

Egg & Node.js 从小作坊走向企业级开发

天猪 / 阿里游戏前端负责人, Egg 核心开发者



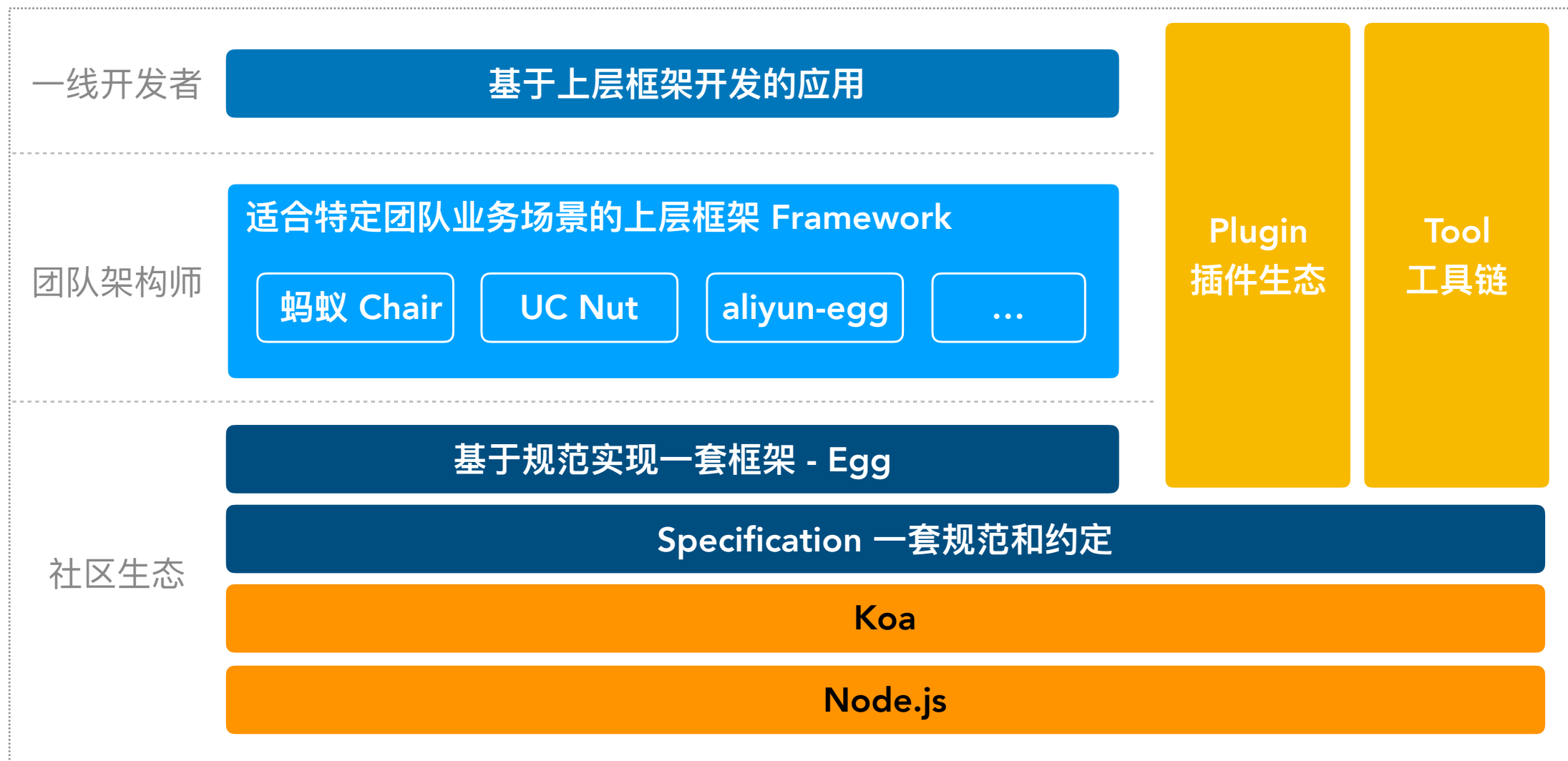
自我介绍



- ▶ 天猪
- ▶ 阿里游戏前端负责人, @广州 @Alibaba UC
- ▶ Node.JS / EggJS / Scrat / Vue
- ▶ Follow me at :
 -  [@atian25](#)
 -  [@liuyong25](#)
 -  [@liuyong25](#)
 -  [@atian25](#)



全景图

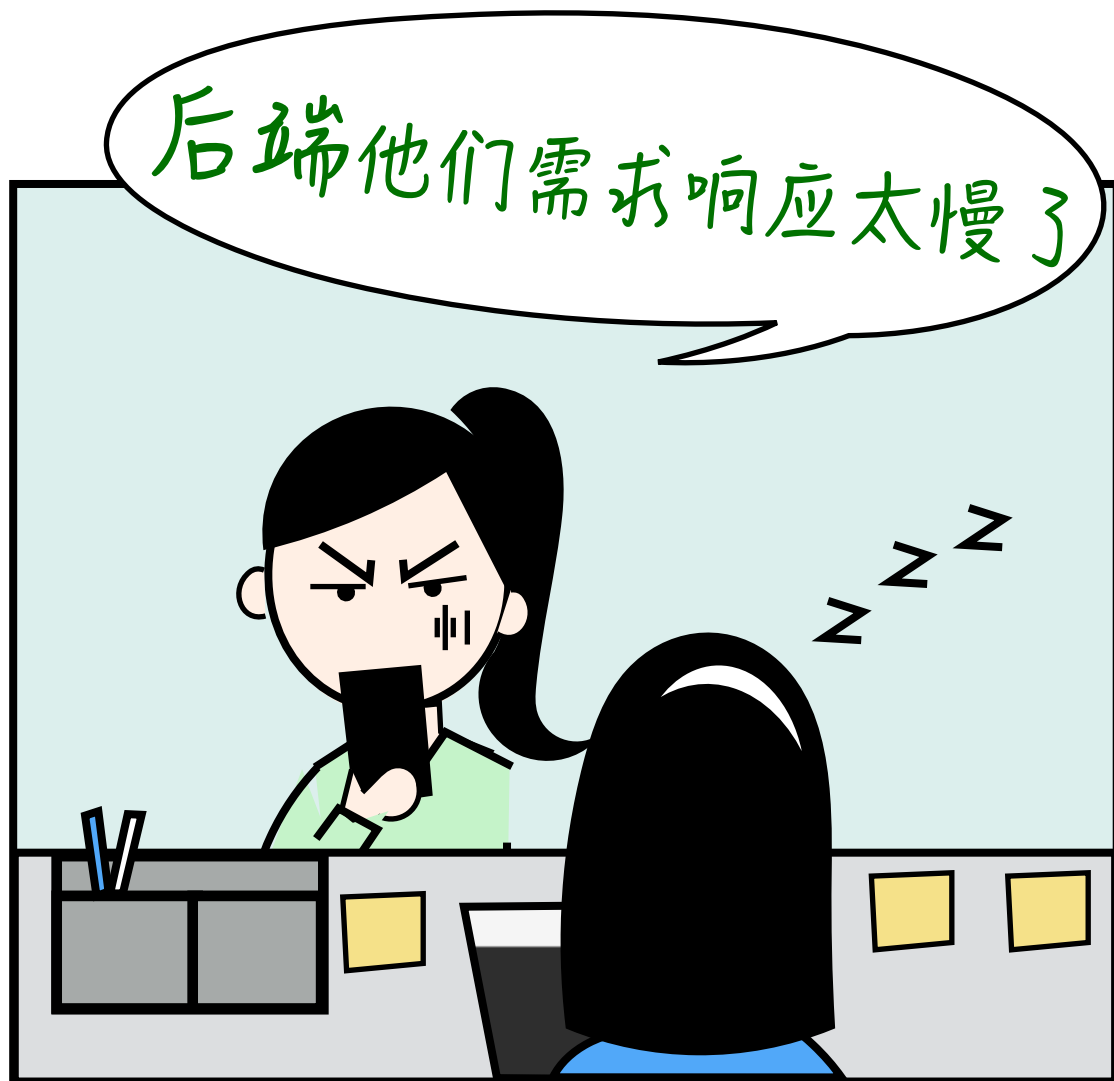


Base on Node.js

[1/8]

你是否遇到过?

