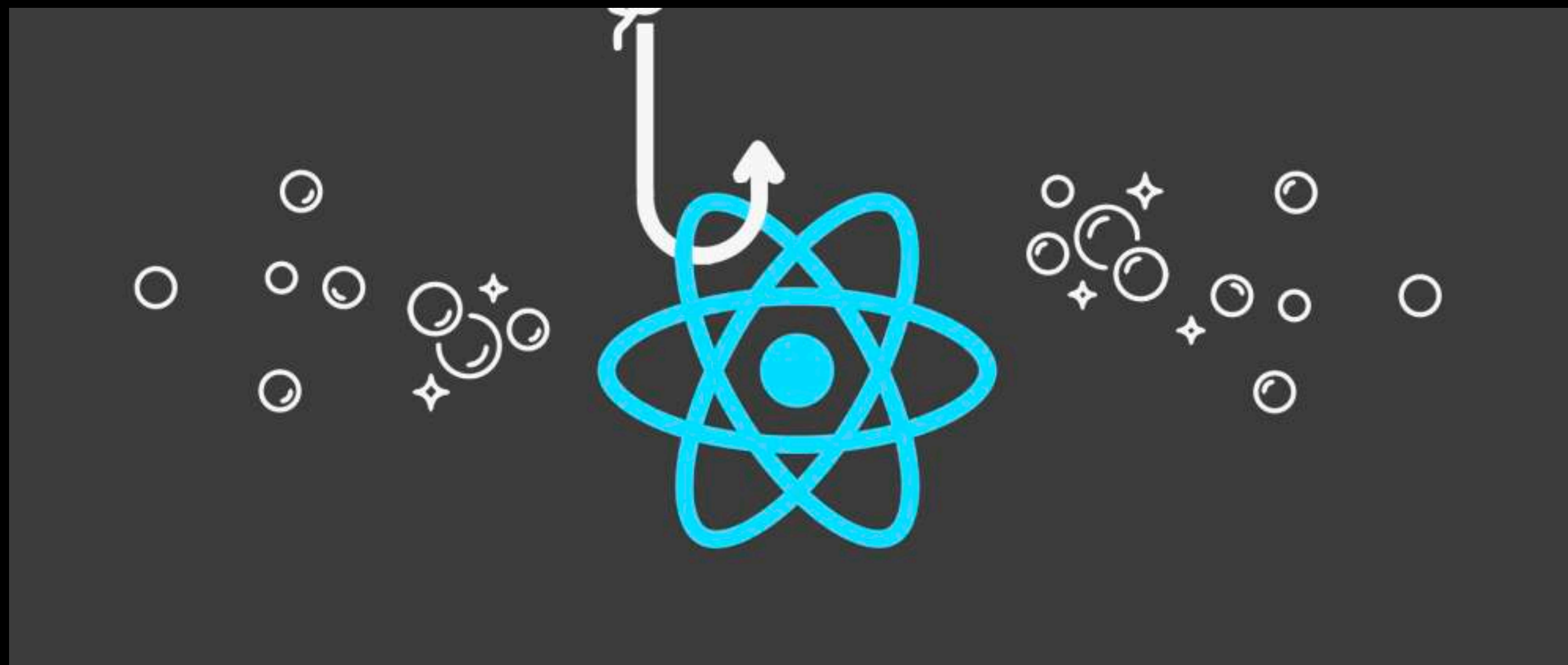
如何在一个框架下支持两种场景？

挑战：前端范式变更



D2 前端技术论坛
D2 FRONTEND TECHNOLOGY FORUM

精心



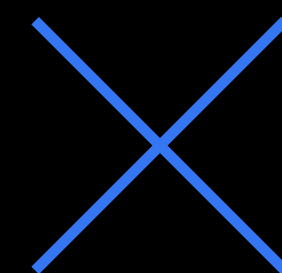
React Hooks



Vue Composition Api

Class Component 向 Function + Hooks 转变

```
1 import { useState } from 'react';
2
3 function Counter() {
4   const [value, setValue] = useState(0);
5
6   function onClick() {
7     setValue(counterValue + 1);
8   }
9
10  return (
11    <div>{value}</div>
12    <button onClick={onClick}>+1</button>
13  </>
14 );
15 }
16 }
```



