

Dive into Node.js Web Framework

死马



死马 @ everywhere

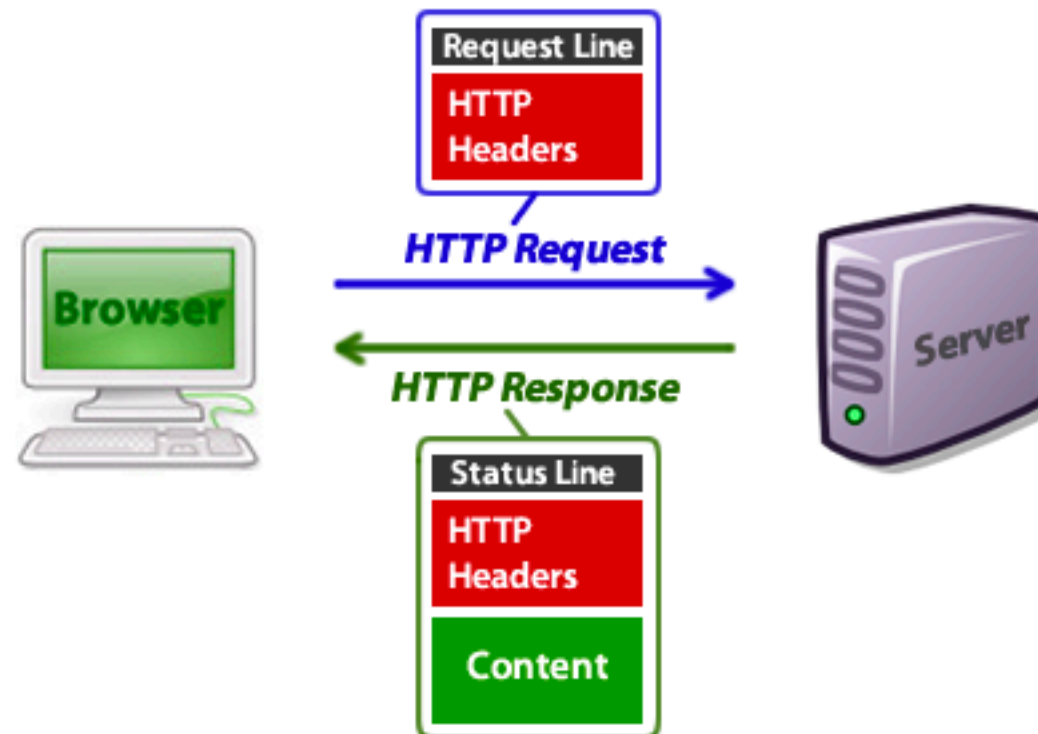
不四 @ 蚂蚁金服

Node.js / Web 开发工程师

Koa.js Egg.js CNPM

HTTP

```
POST /api/posts HTTP/1.1
Host: localhost:3000
Content-Type: application/json; charset=UTF-8
{"title": "hi"}
```



```
HTTP/1.1 201 Created
Content-Type: application/json; charset=utf-8
Content-Length: 8
Date: Mon, 09 Jan 2017 08:40:28 GMT
Connection: keep-alive
{"id": 1}
```

Vanilla Node.js Web Server

```
const http = require('http');
const server = http.createServer((request, response) => {
  response.setHeader('Content-type', 'text/plain');
  if (request.url === '/hello') {
    response.statusCode = 200;
    response.end('world');
    return;
  }
  response.statusCode = 404;
  response.end('not found');
});
server.listen(3000);
```

- ▶ 通过 requestListener 处理请求并发送响应

Vanilla Node.js Web Server

```
const http = require('http');
const server = http.createServer((request, response) => {
  response.setHeader('Content-type', 'text/plain');
  if (request.method === 'POST') {
    const bufs = [];
    request.on('data', data => bufs.push(data));
    request.on('end', () => {
      const data = Buffer.concat(bufs).toString();
      response.end(`receive data: ${data}`);
    });
  } else {
    response.statusCode = 404;
    response.end('not found');
  }
});
server.listen(3000);
```

- ▶ 从 Request 流上获取请求 Body

Vanilla Node.js Web Server

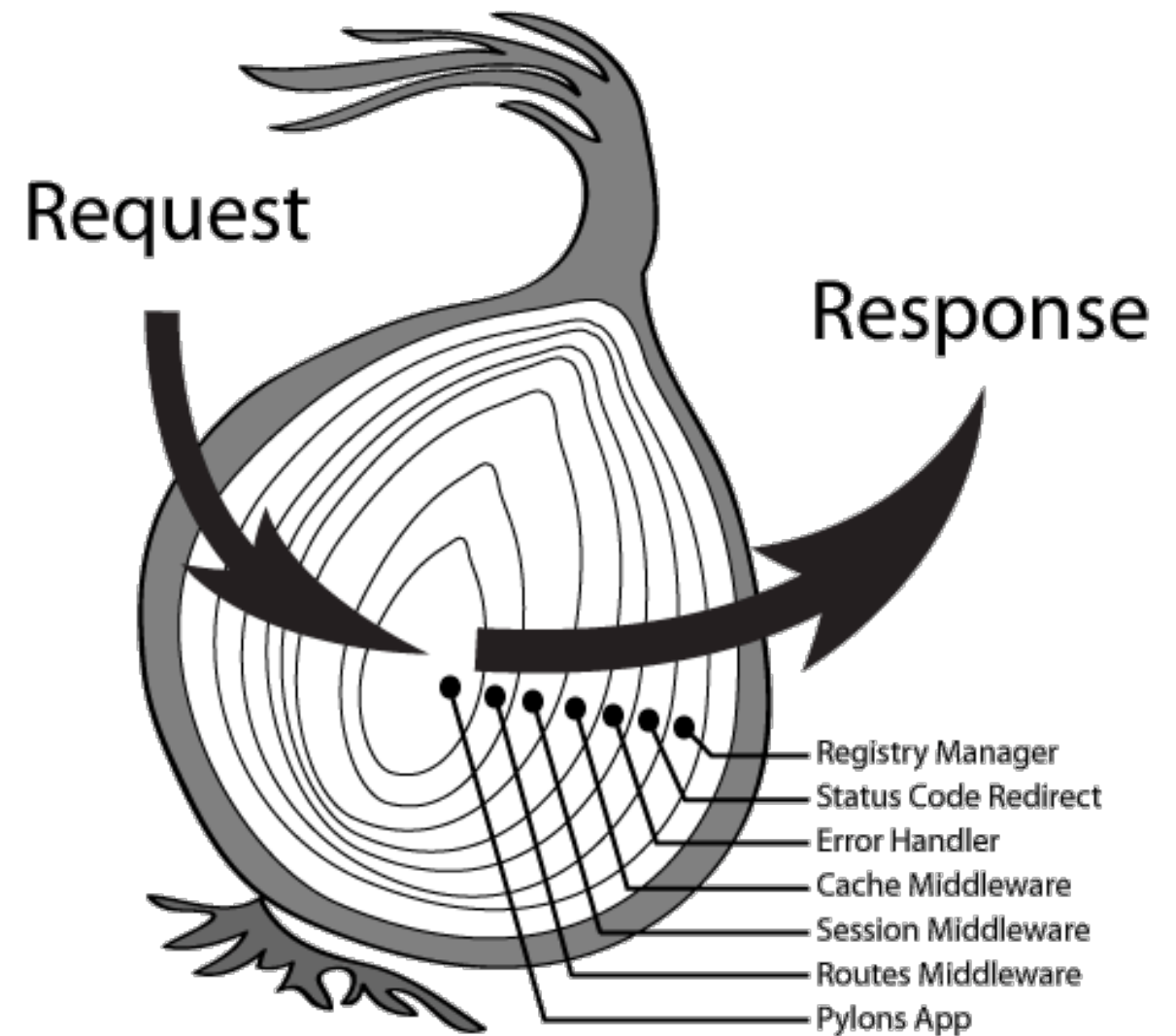
- ▶ 根据请求 Method 和 URL 确定请求的意图
- ▶ 从请求 URL / Header / Body 中读取请求的数据
- ▶ 执行业务逻辑
- ▶ 发送响应数据