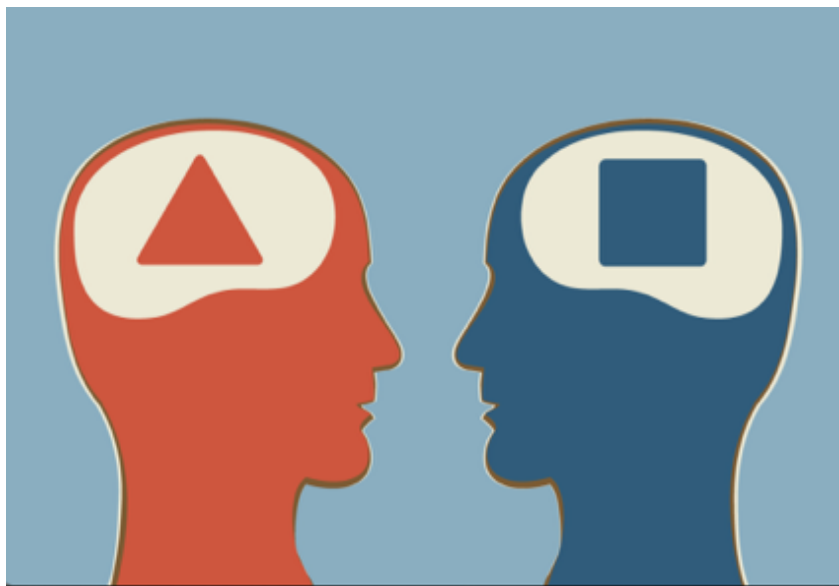
VS



前端小组



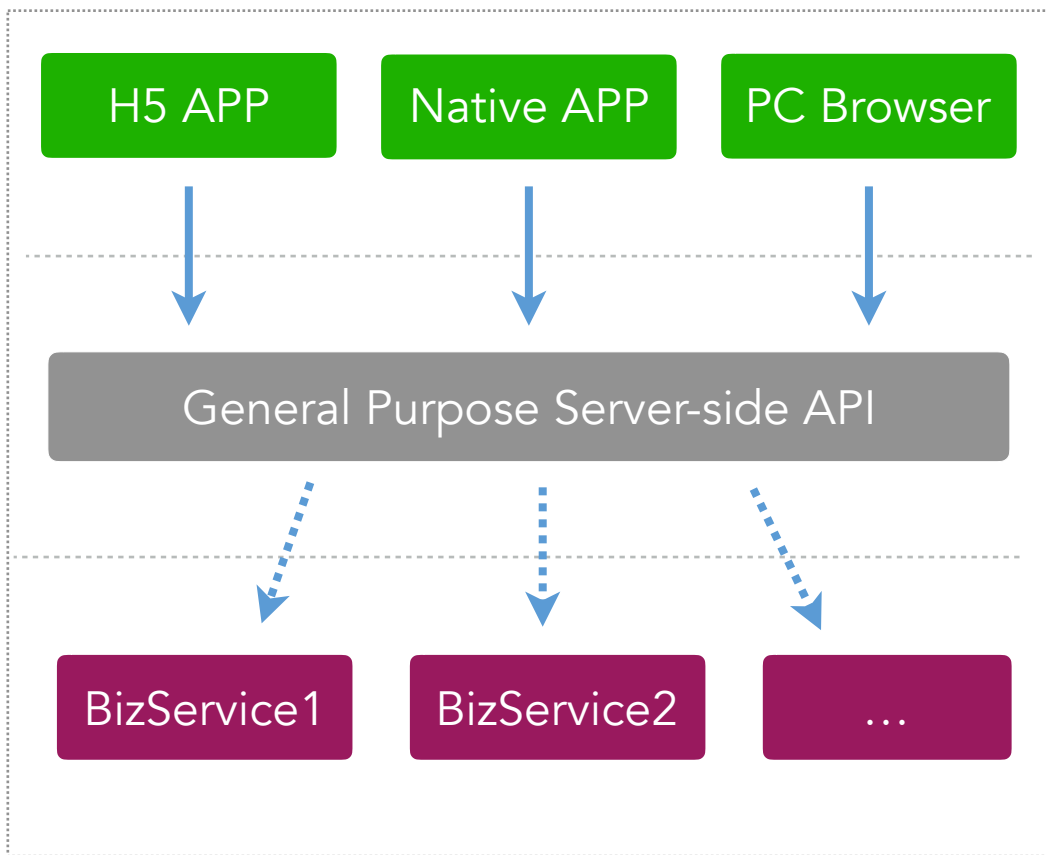
后端小组



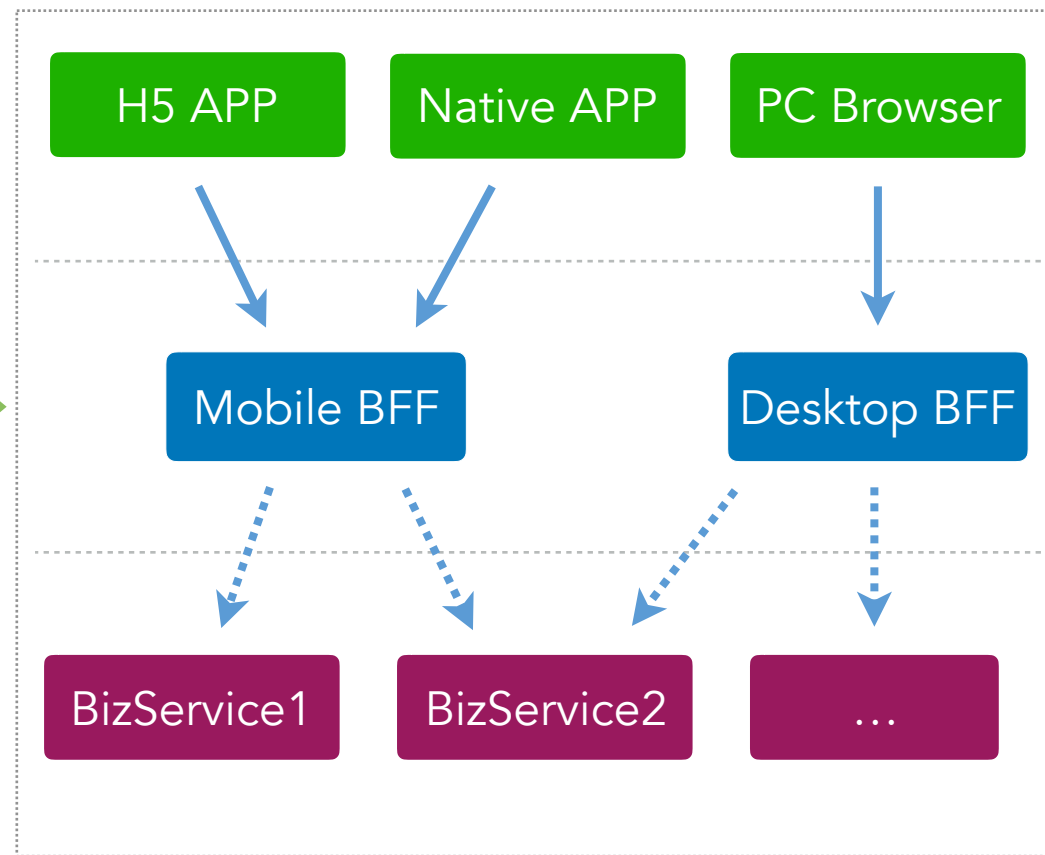
- ▶ 服务下沉与用户体验灵活性的矛盾
 - ▶ 服务趋向稳定，倾向下沉
 - ▶ 用户体验趋向不稳定，诉求服务的高度灵活与定制
 - ▶ 不同的设备对 API 有不同的诉求
 - ▶ API 灵活性对服务开发者要求太高
- ▶ 服务层 API 相对稳定，体验层 API 经常变化
- ▶ 服务端设计的接口究竟是面向 UI 还是只是通用服务？

BFF 为用户体验解围

传统方式:



困境中的答案: Backends for FrontEnds



BFF 为用户体验解围

一件重要的事情：服务自治

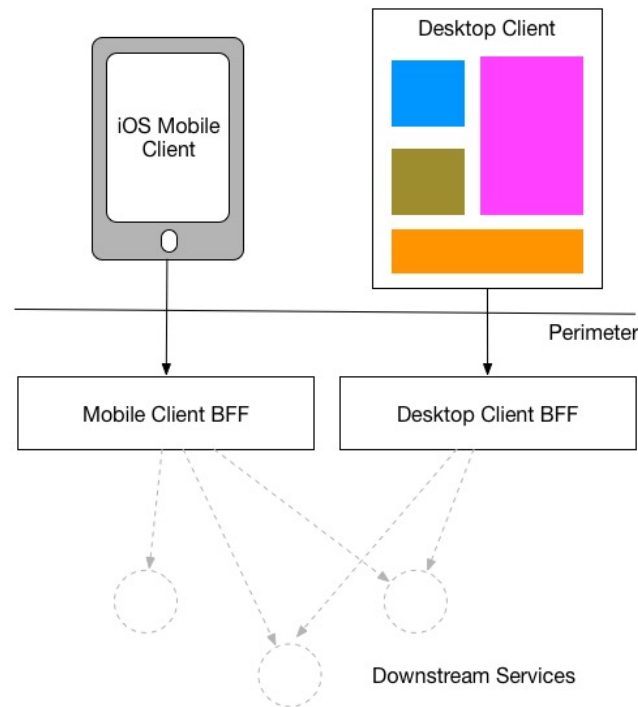
▶ 简而言之，BFF 谁使用谁开发

▶ 服务自治带来灵活与高效

▶ BFF 根据团队的技术栈选型：Java/Node/PHP/Python/Ruby...

▶ 我们的选择：

在我们的业务场景中，相对较优，生态最活跃，最能被前端接受的 Node.js



Base on Koa

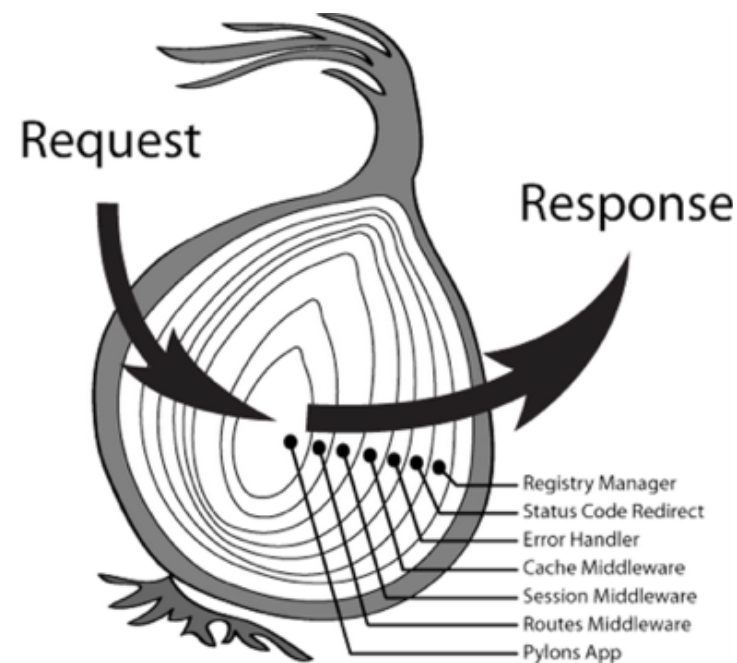
[2/8]

Base on Koa

koa

next generation web framework for node.js

- ▶ Koa based framework
- ▶ 当前解决异步编程最好的 Web 通用框架，洋葱模型
- ▶ 所有源代码 100% 掌握并且参与到核心代码贡献 (@fengmk2 @dead_horse)



Base on Koa

- ▶ 类似于 Connect 的定位，专注于 http 的 abstraction 层，很薄。
- ▶ Middleware 的局限
 - ▶ 定位是拦截用户请求，并在它前后做一些事情。
 - ▶ 但实际情况下，很多功能与请求无关，如定时任务，初始化，Application 扩展。
 - ▶ 功能之间的顺序不能简单的交给开发者，需要统一编排，管理。
- ▶ 对于企业级应用来说，还比较基础，往往还需要非常多的上层封装才能使用。

Specification

[3/8]

先思考几个问题：

企业级开发中需要关注的有哪些点？

而开发人员和团队技术负责人，关注点一样么？

理想的企业级 Web 开发的核心要素

编程模型约束

丰富的扩展点

可维护性

研发效率

多进程管理

日志

错误处理

安全

故障排除体系

本地开发工具包

测试

部署体系

跨语言 RPC

分布式中间件

会话管理

。 。 。

一个大规模团队的基础框架
最重要的是需要遵循一定的约束和约定。

QuickLook

[4/8]

QuickStart

```
$ npm install -g egg-init
```

```
$ egg-init showcase --type=simple
```

```
$ cd showcase && npm install
```

```
$ npm run dev
```

```
$ curl localhost:7001
```

编程模型规范

showcase

```
├─ app
|   ├─ controller (控制器)
|   |   └─ home.js
|   ├─ service (业务逻辑)
|   |   └─ github.js
|   ├─ view (模板)
|   |   └─ home.tpl
|   ├─ public (静态资源)
|   |   └─ main.css
|   └─ router.js (路由)
├─ config (配置)
|   ├─ config.default.js
|   ├─ config.prod.js
|   └─ plugin.js
├─ test (单元测试)
├─ README.md
└─ package.json
```

▶ 约定优于配置

▶ **Loader 机制** - 自动挂载

▶ Controller / Service / Config / ...

编程模型规范 - Controller

showcase

```
├─ app
│   ├── controller
│   │   └─ home.js
│   ├── service
│   │   └─ github.js
│   ├── view
│   │   └─ home.tpl
│   ├── public
│   │   └─ main.css
│   └─ router.js
├─ config
│   ├── config.default.js
│   ├── config.prod.js
│   └─ plugin.js
├─ test
├─ README.md
└─ package.json
```

控制器 + 路由 → 参数处理, 调用 API, 模板渲染

```
// app/controller/home.js
const Controller = require('egg').Controller;

class HomeController extends Controller {
  // 可在 `app/router.js` 中使用 `app.get('/', 'home.index')`
  async index() {
    // 参数处理
    const { query, service } = this.ctx;
    const org = query.org || 'eggjs';
    // 调用业务 API
    const data = await service.github.listReposByOrg(org);
    // 渲染模板
    await this.ctx.render('home.tpl', data);
  }
}

module.exports = HomeController;
```

编程模型规范 - Service

showcase

```
├─ app
  │   ├── controller
  │   │   └─ home.js
  │   ├── service
  │   │   └─ github.js
  │   └─ view
  │       ├── home.tpl
  │       └─ public
  │           └─ main.css
  └─ router.js
├─ config
  │   ├── config.default.js
  │   ├── config.prod.js
  │   └─ plugin.js
├─ test
├─ README.md
└─ package.json
```

业务逻辑层

```
// app/service/github.js
const Service = require('egg').Service;
class Github extends Service {
  // 可在 Controller 中使用 `ctx.service.github.listReposByOrg()`
  async listReposByOrg(org) {
    const endpoint = this.config.github.endpoint;
    const url = `${endpoint}/orgs/${org}/repos`;
    const repos = await this.ctx	curl(url, { dataType: 'json' });
    return repos.data.map(item => {
      return {
        name: item.name,
        // ...
      };
    });
  }
}
module.exports = Github;
```