同一个开发者，前后端思维不同

```
1 import {
2   Controller,
3   Get,
4   Provide
5 } from '@midwayjs/decorator';
6
7 @Provide()
8 @Controller('/')
9 export class HomeController {
10   @Get('/')
11   async home() {
12     return 'Hello Midwayjs!';
13   }
14 }
```

框架有没有可能支持函数式开发，又能与 OOP 并存？



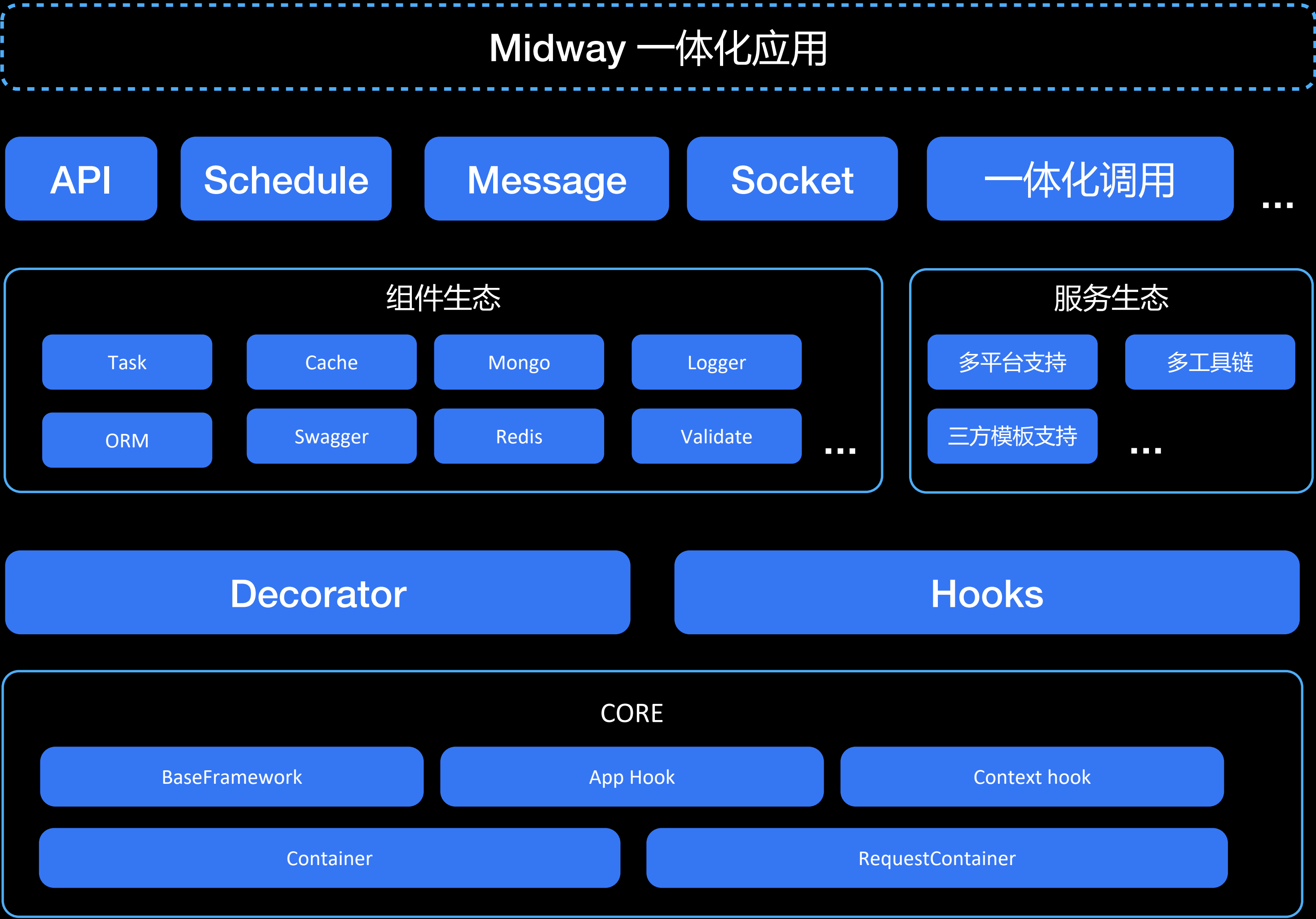
Modern Web

场景化

生态

编码范式

Midway 核心





```
1 export function getUserProfile (id: number) {  
2   const ctx = useContext()  
3   const isAdmin = checkRole(id)  
4  
5   if (!isAdmin) {  
6     return { error: 'no permission' }  
7   }  
8  
9   const profile = await User.find({ id })  
10  return {  
11    data: profile  
12  }  
13 }
```