Vanilla Node.js Web Server

- ▶ 无法专注于业务逻辑
 - ▶ 根据 Method / URL 手动路由
 - ▶ 获取请求数据有许多细节要处理
- ▶ 按照规范和约定实现 Web Server 并不简单
 - ▶ 缓存、压缩、协商
 - ▶ 会话保持、安全

Web Framework

- ▶ 引入中间件模型
 - ▶ 分离处理请求的不同阶段
 - ▶ 更好的复用通用代码



Web Framework

```
async function compress(ctx, next) {
  ctx.vary('Accept-Encoding');

  await next();

  let { body } = ctx;
  if (!body) return;

  // identity
  const encoding = ctx.acceptsEncodings('gzip');
  if (!encoding) ctx.throw(406, 'supported encodings: gzip');

  ctx.set('Content-Encoding', encoding);
  ctx.res.removeHeader('Content-Length');

  const stream = ctx.body = zlib.createGzip(options);

  if (body instanceof Stream) body.pipe(stream);
  else stream.end(body);
}
```

<https://github.com/koajs/compress/blob/master/index.js>

Web Framework

```
// logger
app.use(async (ctx, next) => {
  await next();
  const rt = ctx.response.get('X-Response-Time');
  console.log(`${ctx.method} ${ctx.url} - ${rt}`);
});

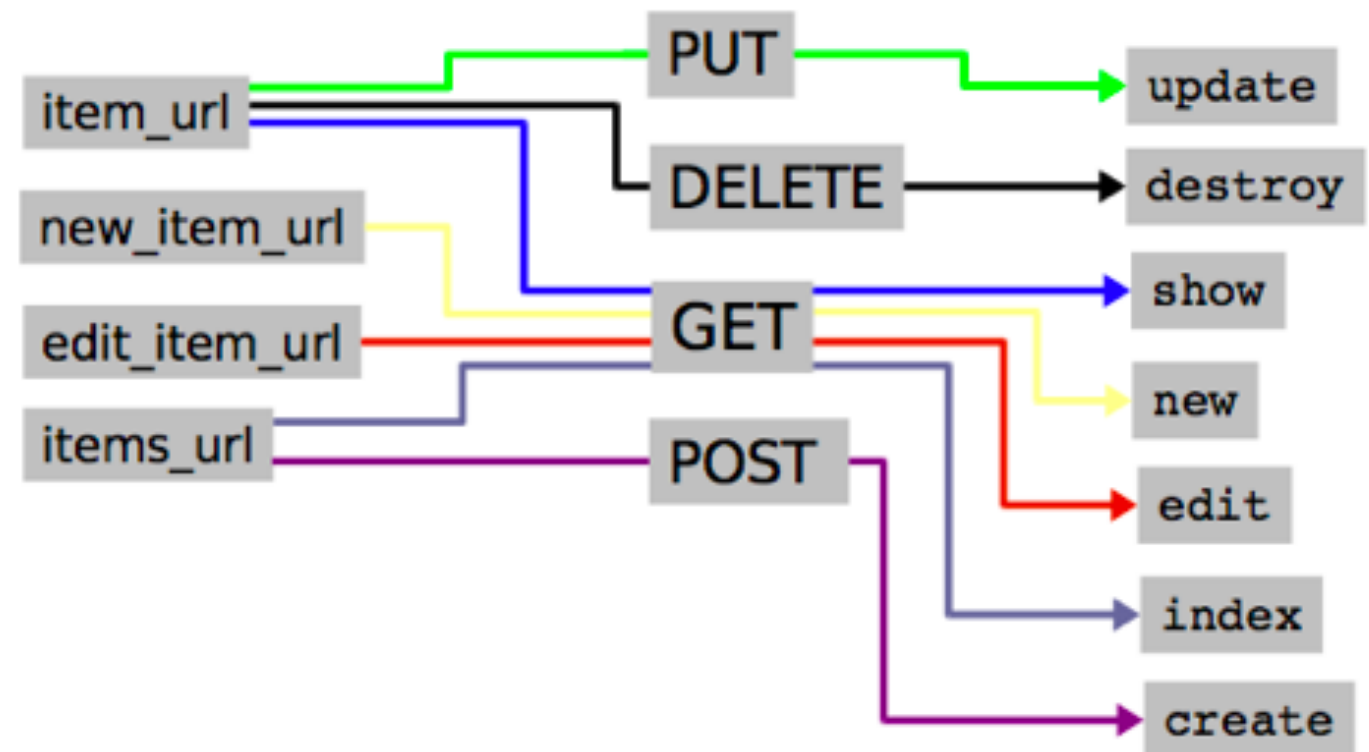
// x-response-time
app.use(async (ctx, next) => {
  const start = Date.now();
  await next();
  const ms = Date.now() - start;
  ctx.set('X-Response-Time', `${ms}ms`);
});

// response
app.use(async ctx => {
  ctx.body = 'Hello World';
});
```