编程模型规范 - Config

showcase

```
├─ app
│   ├── controller
│   │   └─ home.js
│   ├── service
│   │   └─ github.js
│   ├── view
│   ├── public
│   └─ router.js
├─ config
│   ├── config.local.js
│   ├── config.default.js
│   ├── config.prod.js
│   ├── config.unittest.js
│   └─ plugin.js
├─ test
├─ README.md
└─ package.json
```

配置文件 → 根据运行环境类型自动合并

```
// config/config.default.js
// 读取方式: `app.config.github.xx`
exports.github = {
  token: '123',
  endpoint: 'https://api.github.com',
};
```

```
// config/config.prod.js
exports.github = {
  // 在 prod 环境下被覆盖为 `abcd`
  token: 'abcd',
};
```

幕后功臣 - Loader

```
load() {  
  // app > plugin > core  
  this.loadApplicationExtend();  
  this.loadRequestExtend();  
  this.loadResponseExtend();  
  this.loadContextExtend();  
  this.loadHelperExtend();  
  
  // app > plugin  
  this.loadCustomApp();  
  // app > plugin  
  this.loadService();  
  // app > plugin > core  
  this.loadMiddleware();  
  // app  
  this.loadController();  
  // app  
  this.loadRouter();  
}
```

遵循 Specification 实现一套 Loader 即可。

loadService

loadController

loadRouter

loadToContext

loadToApp

loadFile



插件机制

[5/8]

插件机制

- ▶ **核心要素 - 丰富的扩展点**
- ▶ 前面提过 Middleware 局限性，不适合用于承载扩展的职责。
- ▶ **插件机制**
 - ▶ 就是一个**迷你的应用**，一样有 Service / Config / Extend / Middleware / ...
 - ▶ 插件是围绕某个功能组织的扩展集合
 - ▶ 插件之间可以声明依赖关系



代码君的演化之旅

showcase

- ├─ app
 - ├─ controller
 - └─ home.js
 - ├─ service
 - └─ github.js
 - ├─ extend
 - ├─ application.js
 - ├─ context.js
 - └─ helper.js
 - ├─ view
 - ├─ public
 - └─ router.js
- ├─ config
- ├─ test
- ├─ README.md
- └─ package.json

扩展点
Loader

0.1 时代 - 仅仅是应用内部的一段逻辑

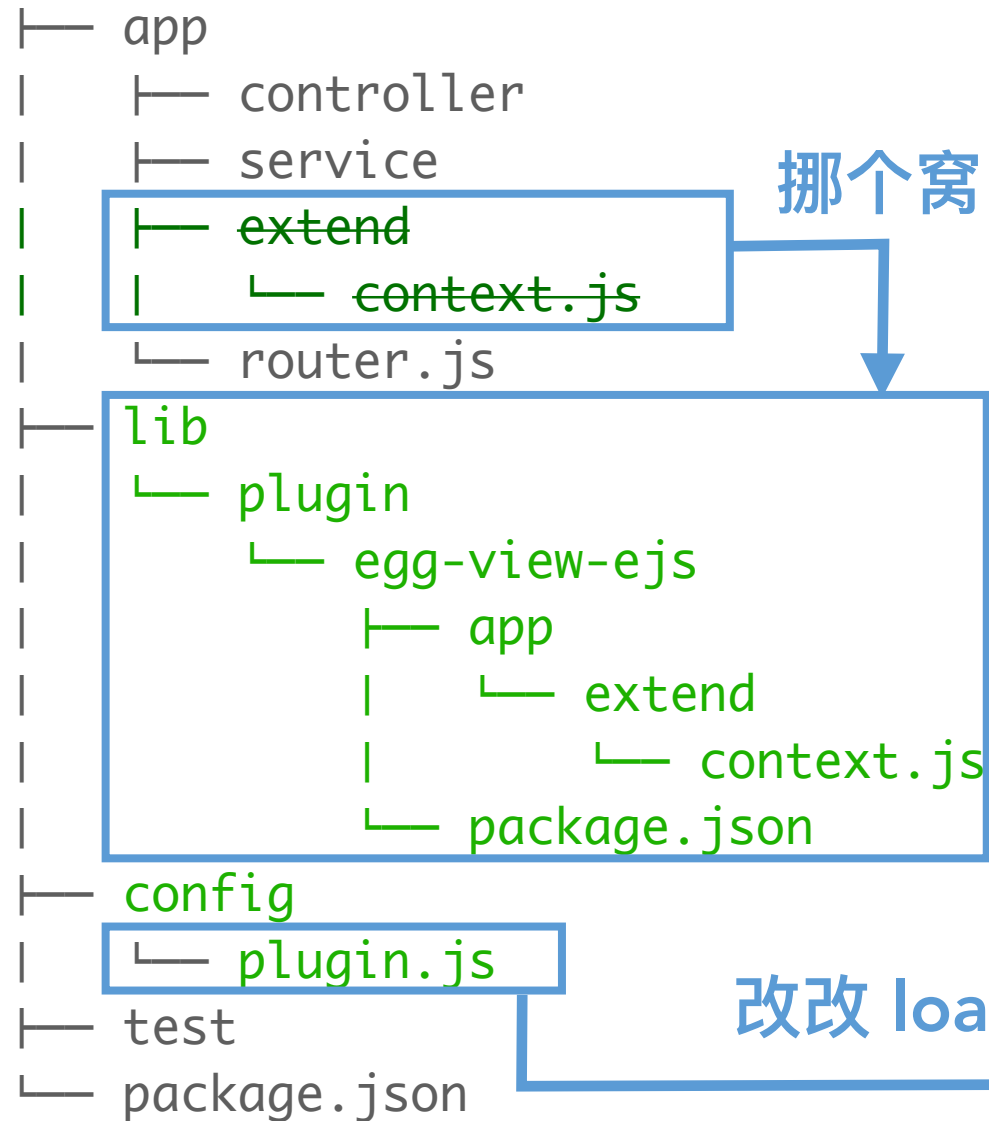
```
// app/extend/context.js
const ejs = require('ejs');
const path = require('path');

// 在 Controller 调用 `await ctx.render(tpl)`
exports.render = async function(tpl, locals) {
  const baseDir = this.app.config.baseDir;
  const root = path.join(baseDir, 'app/view');

  this.body = await new Promise((resolve, reject) => {
    const filePath = path.join(root, tpl);
    ejs.renderFile(filePath, locals, (err, html) => {
      if (err) return reject(err);
      return resolve(html);
    });
  });
};
```

代码君 v0.2

showcase



0.2 时代 - 在应用内部孵化的插件雏形

```
// config/plugin.js
const path = require('path');
const pluginDir = path.join(__dirname, '../lib/plugin');

// 挂载内部路径的插件
exports.ejs = {
  path: path.join(pluginDir, 'egg-view-ejs'),
};
```

然后可以王者一把了

1.x 时代 - 独立插件

showcase

├─ app

├─ lib

├─ plugin

└─ egg-view-ejs

├─ config

└─ plugin.js

├─ test

└─ package.json

分家啦~

发布个独立 npm 包

换个联系方式

```
// config/plugin.js
// 挂载独立插件
exports.ejs = {
  package: 'egg-view-ejs',
};
```

egg-view-ejs

├─ app

├─ extend

└─ context.js

├─ config

├─ test

├─ fixtures

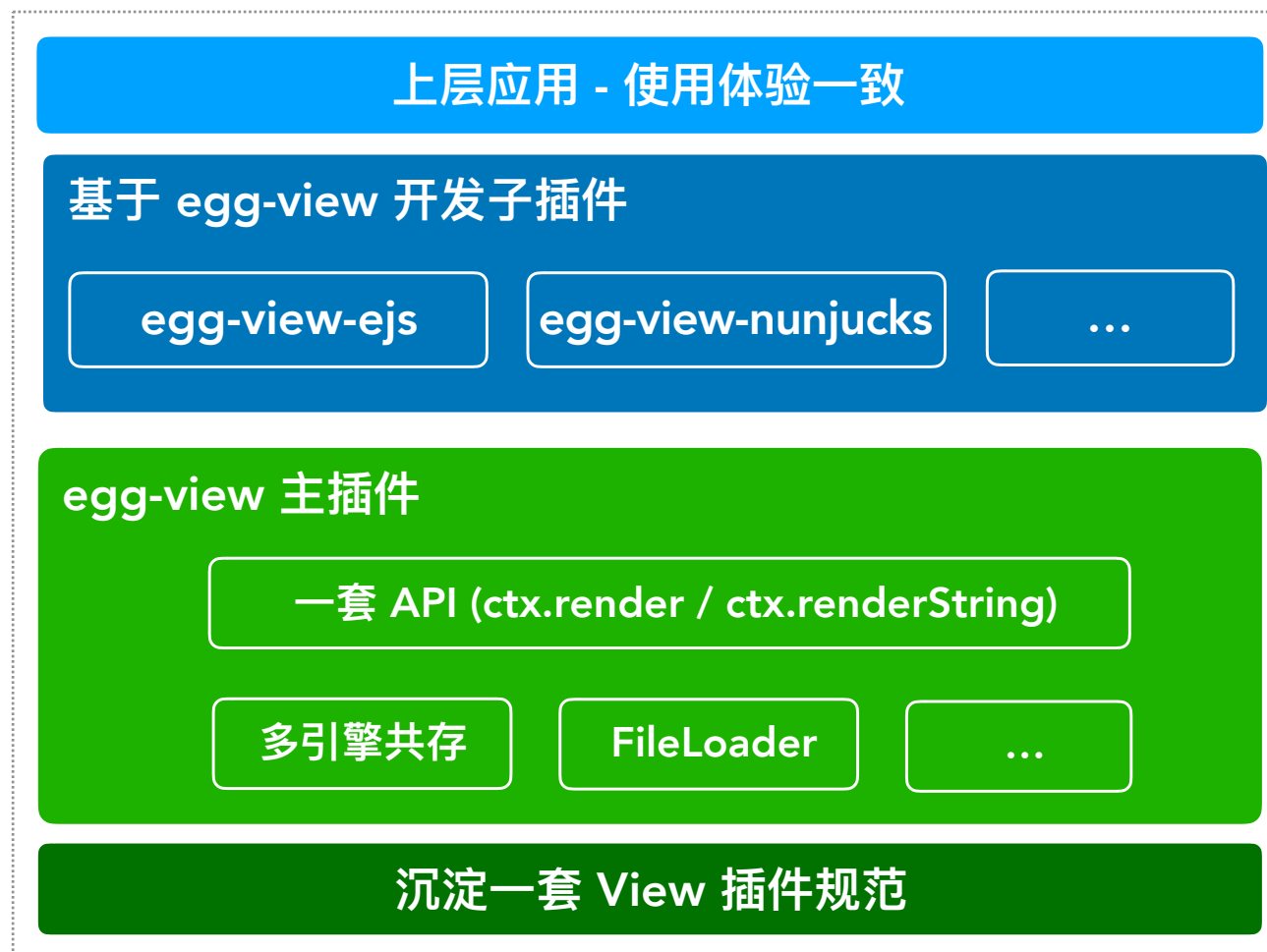
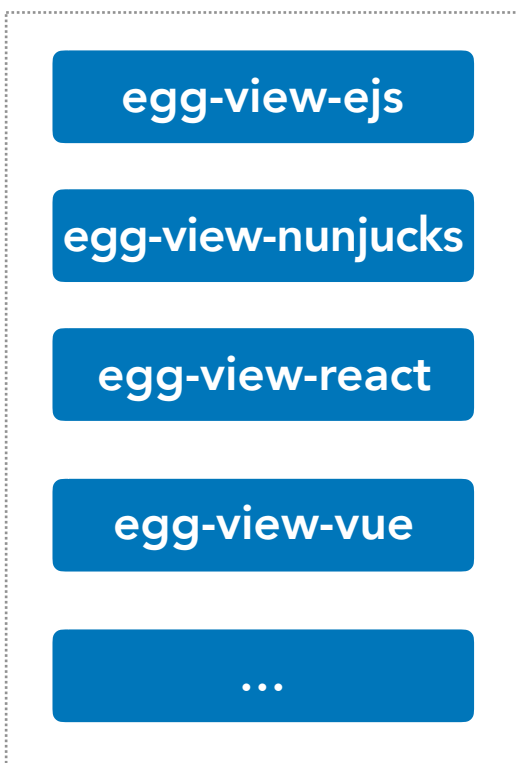
├─ view.test.js