path: `/api/getUserProfile`

method: `args.length === 0 ? GET : POST`

body: {
 args: [id]
}

return type: `User`

JavaScript 函数即接口 – 统一 & 无协议

基于函数元信息生成接口

Hooks：获取请求上下文



D2 前端技术论坛
D2 FRONTEND TECHNOLOGY FORUM

精心

○ ○ ○

```
import { useContext } from '@midwayjs/hooks'

export async function getQuery () {
  const ctx = useContext()
  return ctx.query
}
```

获取 URL 查询参数

○ ○ ○

```
import { useContext } from '@midwayjs/hooks'
import { useRequest } from './useRequest'

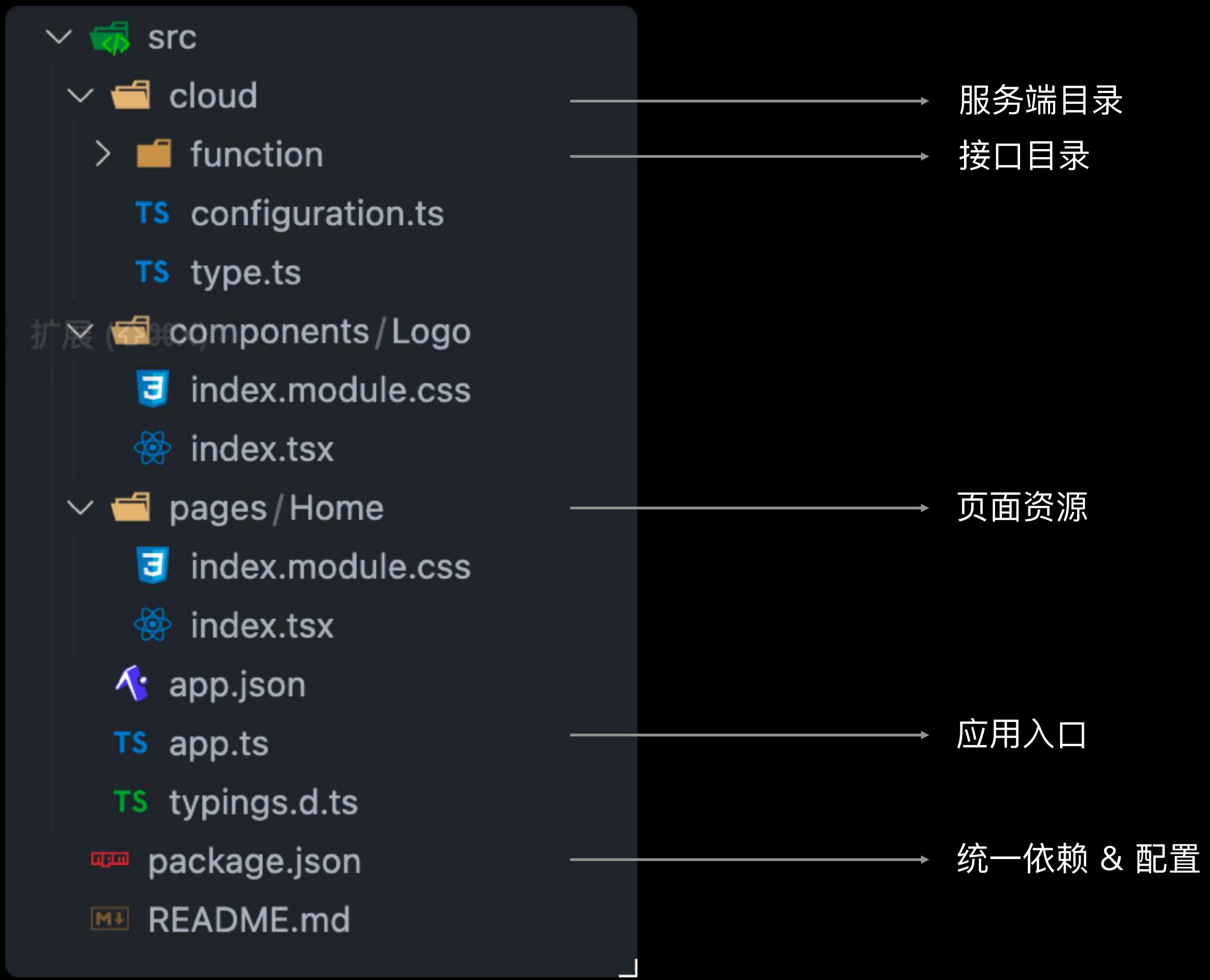
export async function demo() {
  const ctx = useContext()
  const request = useRequest()

  return {
    url: ctx.url,
    header: request.headers
  }
}
```

