去哪儿网 Node.js 实践及性能监控方案

兴百放

2018.06

React从入门到精通

掌握当下最热门的前端利器

王沛
eBay 资深技术专家

你将获得

1. 全面学习 React 常用技术栈
2. 深入理解 React 设计模式
3. 常见场景下的编程实战指南
4. 掌握用 React 开发大型项目的能力

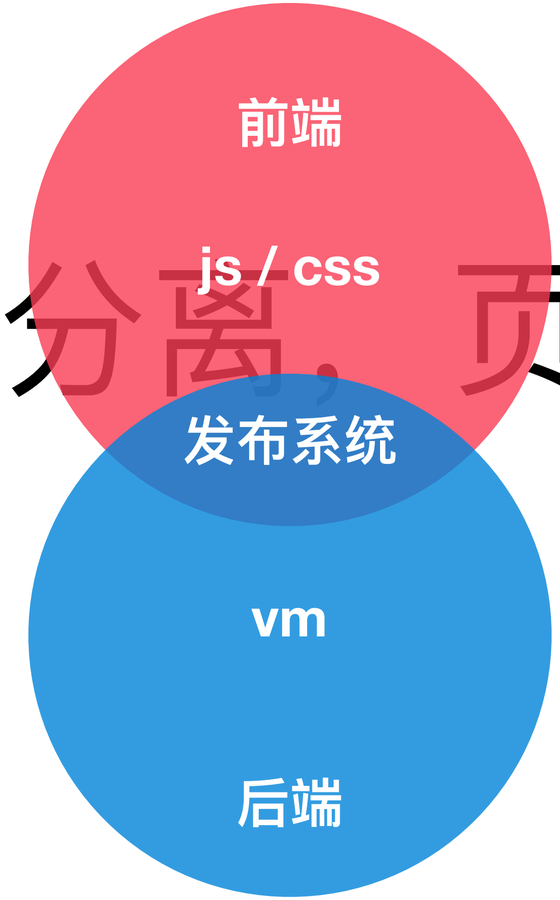
立即扫码，免费试看

TABLE OF

CONTENTS 大纲

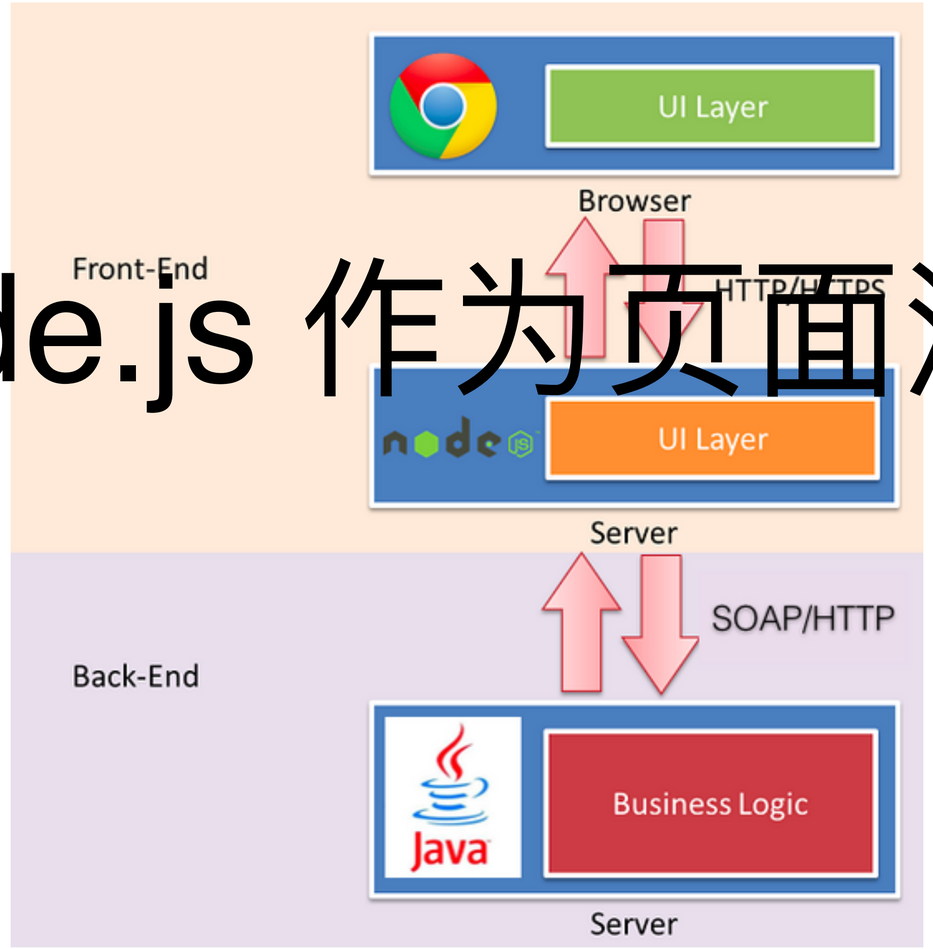
- 去哪儿网前后端分离方案和静态资源离线包
- Node.js 和 React SSR 实践
- 应用和性能监控方案