- ▶ 干净的分离出不同阶段的处理代码

Web Framework

- ▶ 实现便捷易用的路由
 - ▶ 识别请求，分发到 Controller
 - ▶ 定义清晰，便于后期维护



Web Framework

```
module.exports = app => {  
  const { router, controller } = app;  
  app.redirect('/', '/news');  
  router.get('/news', controller.news.list);  
  router.get('/news/item/:id', controller.news.detail);  
  router.get('/news/user/:id', controller.news.user);  
};
```

▶ 声明式路由

Web Framework

```
@Controller('cats')
export class CatsController {
  @Post()
  create(@Body() createCatDto) {}

  @Get()
  findAll(@Query() query) {}

  @Get('/:id')
  findOne(@Param('id') id) {}

  @Put('/:id')
  update(@Param('id') id, @Body() updateCatDto) {}

  @Delete('/:id')
  remove(@Param('id') id) {}
}
```

► 装饰器路由

Web Framework

```
// server/controller/post.js
class Controller {
  async list() {
    return posts;
  }

  async create() {
    return post;
  }
}
```

```
// client/proxy/post.js
class Proxy {
  async list(params) {
    return request('post.list');
  }

  async create(params) {
    return request('post.create', params);
  }
}
```

► 自动路由

Web Framework

- ▶ 封装 Request / Response 对象
 - ▶ 快速获取请求参数 (Query, Header, Body, File)
 - ▶ 快速按照约定的数据格式响应数据 (JSON, HTML, Stream)

Web Framework

```
// koa  
ctx.url  
ctx.path  
ctx.query  
ctx.body  
  
ctx.status=  
ctx.body=
```

- ▶ 通过 ctx 对象的 Setter Getter 进行封装

Web Framework

```
// nest
findAll(@Query() query) {
  return posts;
}
findOne(@Param('id') id) {
  return post;
}
@HttpCode(201)
@Header('Cache-Control', 'none')
create(@Body() body) {
  return { success: true };
}
```

- ▶ 通过 decorator 进行封装

Web Framework

- ▶ 提供分离和复用代码的架构
- ▶ 提供便捷易用的路由方案
- ▶ 更高层的封装请求响应对象
- ▶ 让应用代码专注实现业务逻辑

Express

koa

fastify 

用于企业生产环境？

- ▶ 核心太精简，投入生产需要引入大量社区组件
- ▶ 缺乏代码质量、安全性、稳定性保障机制
- ▶ 没有固定的编程模型约束
- ▶ 缺乏内部系统对接方案

Web Framework for Enterprise

编程模型约束

研发效率

可维护性

扩展性

稳定性

易测试

错误处理

安全

故障排查能力

日志

链路追踪

监控

服务对接

跨语言 RPC

前端工程化

中间件

编程模型约束

```
├─ 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.test.js
|   ├── config.prod.js
|   └─ plugin.js
├─ test (单元测试)
├─ README.md
└─ package.json
```

▶ 约定优于配置

▶ Controller / Service / Config / ...

编程模型约束

```
├─ 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.test.js
|   ├── config.prod.js
|   └─ plugin.js
├─ test (单元测试)
├─ README.md
└─ package.json
```