自定义 Hooks

无需手动传入参数 & 与前端保持一致

Hooks：面向全栈应用设计



统一依赖管理

统一工程配置

共享 src/代码/类型


```
1 import { getUserProfile } from './api'
2 import { useAsyncEffect } from './utils'
3 import { useState } from 'react'
4
5 export default () => {
6   const [name, setName] = useState('')
7   useAsyncEffect(async () => {
8     const profile = await getUserProfile()
9     if (profile.error) {
10       alert(profile.error);
11     }
12     setName(profile.name)
13   }, [])
14
15   return <div>{name}</div>
16 }
```

导入后端函数

1

调用接口函数

2

1

导入函数

2

调用函数并使用返回值

“零” API 调用

渐进式 - 像搭积木一样演进

项目类型	开发方式	拓展组件	触发器	部署平台
纯接口项目	IoC + 装饰器	Config	HTTP	Server
		Middleware	WebSocket	
		ORM		
一体化项目	函数式	Swagger	gRPC	FaaS
		Cache	RabbitMQ	

渐进式 - 像搭积木一样演进

项目类型

开发方式

拓展组件

触发器

部署平台

IoC + 装饰器

Config

HTTP

Middleware

WebSocket

Server

ORM

FaaS

gRPC

Swagger

函数式

Cache

RabbitMQ

纯接口项目

一体化项目

前端一体化应用

渐进式 - 像搭积木一样演进

项目类型

开发方式

拓展组件

触发器

部署平台

IoC + 装饰器

Config

HTTP

Middleware

WebSocket

Server

ORM

FaaS

gRPC

Swagger

函数式

Cache

RabbitMQ

纯接口项目

一体化项目

复杂的企业级应用

渐进式 - 像搭积木一样演进

项目类型

开发方式

拓展组件

触发器

部署平台

IoC + 装饰器

Config

HTTP

Middleware

WebSocket

Server

ORM

FaaS

gRPC

Swagger

函数式

Cache

RabbitMQ

纯接口项目

一体化项目

随着时间流逝
复杂度增加的应用



Built on Midway

Hooks

更快、更具生产力的应用方案

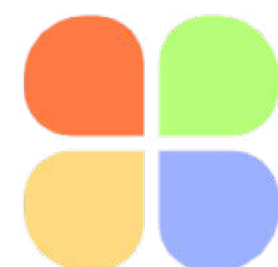
全栈架构

函数式 & Hooks

类型安全



2020.04 发布



2500+ 应用



阿里前端主流模式

- 企业内仍存在简单场景与复杂场景，框架设计应考虑到此问题
- Node.js Web 框架应关注开发者体验，面向前端工程师设计
- 云 + 端的研发模式将成为未来的主流研发模式