├─ README.md

└─ package.json

然后发个微博炫耀下

2.x 时代 - 插件规范 + 差异化定制

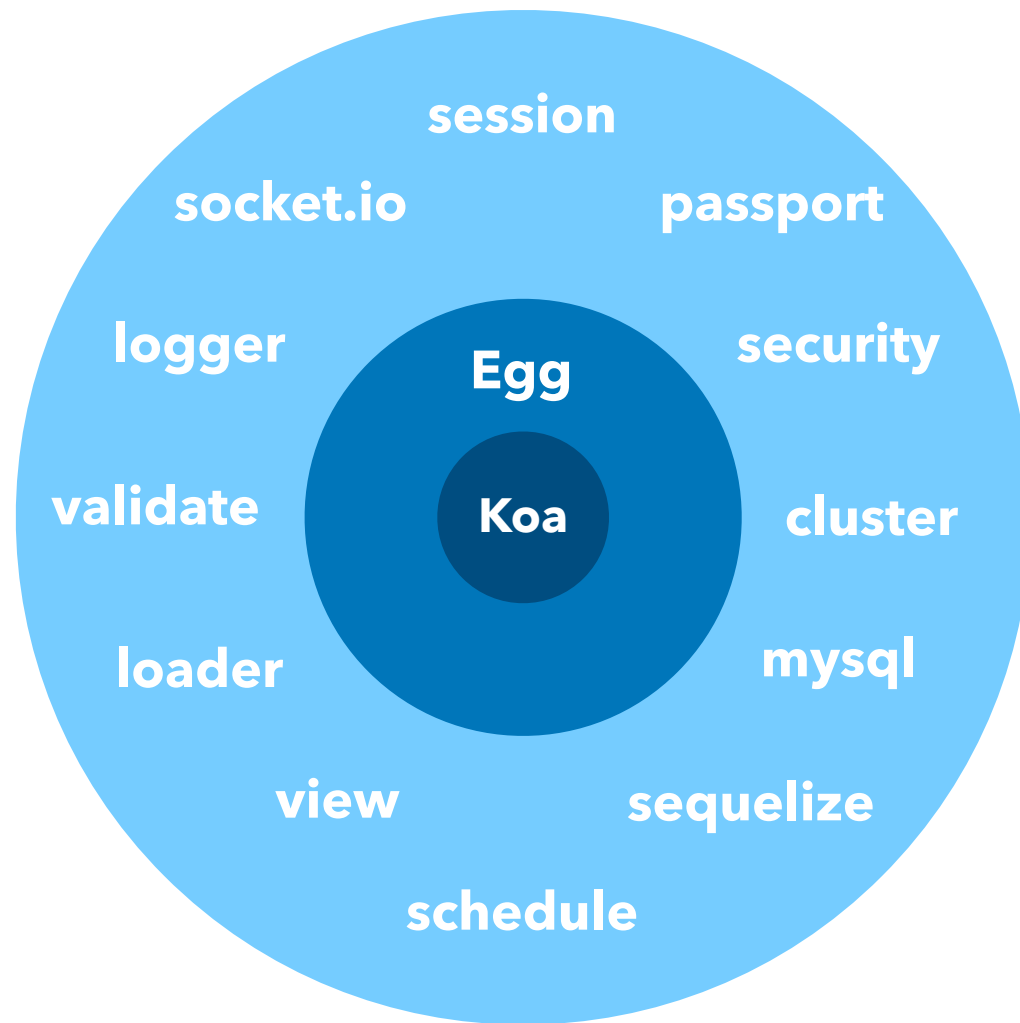


插件生态

微内核 + 插件机制

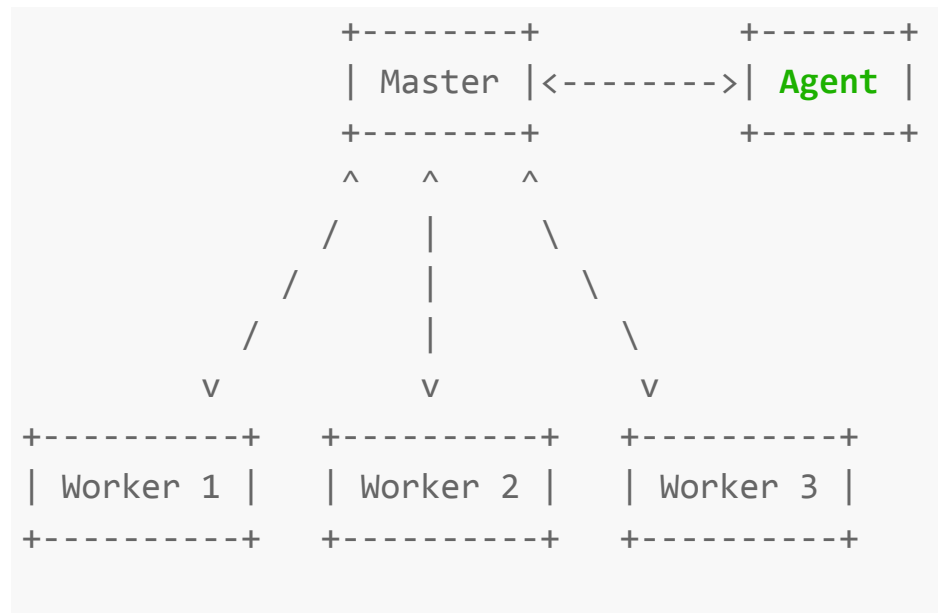
绝大部分功能由插件实现 → 企业级开发要素

- ▶ 进程: `egg-cluster`
- ▶ 安全: `egg-security`
- ▶ 模板: `egg-view-nunjucks / ...`
- ▶ 日志: `egg-logger / egg-tracer / ...`
- ▶ 数据库: `egg-mysql / egg-sequelize`
- ▶ 统一登录: `egg-passport-xxx`
- ▶ 兼容 **Koa** 中间件生态
- ▶ 大部分的插件都只需对社区模块简单包装...



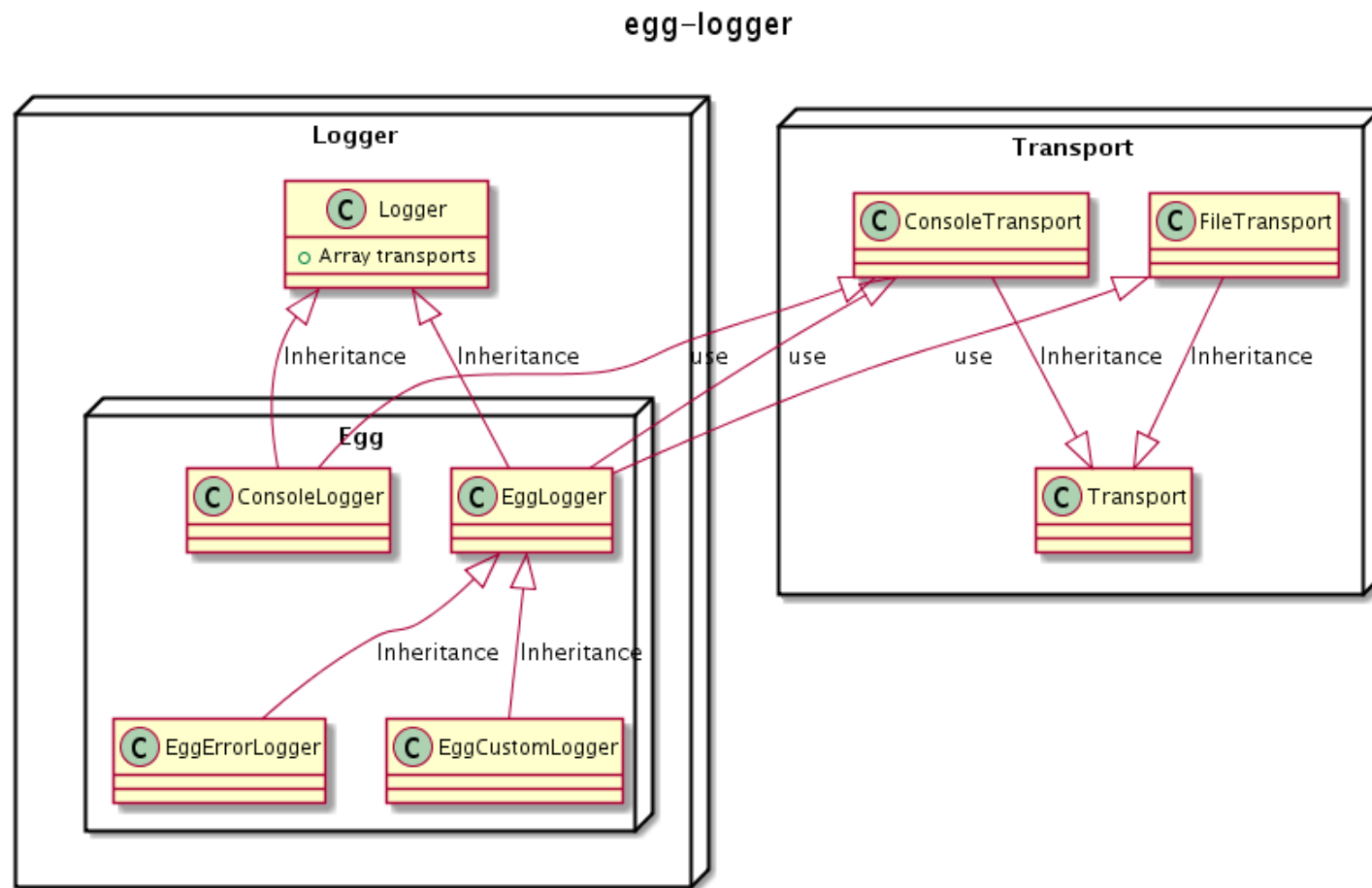
插件生态 - 进程管理

- ▶ 健壮性，处理未捕获异常，优雅退出
- ▶ **Master / Worker** 之外，还多了个 **Agent** 进程
 - ▶ 只有一个 **Agent**，负责脏活累活
 - ▶ 不对外服务，专门处理公共事务
 - ▶ 如：定时日志切割，公共资源访问，后端长连接
- ▶ 提供进程间通信模型
- ▶ 自行实现，不使用 **PM2**，越精简的代码越稳定