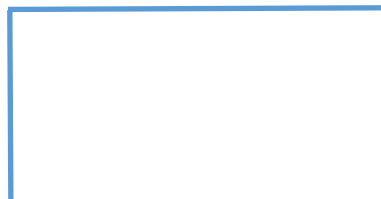
▶ 根据环境自动加载配置

▶ 一次构建，多环境运行

开箱即用的测试方案

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'));
  });
});
```

开箱即用的测试方案

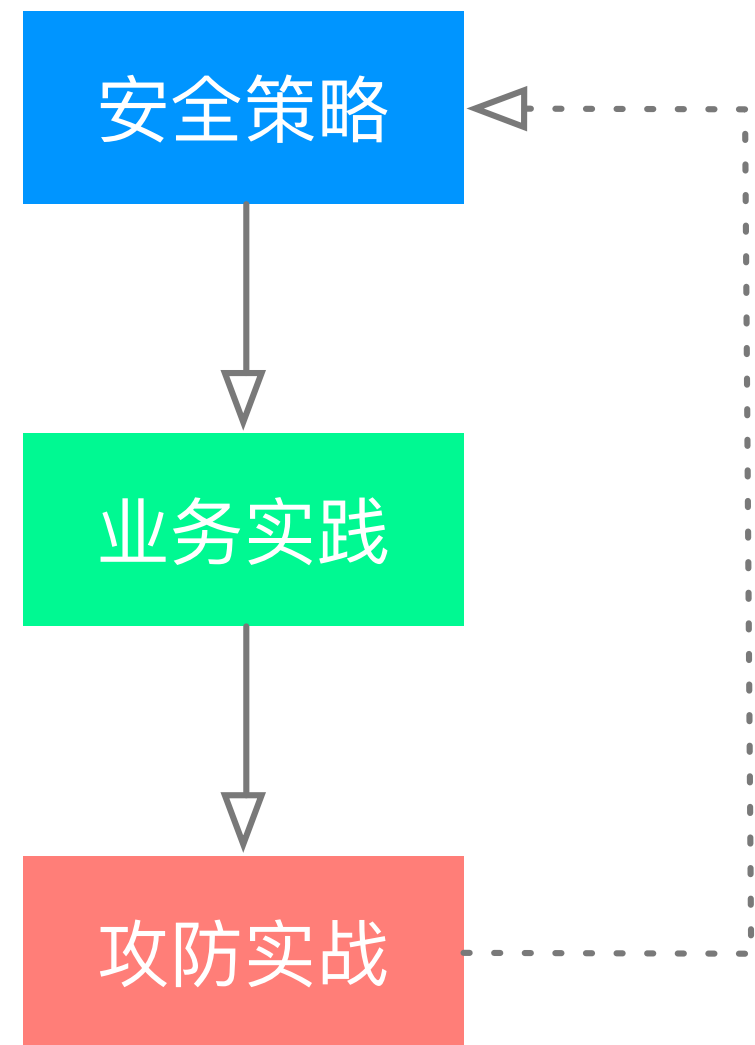
- ▶ Runner 和覆盖率 (mocha + nyc / jest)
- ▶ 断言库 (power-assert / jest)
- ▶ 测试请求库 (supertest)
- ▶ 框架模块的 mock 能力
- ▶ 简化测试用例编写难度, 提升开发者测试意愿

安全

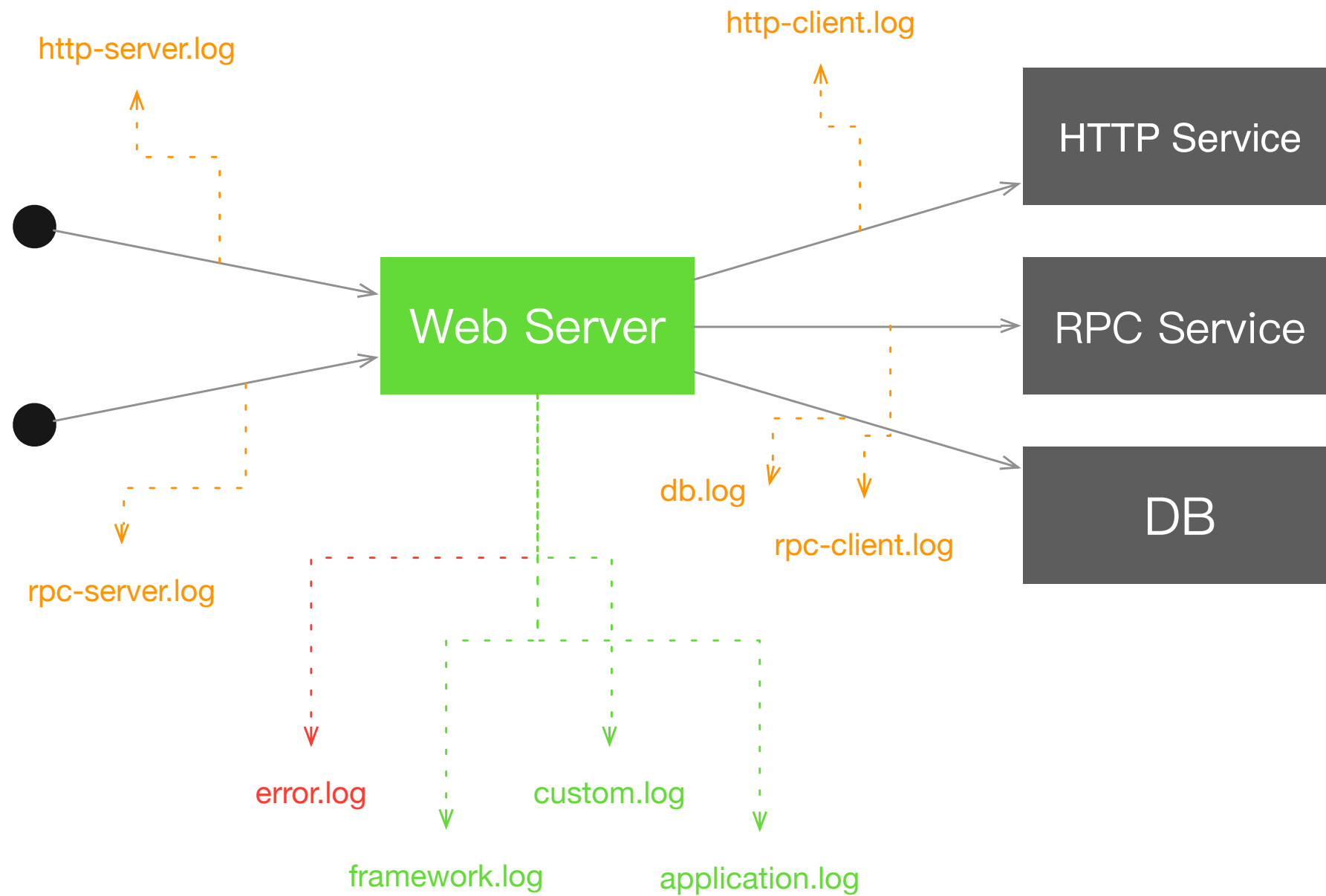
- ▶ 对称/非对称加解密, 加签验签支持
- ▶ 重定向白名单控制
- ▶ csrf 跨站攻击防范
- ▶ ssrf 服务器端请求伪造
- ▶ jsonp 跨站攻击防范
- ▶ xss sanitise: jsonp callback, html, url, js, path, json, cli