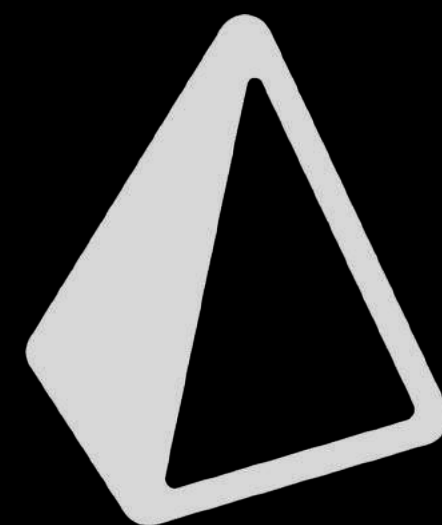


D2 前端技术论坛
D2 FRONTEND TECHNOLOGY FORUM

精心

03

未来 - 面向标准 & 规划



Prisma

MySQL

PostgreSQL

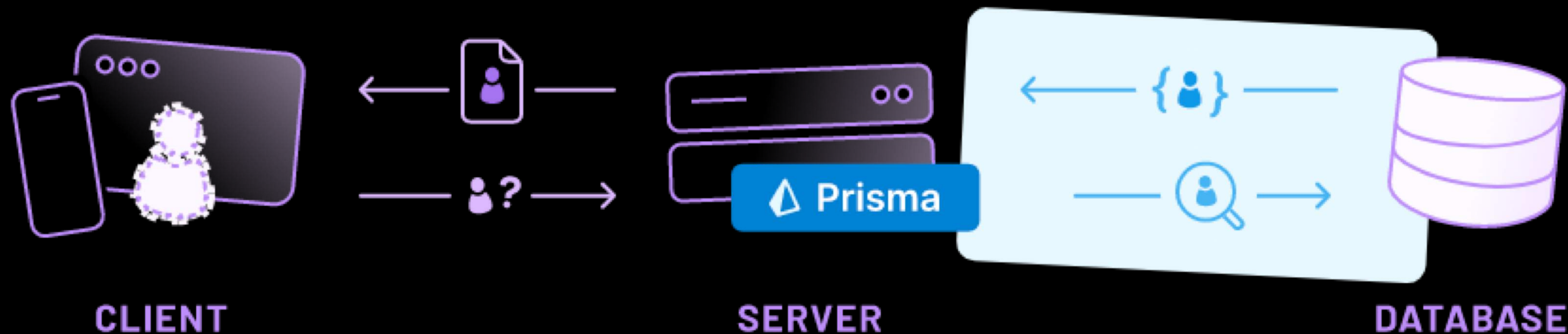
SQLite

```
schema.prisma
1  datasource db {
2    provider = "sqlite"
3    url      = env("DATABASE_URL")
4  }
5
6  generator client {
7    provider = "prisma-client-js"
8  }
9
10 model Post {
11   id      Int @id @default(autoincrement())
12   title   String
13   content String?
14   published Boolean @default(false)
15   author  User? @relation(fields: [authorId], references:
16   authorId Int?
17 }
```

生成 Prisma Client

```
index.tsx
1  await prisma.user
2    .findUnique({
3    where: { email: 'ada@prisma.io' }
4  })
5  .posts({
6    where: {
7      title: {
8        |
9      }
10   }
11  })
```

- contains
- endsWith
- equals
- gt
- gte



从前端到后端再到数据库的全链路类型安全方案


```
const countBlock = module {
  let i = 0;

  export function count() {
    i++;
    return i;
  }
};

const uppercaseBlock = module {
  export function uppercase(string) {
    return string.toUpperCase();
  }
};

const { count } = await import(countBlock);
const { uppercase } = await import(uppercaseBlock);

console.log(count()); // 1
console.log(uppercase("daniel")); // "DANIEL"
```

JS Module Blocks(Stage 2)

```
// filename: app.js
module "#count" {
  let i = 0;

  export function count() {
    i++;
    return i;
  }
}

module "#uppercase" {
  export function uppercase(string) {
    return string.toUpperCase();
  }
}

import { count } from "#count";
import { uppercase } from "#uppercase";

console.log(count()); // 1
console.log(uppercase("daniel")); // "DANIEL"
```

JS Module Fragments(Stage 1)

展望：云端融合

- 与前端委员会标准化小组推进 TC39 提案
- 反馈场景 & 谋求推进至 Stage 2

Github

[tc39/proposal-module-fragments/issues/14](https://github.com/tc39/proposal-module-fragments/issues/14)

```
1 export module server {
2   import { prisma } from './prisma'
3
4   // get articles from database
5   export async function getArticles (type: string) {
6     const articles = await prisma.articles.find({
7       where: { type }
8     })
9     return articles
10  }
11 }
12
13 export module ssr {
14   import { useContext } from '@midwayjs/hooks'
15
16   // fetch github stars for server side rendering
17   export async function getServerSideProps () {
18     const ctx = useContext()
19     const stars = await fetchGithubStars(ctx.query.name)
20     return props: { stars }
21   }
22 }
23
24 export module client {
25   import { useFetch, PromiseType } from './utils'
26   import { getArticles } from server
27   import { getServerSideProps } from ssr
28
29   type Props = PromiseType<typeof getServerSideProps>
30
31   export default function App (props: Props) {
32     const { data } = useFetch(getArticles)
33     return <h1>Stars: {props.stars} Articles: {data}</h1>
34   }
35 }
```


让 Node.js Web 开发更简单 & 有趣

Thanks