插件生态 - 日志

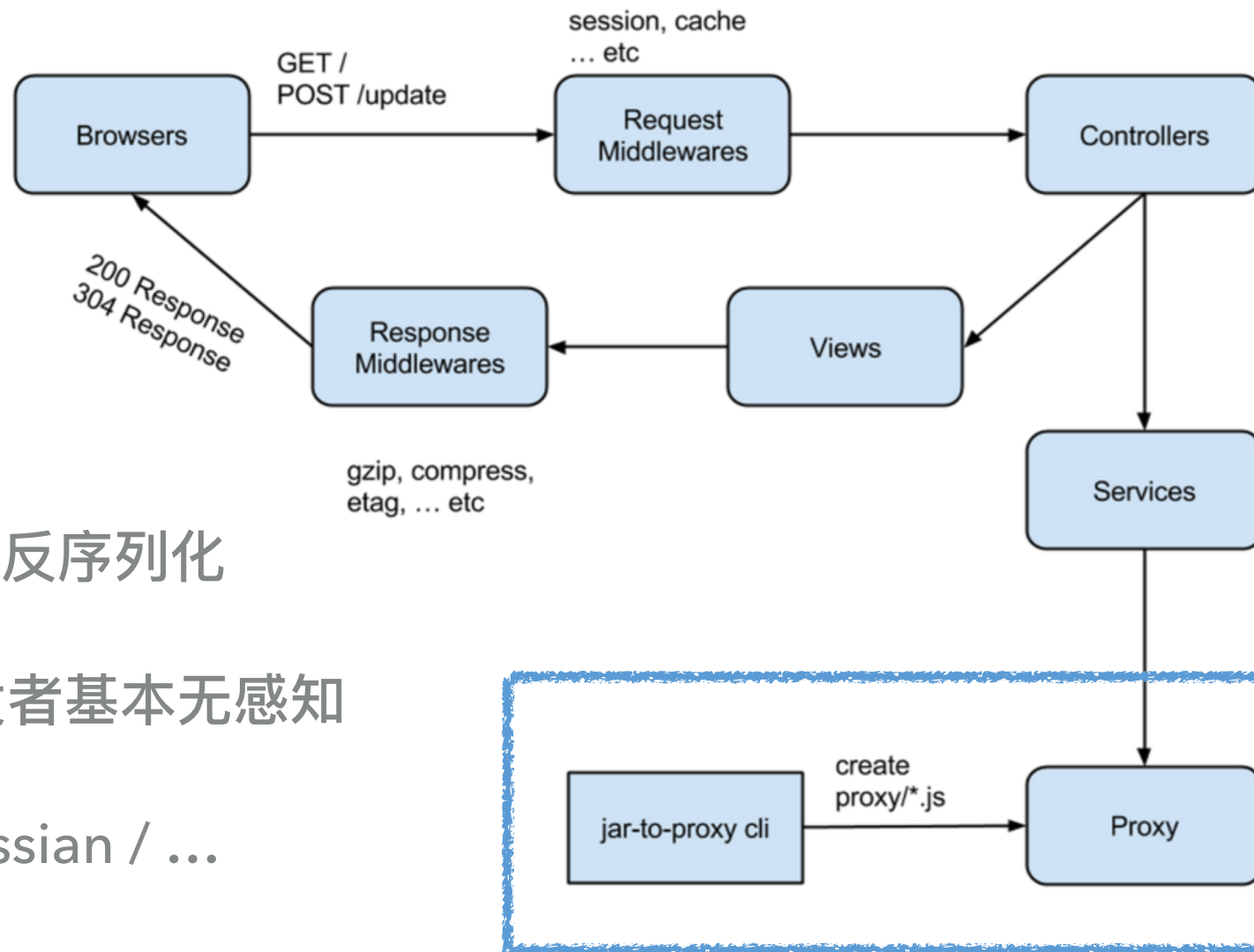
- ▶ 日志分级
- ▶ 自定义日志
- ▶ 多进程日志
- ▶ 自动切割
- ▶ 统一错误日志
- ▶ 高性能
- ▶ 全链路跟踪 (open-tracer)
- ▶ 统一的日志规范, 方便接入监控



插件生态 - 安全

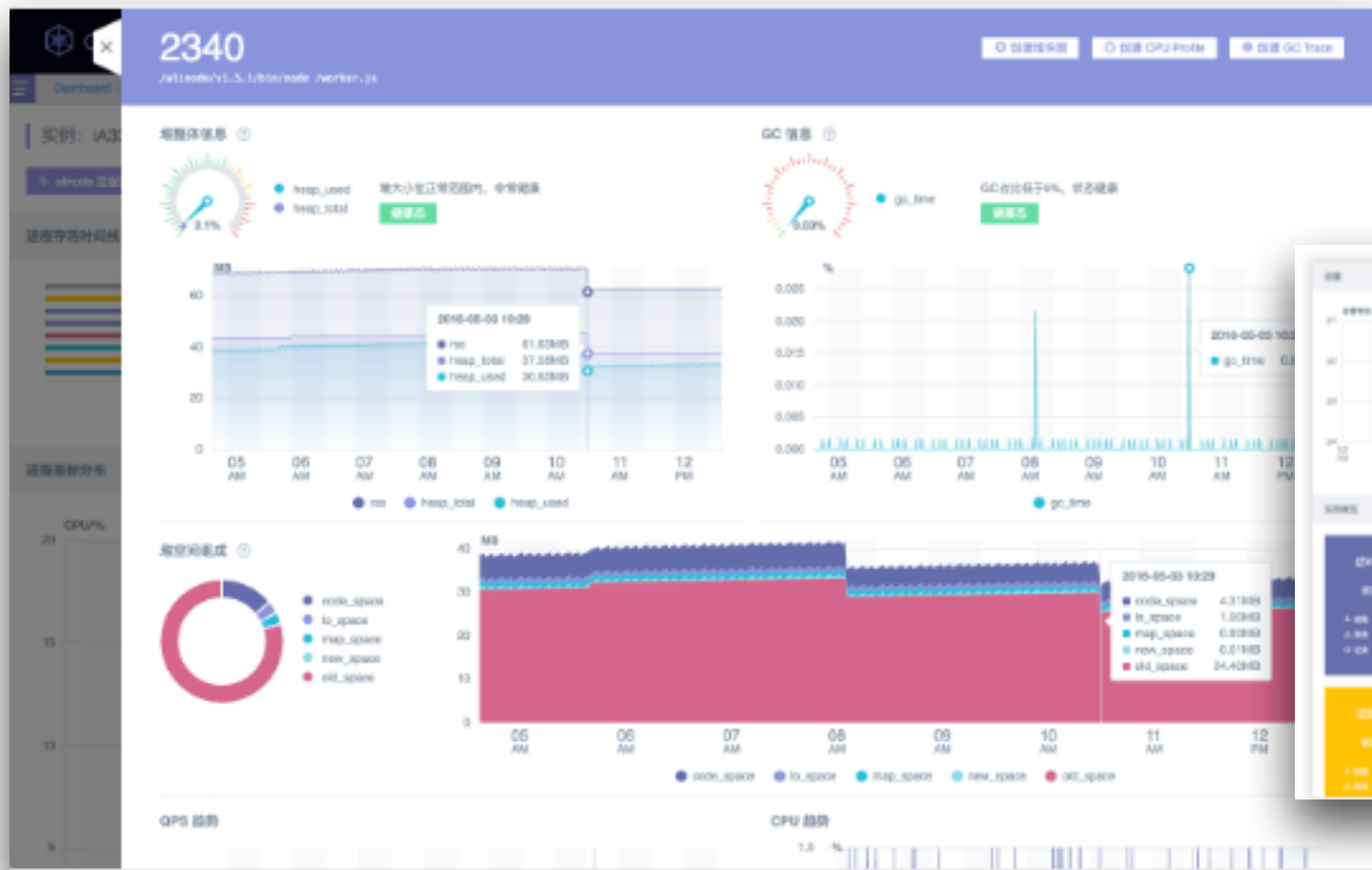
- ▶ 集合了阿里集团多年的安全经验积累
- ▶ 内建安全机制
- ▶ **XSS / CSRF / 常见安全头 / 钓鱼 / 标签清洗 / 中间人攻击 / ...**
- ▶ 对称/非对称加解密
- ▶ 结合代码扫描
- ▶ 传送门: [egg-security](#)

插件生态 - 跨语言 RPC

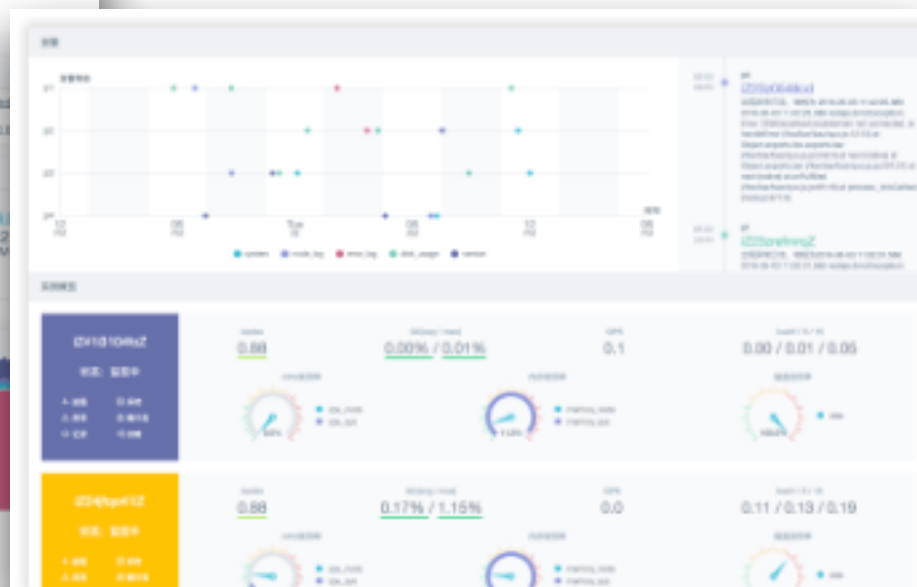


- ▶ 微服务时代，异构语言
- ▶ 通过自动化手段完成序列化与反序列化
- ▶ 达到 JS 与 Java 互操作对开发者基本无感知
- ▶ HSF / GRPC / ProtoBuf / Hessian / ...