- ▶ http security headers: hsts, csp, xssProtection 等
- ▶ 默认禁止 trace track options 请求
- ▶ [egg-security](#), [helmet](#), [lusca](#), [koa-lusca](#)

安全

- ▶ 框架需要优先考虑安全性，同时兼顾易用性
- ▶ 没有银弹，框架无法解决所有安全问题
- ▶ 安全攻防是持久战，需要时刻保持安全意识



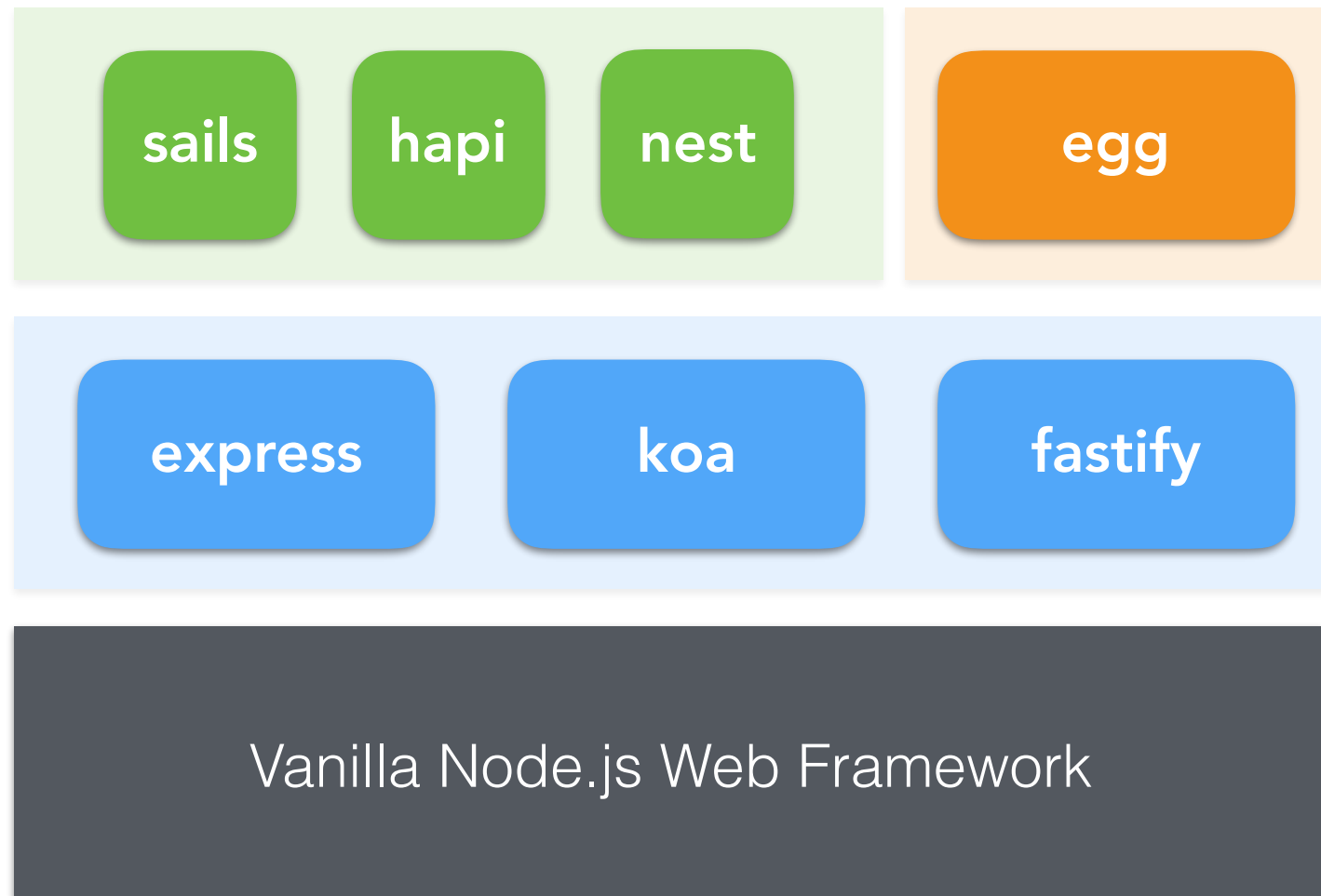
完善的日志记录方案



完善的日志记录方案

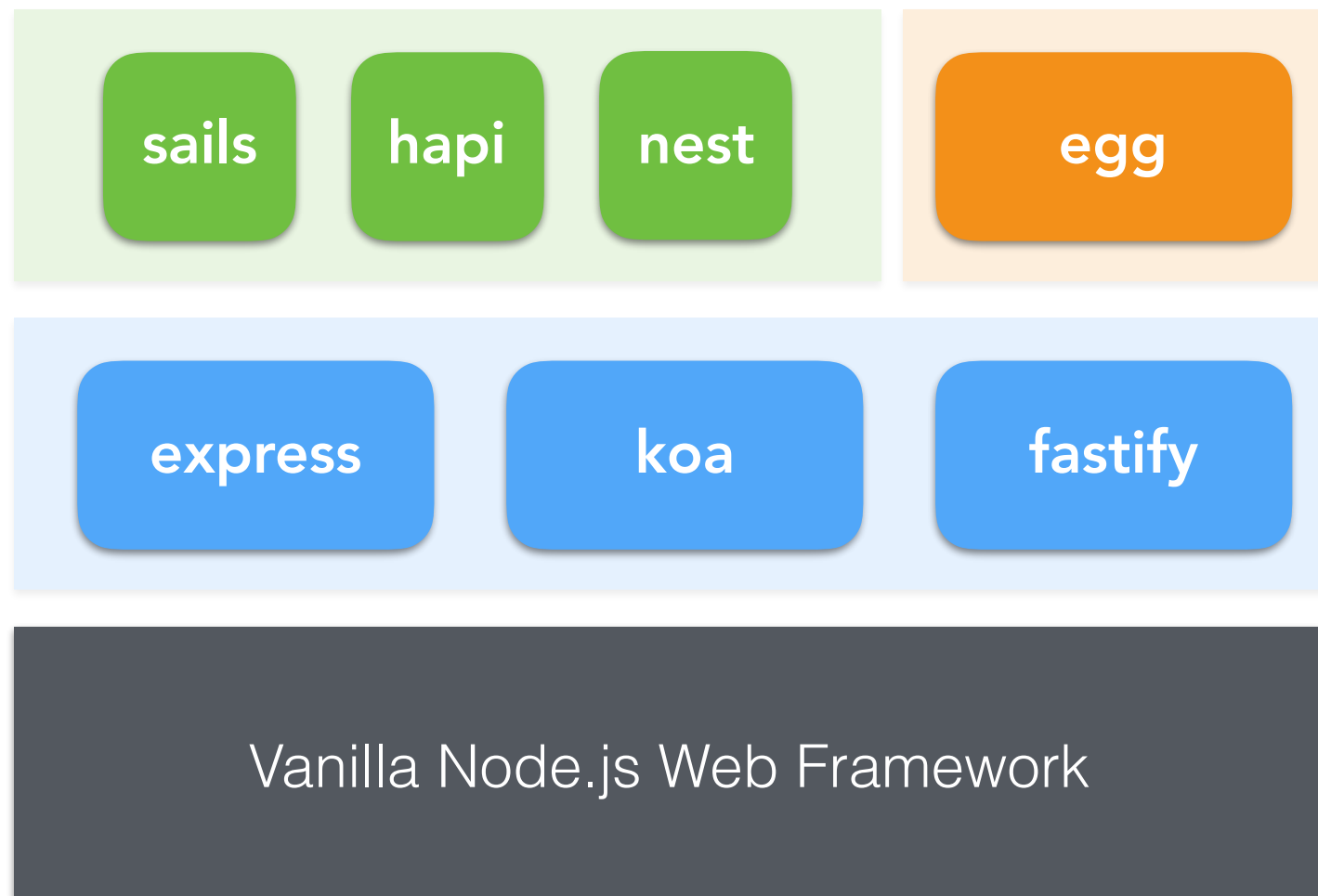
- ▶ 调用链路摘要日志
- ▶ 错误日志统一记录
- ▶ 框架层日志与应用层日志隔离
- ▶ 灵活的自定义日志
- ▶ 按时间、大小自动切割日志文件
- ▶ 支持输出到文件和日志采集系统

Node.js Web Frameworks



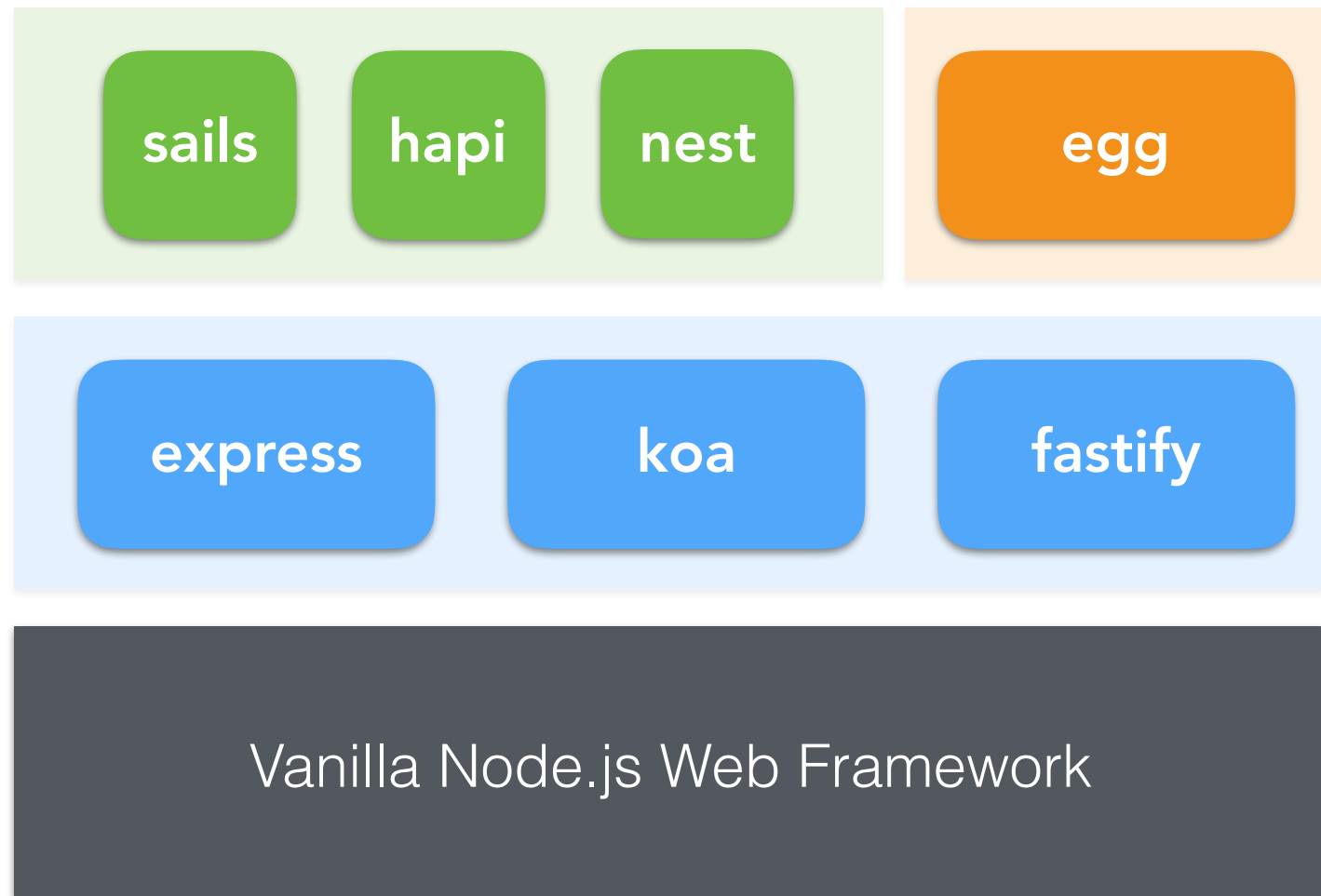
- ▶ 基础框架，核心精简扩展性高
 - ▶ 中间件模型
 - ▶ 封装请求响应对象
 - ▶ 路由方案

Node.js Web Frameworks



- ▶ 富应用框架，更专注架构问题
 - ▶ 解决底层 infrastructure
 - ▶ 提供开箱即用的研发体验
 - ▶ 开发者专注于业务逻辑

Node.js Web Frameworks



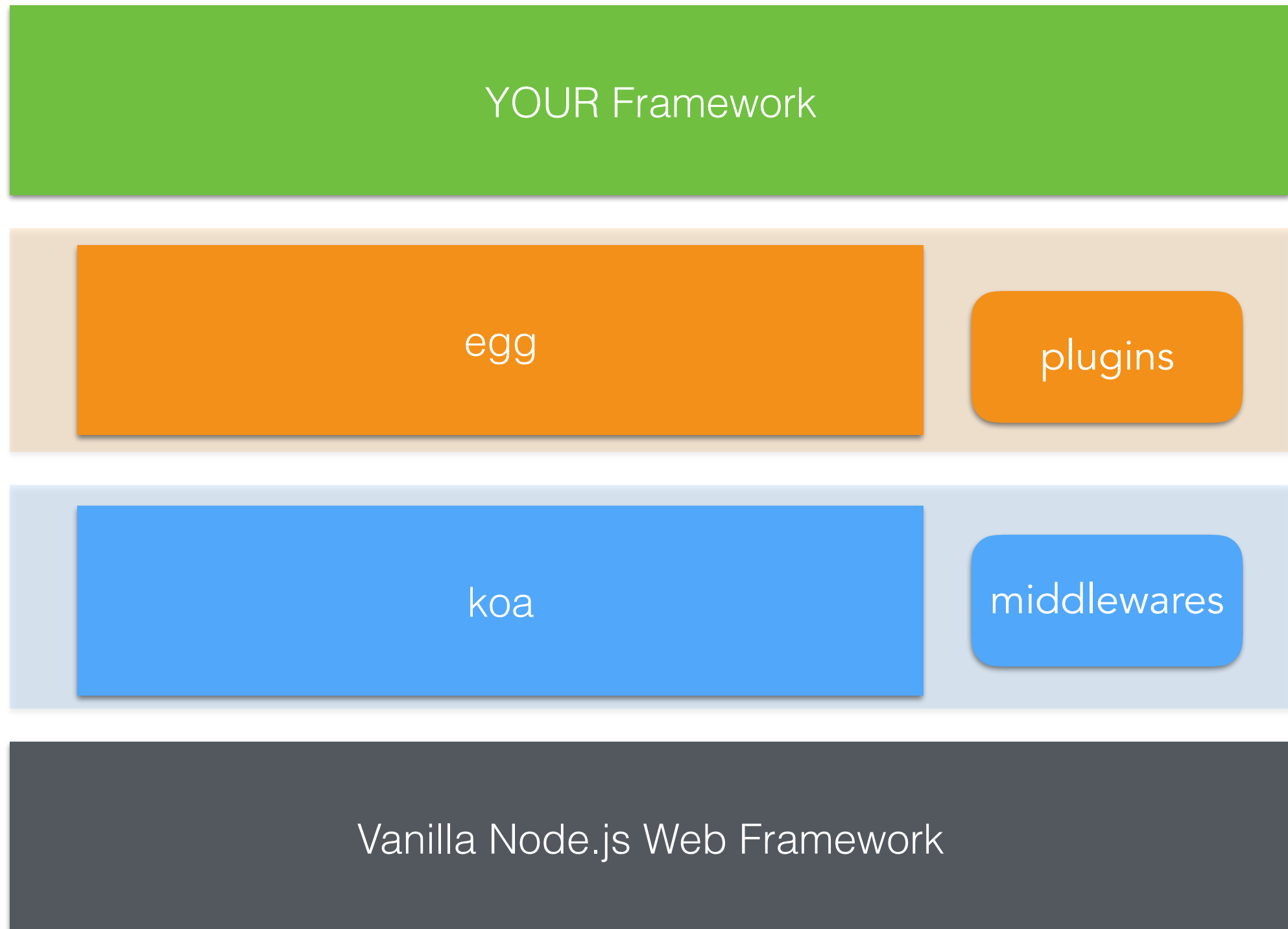
► 框架的框架

- 约定统一的编程模型
- 灵活可扩展的插件机制
- 适合团队定制化自己的框架

Web Framework for YOU

- ▶ 提供开箱即用的研发体验
- ▶ 统一团队编程模型，让开发者更好的云动
- ▶ 接入企业内复杂的技术方案和服务
- ▶ 让产品工程师可以专注于产品

Web Framework for YOU



高度可扩展与插件化

- ▶ 插件是一个『迷你应用』
 - ▶ 扩展框架的内置对象
 - ▶ 插入自定义中间件
 - ▶ 设置定时任务
 - ▶ 封装业务逻辑
 - ▶ 自定义初始化
- ▶ 从中间件到插件，更好的实现逻辑分离