插件生态 - 故障排除 AliNode



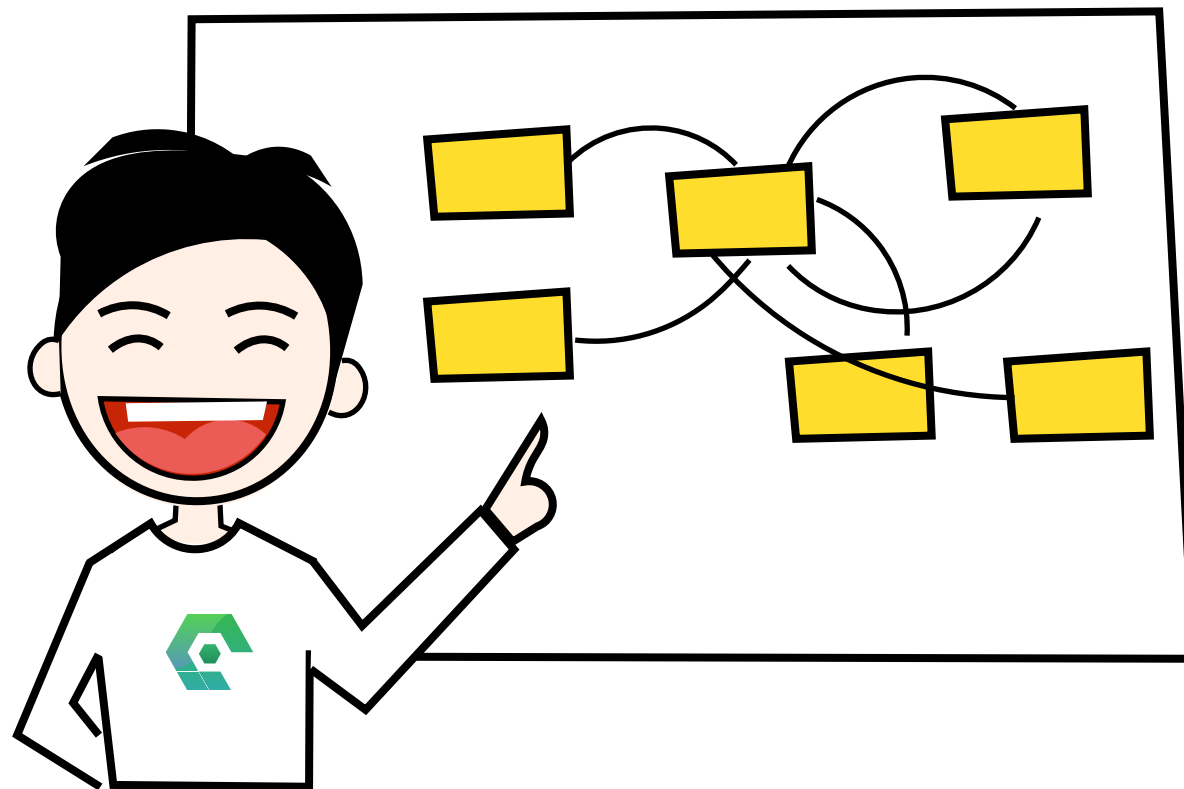
性能分析, 问题定位



提供 egg-alinode 插件自动接入

<http://alinode.aliyun.com/>

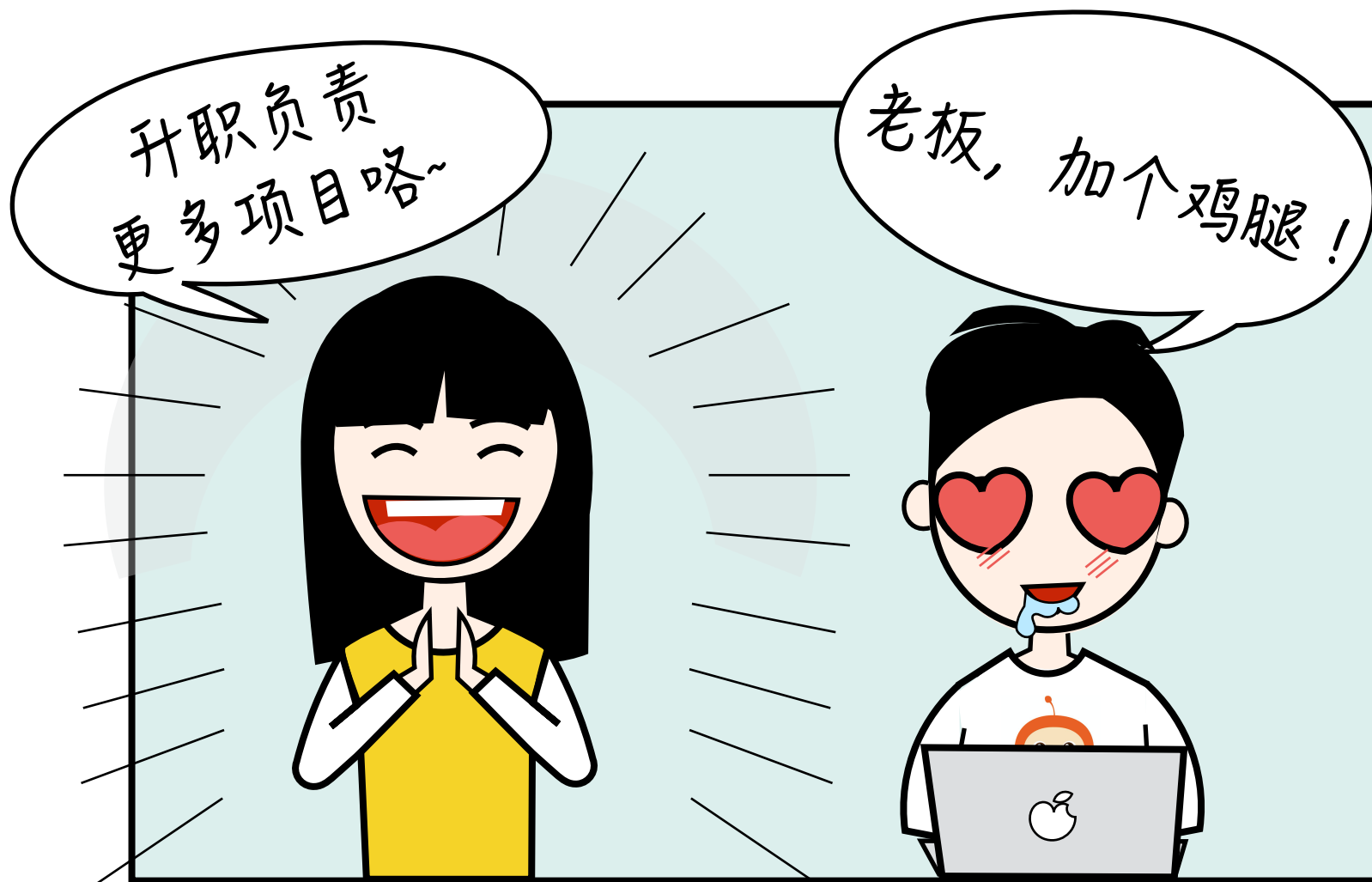
企业级开发的核心要素就这么搞定了!



框架定制能力

[6/8]

升职加薪后的故事



升职加薪后的故事



升职加薪后的故事



升职加薪后的故事



作为团队的技术负责人

▶ 你的团队是否遇到过：

- ▶ 维护很多个项目，每个项目都需要复制拷贝诸如 `gulpfile.js` / `webpack.config.js` 之类的文件。
- ▶ 每个项目都需要使用一些相同的类库，相同的配置。
- ▶ 在新项目中对做了一个优化后，如何同步到其他项目？
- ▶ 紧急修复后如何快速升级所有项目？

▶ 你的团队是否需要：

- ▶ 统一的技术选型，包括基础措施，插件，模板，数据库 ...
- ▶ 统一的默认配置 / 统一的部署方案 / 统一的约束 / ...

框架定制能力

- ▶ 『框架』是对适合 **特定业务场景** 的 **最佳实践** 的约束和封装
- ▶ **便于统一团队的开发模式，一致的开发体验，一次性学习成本**
- ▶ 跟应用，插件区别不大，一样有 Service / Config / Extend / Middleware / ...
- ▶ 支持多层继承，支持自定义加载规范
- ▶ 基于 npm , 便于统一管控，升级，维护

3.x 时代 - 定制团队框架

```
showcase
├─ app
│   ├─ controller
│   ├─ service
│   └─ view
├─ public
└─ router.js
```

```
├─ config
│   ├─ config.default.js
│   └─ config.prod.js
└─ plugin.js
```

```
├─ test
├─ README.md
└─ package.json
```

分家啦~ 发布个独立 npm 包

声明依赖

```
// package.json
{
  "name": "showcase",
  "egg": {
    "framework": "yadan"
  },
  "dependencies": {
    "yadan": "^1.0.0"
  }
}
```

```
yadan
├─ app
│   └─ service
│       └─ extend
│           └─ context.js
├─ config
│   └─ config.default.js
└─ plugin.js
```

```
├─ test
├─ app.js
├─ README.md
└─ package.json
```

框架升级后，应用无需修改代码，重新部署一次即可。

3.x 时代 - 自定义加载规约 (Egg 的核心能力 - Loader)

yadan

- ├─ app
 - │ ── service
 - │ ── extend
- ├─ config
 - │ ── config.default.js
 - │ ── plugin.js
- ├─ test
- ├─ **app.js**
- └─ package.json

showcase

- ├─ app
 - │ ── controller
 - │ ── service
 - │ ── **model**
 - │ ── **user.js**
 - │ ── **news.js**
- ├─ config
- └─ package.json

```
// app/extend/context.js
module.exports = app => {
  // 读取所有的 `app/model` 目录
  const directory = this.getLoadUnits().map(unit => path.join(unit.path, 'app/model'));
  // 挂载 `ctx.model.news.list()`
  app.loader.loadToContext(directory, 'model');
};
```

参见 [egg-sequelize](#)

代码君的演化之旅

渐进式

几乎无痛

闭环

应用内部逻辑 → 应用内部插件 → 独立插件 → 插件集规范 → 沉淀到框架

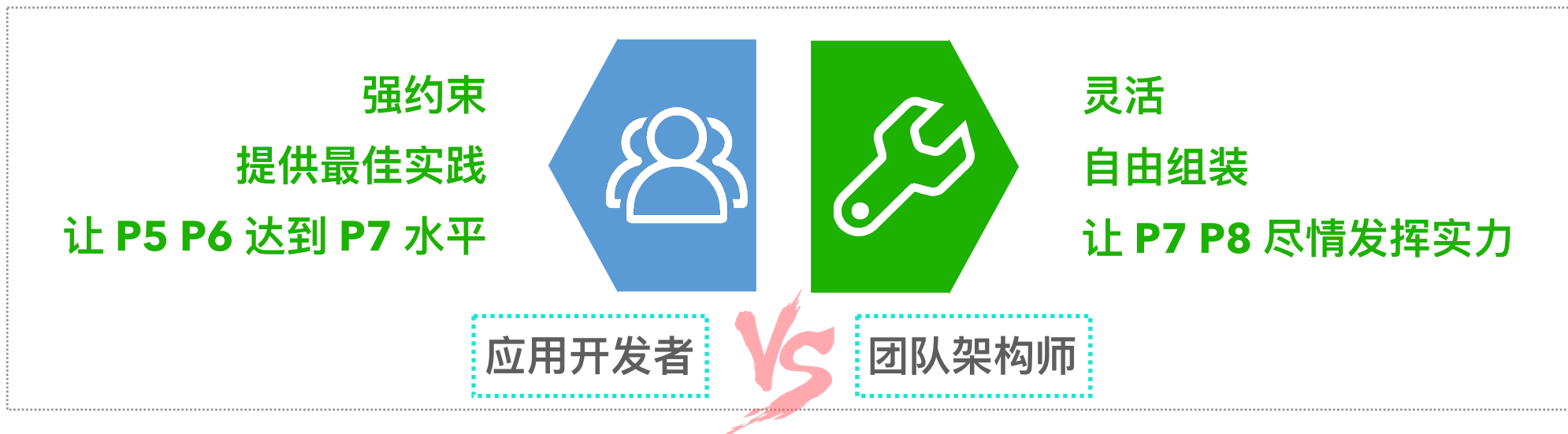


差异化定制

生态共建

完美平衡

应用开发者 vs 团队架构师

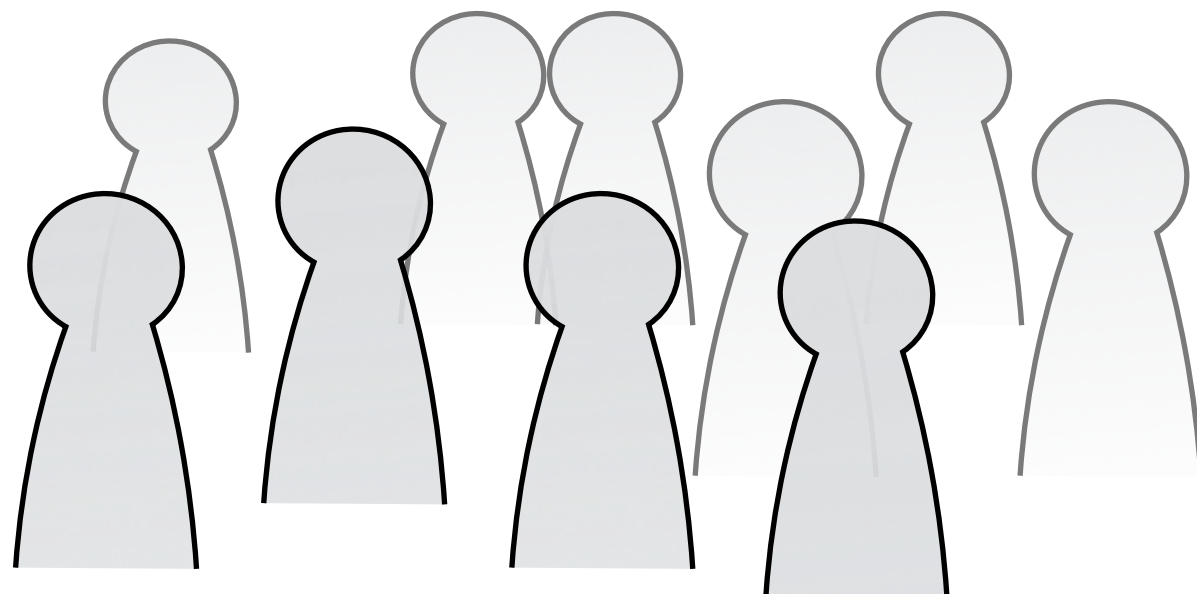


- ▶ 帮业务做决定，提供最佳实践的同时，给业务一定的自由度。
- ▶ 管生还管养，整合开发，调试，测试和构建，支撑研发的每一个阶段。
- ▶ 对开发者强约束提升研发效率。
- ▶ 对于团队架构师，则提供灵活的定制能力。