```
├── app
│   ├── middleware (中间件)
│   │   └── auth.js
│   ├── schedule (定时任务)
│   │   └── clean_temp_file.js
│   ├── extend (扩展对象)
│   │   ├── context.js
│   │   ├── request.js
│   │   ├── resposne.js
│   │   └── application.js
│   └── service (业务逻辑)
│       └── user_center.js
├── config (配置)
│   ├── config.default.js
│   ├── config.test.js
│   ├── config.prod.js
│   └── plugin.js
├── app.js (自定义初始化)
└── package.json
```

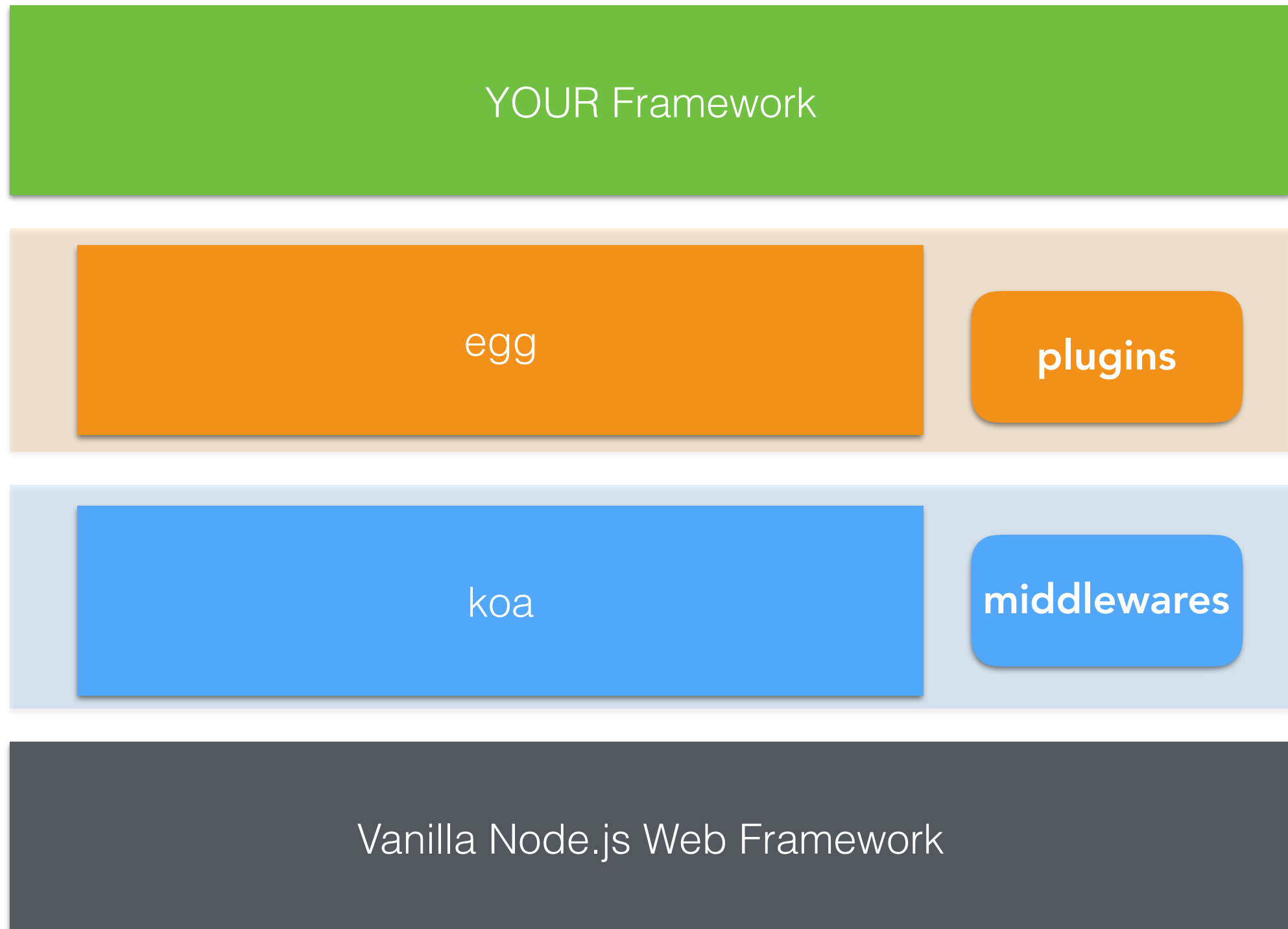
插件组合成框架

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



- ▶ 『框架』是对适合 特定业务场景 的 最佳实践 的约束和封装
 - ▶ 统一技术选型，数据库、模板、前端框架及各种中间件设施
 - ▶ 统一部署和运维和devops 方案
 - ▶ 统一的编码模型和代码风格

Web Framework for YOU



Web Framework for YOU

```
async update() {
  const { ctx } = this;
  ctx.requireUser();
  // 参数校验
  const params = ctx.permitAndValidateParams({
    id: 'int',
    title: 'string',
    body: 'string?',
    book_id: 'int',
  });
  // 权限校验
  const doc = await ctx.model.Doc.findById(id);
  ctx.authorized('update', doc);
  // 调用 service 完成业务逻辑
  const doc = ctx.service.doc.updateDoc(doc, params);
  // 响应
  ctx.success(ctx.serialize('web.doc', doc));
}
```

► 语雀的 Controller 实践

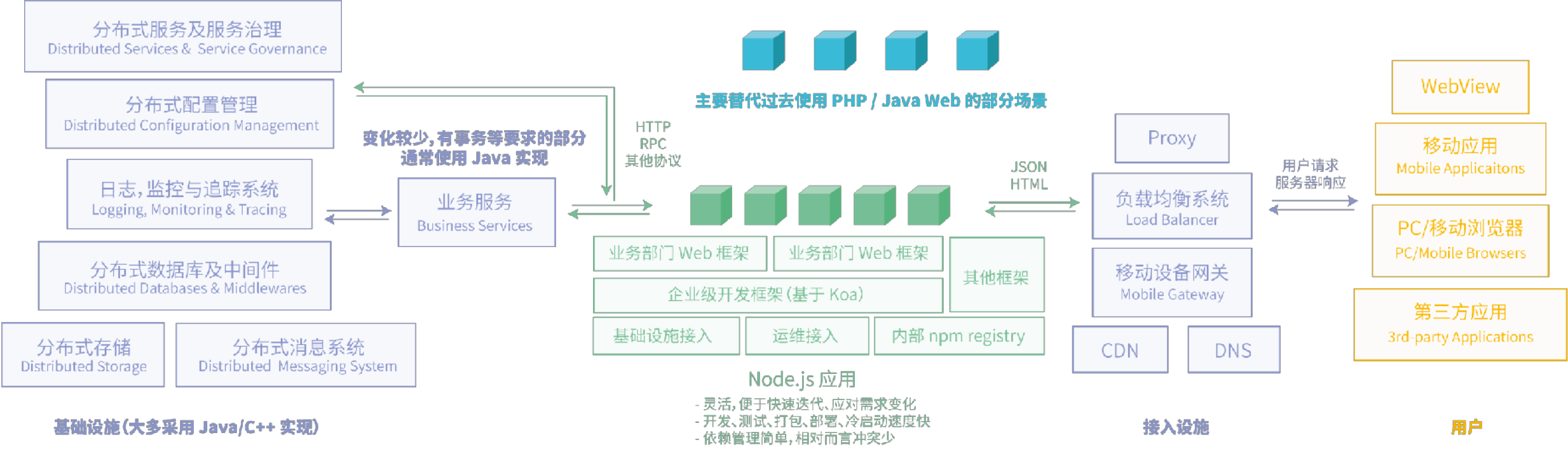
Web Framework for YOU

```
export class HomeController {  
  @route('/api/xxx', { name: '获取xxx数据' })  
  async getXXX(data: BaseValidateRule<{  
    type: 'object',  
    rule: {  
      str: { type: 'string', max: 20 },  
      count: { type: 'number', max: 10, required: false },  
    },  
  }) {  
    return data.str;  
  }  
}
```

► 云凤蝶的 Controller 实践

Web Framework in Alibaba

阿里的 Node.js 应用场景



<http://alinode.aliyun.com/blog/37>

Web Framework in Alibaba

一线开发者

基于上层框架开发的应用

团队架构师

适合特定团队业务场景的上层框架 Framework

蚂蚁 Chair

UC Nut

Midway

...

Plugin
插件生态

Tool
工具链

开源社区生态

基于规范实现一套框架 - Egg

Specification 一套规范和约定

Koa

Node.js

感谢聆听