团队开发规范 + 工具链

[7/8]

再好的规范，不遵循，你也没辙。



基于 GitHub 的协作模式

- ▶ 基于 GitHub 的『一站式 + 硬盘式 + 异步协作』模式
- ▶ 提供给程序员的服务，如果能够做到不转移、不分散他们的注意力，就是积了大德。
- ▶ GitLab / GitHub 做到了，一切以代码为核心，围绕在代码周边的是：需求（issue）、持续集成（gitlab-ci）、Code Review（原生态天然自带功能，非常自然）等等
- ▶ 『基于 GitHub 的团队协作模式 - 前生篇』

基于 GitHub 的协作模式

发起 RFC 提案 → Coding → Pull Request → CI 持续集成 → Code Review → Release

- ▶ RFC 提案：背景 / 遇到的问题 / 解决思路 / 方案概要 / 伪代码
- ▶ 编写提案，是整理思路的过程，由此发起脑爆的效率 high，而且是可追溯的。
- ▶ 即使是当时没有参与讨论的开发者，事后也能通过 issue 了解某个功能设计的前因后果。
- ▶ CI 包括 Lint / 单测 / 覆盖率 / 安全 等检查，Code Review 需至少 2 个人 +1，Release 遵循 Semver 规则。

基于 GitHub 的协作模式

[RFC] Rethink view #398

Closed popomore opened this issue on Feb 17 · 72 comments



popomore commented on Feb 17 • edited by atian25

Owner + 👤 ✎

我们现在的模板规范, 约定插件名为 view, 这样可以通过配置不同插件来切换模板, controller 不需要修改。

但是会遇到多模板的情况的, 比如 markdown 渲染会和普通模板同时存在, 而且应用在模板过渡场景也会遇到多模板的情况, 所以这种模板约定就会存在问题。

方案

提供模板注册机制, 开发者可根据习惯来配置模板。



popomore merged commit fb71b03 into master on Feb 22

6 checks passed

- ✓ Node Security No known vulnerabilities found
- ✓ codecov/patch 100% of diff hit (target 100%)
- ✓ codecov/project 100% (+0%) compared to ef5dc24
- ✓ continuous-integration/travis-ci/pr The Travis CI build passed
- ✓ continuous-integration/travis-ci/push The Travis CI build passed
- ✓ security/snyk No new vulnerabilities



fengmk2 reviewed on Feb 20

[View changes](#)

lib/context_view.js

```
31 + */
32 + render(...args) {
33 +   const self = this;
34 +   return co(function* () {
```



fengmk2 on Feb 20 Owner

可以跟现在 benchmark 里面的 render 对比, 性能应该减少不少



popomore on Feb 20 Owner

应该会有减少, 我测下。



atian25 on Feb 20 Owner

eggjs / egg-view-nunjucks



build passing

Current Branches Build History Pull Requests > Build #149

✓ Pull Request #11 feat: [BREAKING_CHANGE] depend on egg-view

- 📄 Commit dc18d52
- 📄 #11: feat: [BREAKING_CHANGE] depend on egg-view
- 📄 Branch master
- 👤 popomore authored and committed

Build Jobs

- ✓ # 149.1 📄 </> Node.js: 6
- ✓ # 149.2 📄 </> Node.js: 7

[Details](#)



codecov-io commented on Feb 19 • edited

Codecov Report

! No coverage uploaded for pull request base (master@46be1 means.
The diff coverage is 100% .

##	Coverage	Diff	##
##	master	#1 +/-	##
Coverage	? 100%		
Files	? 5		
Lines	? 87		
Branches	? 0		
Hits	? 87		
Misses	? 0		
Partials	? 0		

Impacted Files	Coverage Δ
lib/context_view.js	100% <100%> (0)
app/extend/context.js	100% <100%> (0)
lib/view_manager.js	100% <100%> (0)
app/extend/application.js	100% <100%> (0)
config/config.default.js	100% <100%> (0)

[How to create a new repository](#)

单元测试很重要

- ▶ 你的代码质量如何度量?
- ▶ 你是如何保证代码质量?
- ▶ 你敢随时重构代码吗?
- ▶ 你是如何确保重构的代码依然保持正确性?
- ▶ 你是否有足够信心在没有测试的情况下随时发布你的代码?
- ▶ 如果答案都比较犹豫，那么就证明我们非常需要单元测试。




传送门：五个震耳欲聋的问题

单元测试很重要

showcase

```
├─ app
│   ├── controller
│   │   └─ home.js
│   ├── service
│   │   └─ github.js
│   └─ view
├─ public
├─ router.js
├─ config
├─ test
│   ├── app
│   │   ├── controller
│   │   │   └─ home.test.js
│   │   └─ service
│   │       └─ github.test.js
├─ README.md
└─ package.json
```



```
// test/app/controller/home.test.js
const { app, assert } = require('egg-mock/bootstrap');

describe('test/app/controller/home.test.js', () => {
  it('should GET /', async () => {
    const response = await app.httpRequest().get('/');
    assert(response.status === 200);
    assert(response.body.includes('egg'));
  });
});
```

```
// test/app/service/github.test.js
const { app, assert } = require('egg-mock/bootstrap');

describe('test/app/service/github.test.js', () => {
  it('listByOrgs', async () => {
    const ctx = app.mockContext();
    const github = ctx.service.github;
    const result = await github.listReposByOrg('eggjs');
    assert(result.length === 10);
    assert(result[0].name.includes('egg'));
  });
});
```

既要有流程，也要有工具来降低使用成本。

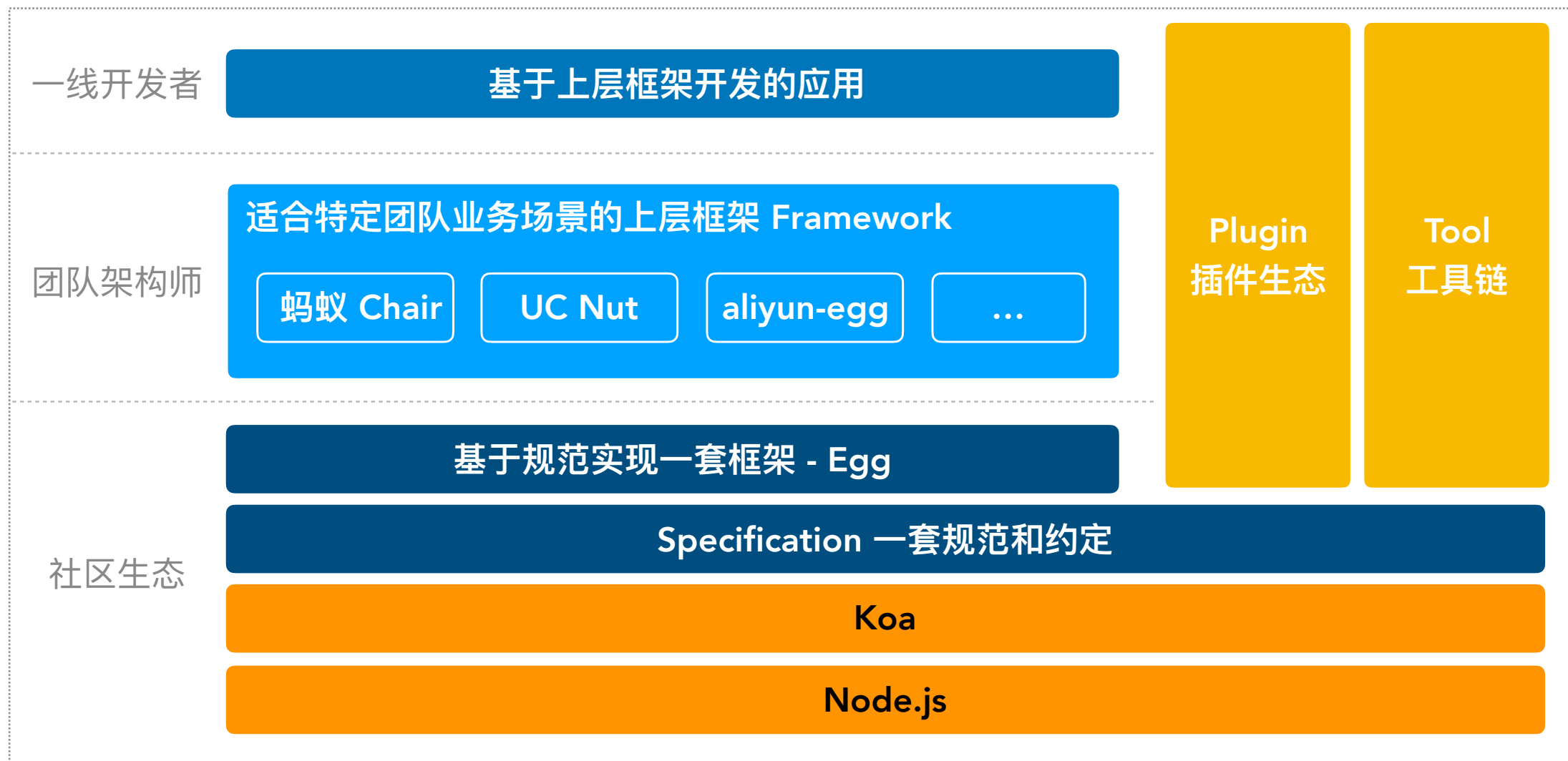
- ▶ 核心要素：研发效率
- ▶ egg-bin 工具集成了 **dev / debug / test / cov** 等功能，减少开发期配置成本。
- ▶ egg-mock 方便编写测试用例，coffee 方便多进程和命令行工具的测试。
- ▶ cnpm 加速依赖安装和 GFW 问题。
- ▶ 框架万年兼容 vs 自动化升级旧代码 (egg-codemod / jscodeshift)
- ▶ autod / egg-development / egg-development-stub / ...

版本发布遵循 Semver 规则

团队协作需要相互信任，遵守彼此的承诺。

- ▶ 遵循 Semver，使用 ^ 来引入依赖，不锁版本，更积极的升级策略。
- ▶ 紧急升级只需重新构建部署一次，无需众多应用修改代码。
- ▶ 完善的测试 / 及时止血 / 快速修复
- ▶ 应用开发者 vs 团队技术负责人的视角
- ▶ 『为什么我不使用 shrink-wrap (lock)』 / 『如何挑选高质量的 Node.js 模块?』

全景图



Egg

[8/8]

寓意 — 孵化新生



Born to build

better enterprise
frameworks and apps with
Node.js & Koa

为企业级框架和应用而生

开始使用

GitHub



<https://eggjs.org/>

Egg Timeline

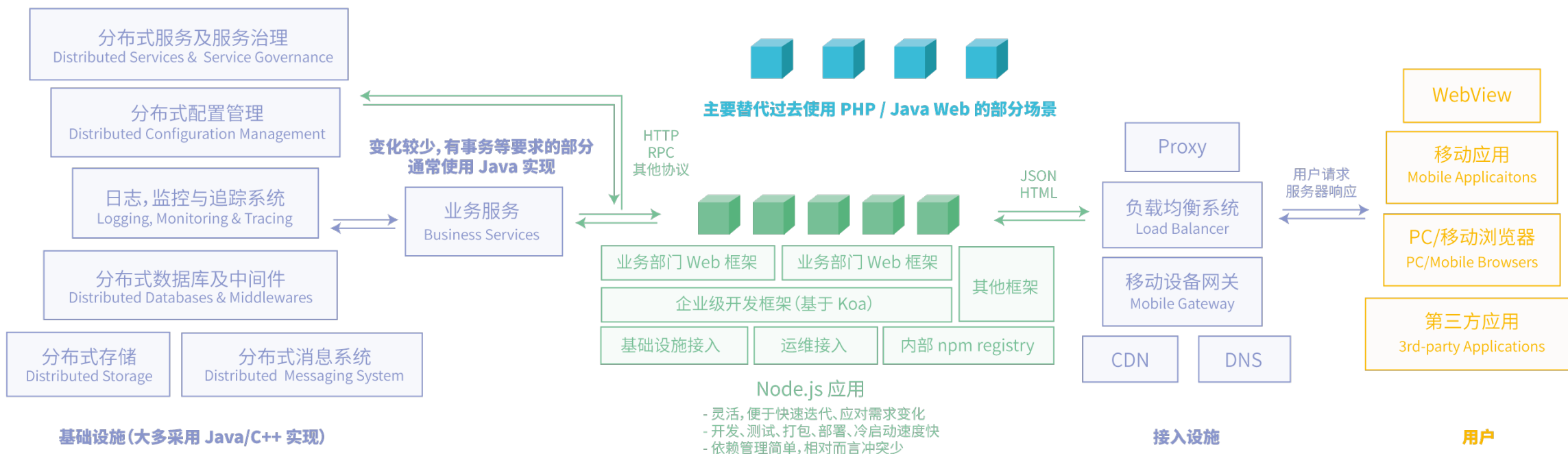
- 2011 ○ 阿里是业界最早的一批使用 Node.js 来做线上大流量应用的公司
- 2013 ○ Chair, 支付宝基础 Web 框架, Egg 前身
- 2015.11 ○ **Egg 第一个内部版本, 跨部门闭关共建**
- 2016.05 ○ 广泛使用在绝大部分阿里的前端 Node.js 应用
- 2016.09 ○ Egg 正式开源, JSConf China 2016
- 2017.03 ○ **Egg 发布 1.0 正式版 (对应内部 3.0)**

发起 JSConf China
其实是为了推动市场
人才储备, 方便招人。



Egg 在阿里

阿里的 Node.js 应用场景



▶ Java - 基础设施

▶ Node - 灵活, 快速迭代

Egg 在阿里



社区化运作模式，内外继承，最大程度复用，一份代码，无双向同步

Egg 在阿里



- ▶ 知乎热门问答：如何评价阿里开源的企业级 Node.js 框架 EggJS
- ▶ **Egg 是阿里 Node.js 应用的核心基础设施**，由各 BU 前端 Leader 合力共建。
- ▶ 广泛使用在阿里的绝大部分部门，包括天猫，蚂蚁，UC，游戏，优酷等等。
- ▶ 稳定支撑了 15 和 16 年天猫双11大促，顶级流量压力。
- ▶ Node 的 Collaborator 有 5 个国人，其中 4 个在阿里，并且有国内唯一的一个核心技术委员会 CTC 成员。

为什么你应该在 Egg 的基础上 定制自己的框架

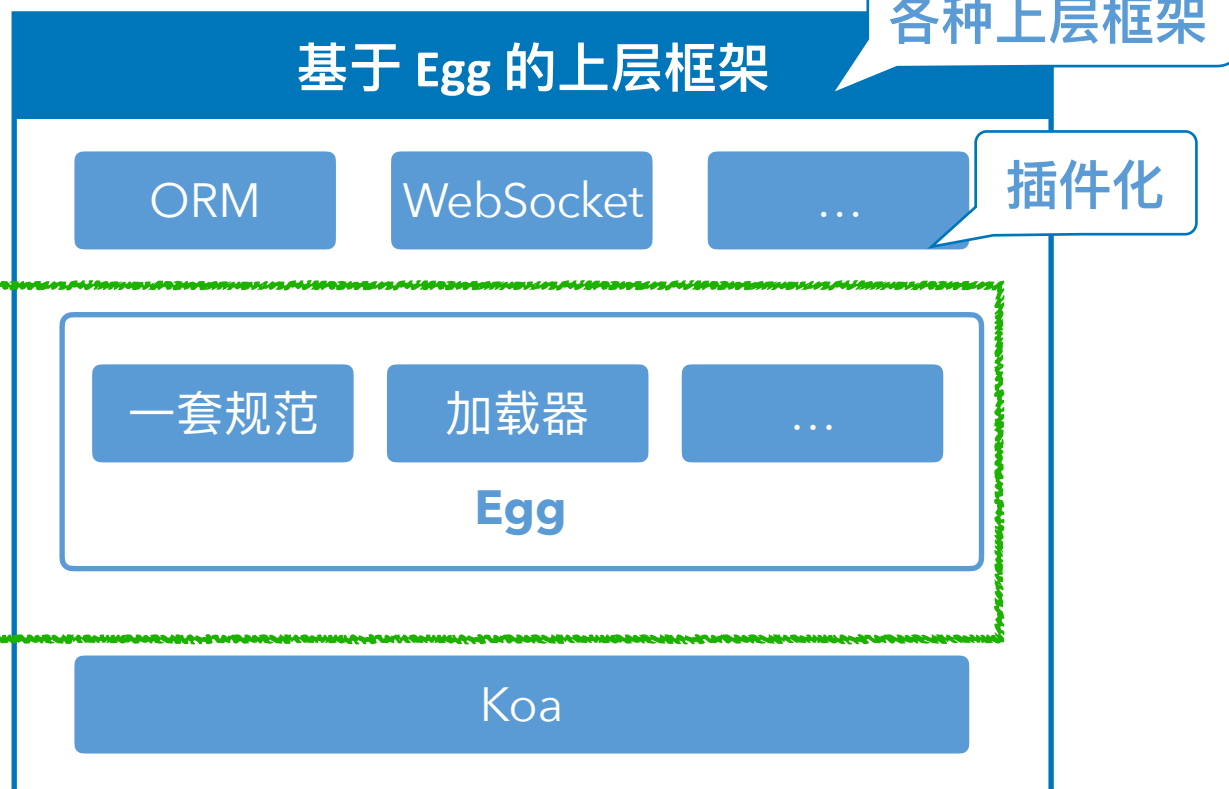
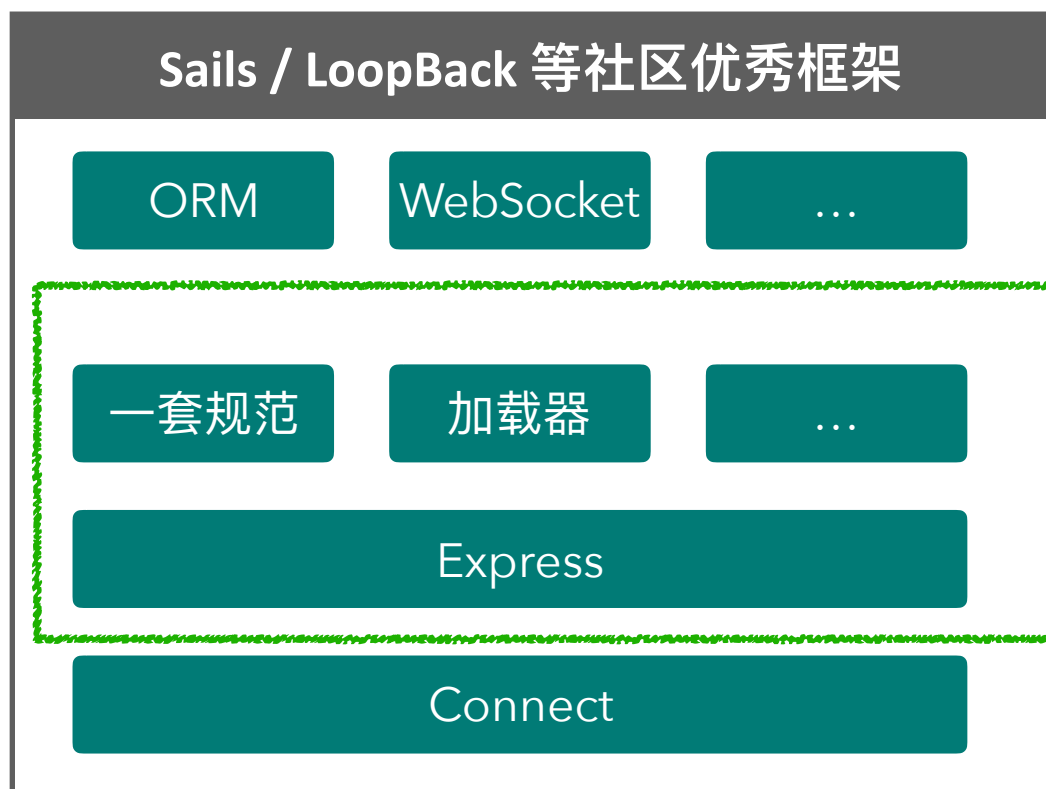
为什么用 Egg ?

- ▶ 其实大家用不用 Egg 的关系不大，Egg 对社区最大的回馈是它的经验沉淀。
- ▶ 但如果你用 Egg，将享受到：
 - ▶ 拥抱社区生态，避免重复造轮子。
 - ▶ Egg 为微内核 + 插件模式，不含阿里内部业务逻辑，很干净，高度可定制。
 - ▶ 渐进式，极具伸缩性，既适合个人小项目快速开发，也适合企业级的团队开发协作。
 - ▶ Egg 的核心开发者是一群资深开源参与者，并拥有大规模企业应用的工程实践经验。
 - ▶ 沉淀自阿里各行各业不同领域最佳实践的插件，涵盖了常见的业务开发场景，稳定支撑了 15 和 16 年天猫双11大促，顶级流量压力。



Egg vs 其他框架

- ▶ 无法直接对比，不是一个层面的概念。
- ▶ Sails / LoopBack 等属于上层框架；Egg 是微内核 + 插件模式，为上层框架服务。
- ▶ 基于 Egg 封装的适合某种业务场景的上层框架，方能直接对比（譬如写个 egg-sails 🤪）。

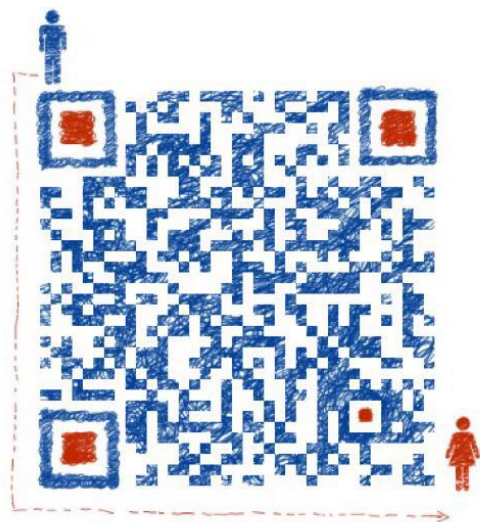


如何参与 Egg

- ▶ 就是一个开源项目而已，没啥特殊的，完全按照社区的方式参与即可。
- ▶ 最简单，试用 Egg，遇到问题尽管给我们提 Issue。
- ▶ 😂 帮我们翻译英文文档，传送门：[English translation](#)
- ▶ 分享你的使用/踩坑心得，只有记录下来的才能成为你真正了解的。
- ▶ 社区没有你需要的插件？很简单，参与进去自己写插件，回馈社区。
- ▶ 封装适合特定业务场景的框架，分享给社区。（骨架 is not enough!）
- ▶ 参与到 Egg 的开发中，解决 Issue，提出新的 RFC 提案并实现。

跨出那一步，参与开源真的没那么难。

说这么多，不如『眼见为实』。



Connect me with WeChat

阿里巴巴 Alibaba
@广州 / 杭州 / 北京 / ...
前端 / 客户端 / 大数据 / 后端

GitHub: <https://github.com/eggjs>

知乎专栏: <https://zhuanlan.zhihu.com/eggjs>

WE ARE JUST **ON** THE WAY



THANK YOU.