1. 项目分离，页面分离

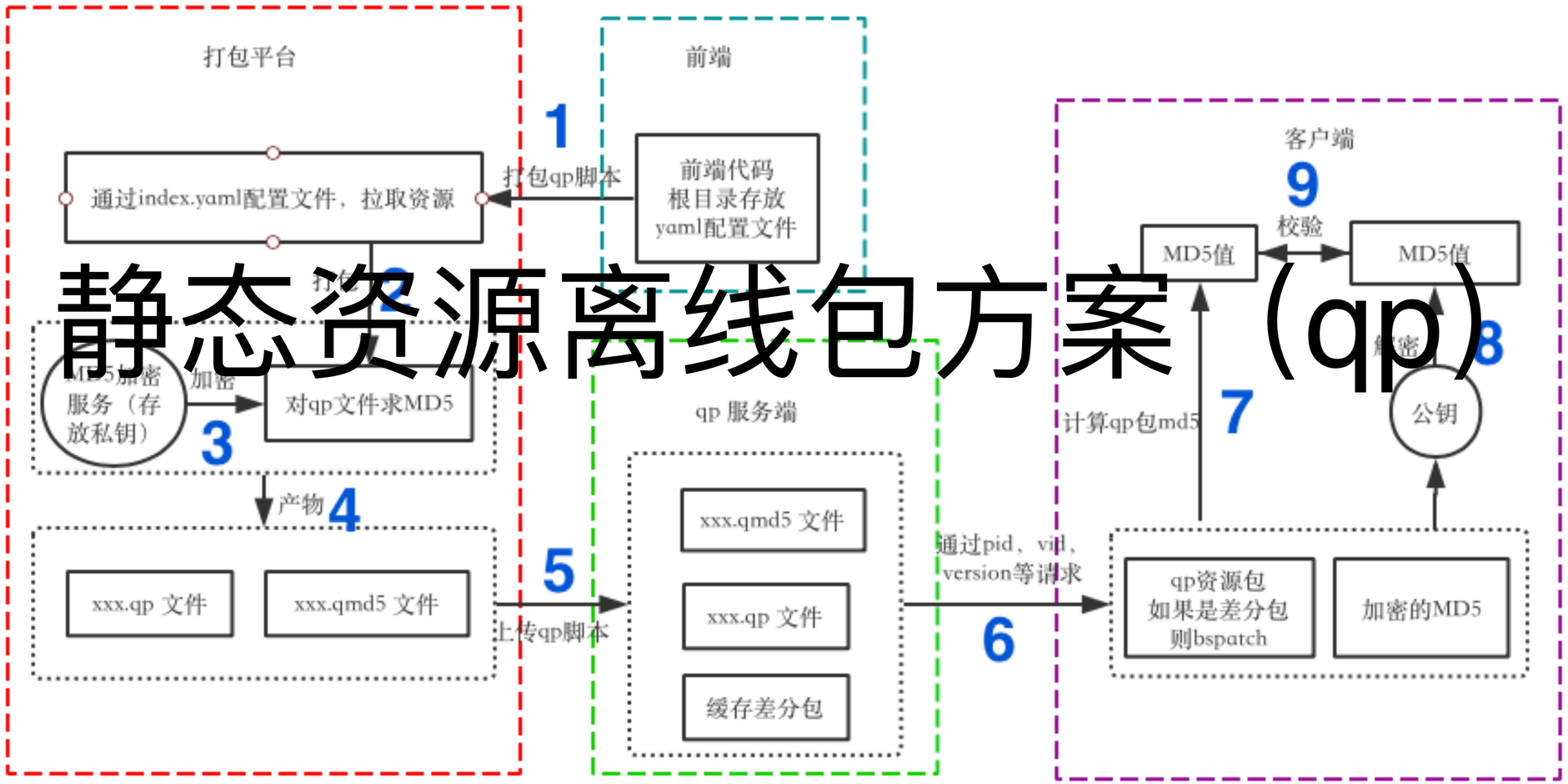


2. 项目分离，部分页面分离

3. Node.js 作为页面渲染层



静态资源离线包方案（qp）



项目中如何使用 qp 包

- 在项目根目录中，新建 index.yaml 配置文件
  - 唯一id
  - 针对的 iOS 和 Android 版本
  - 打包内容
  - 忽略内容
- 使用打包平台发布，无需人工干预

用户对离线包完全透明，且无感知。

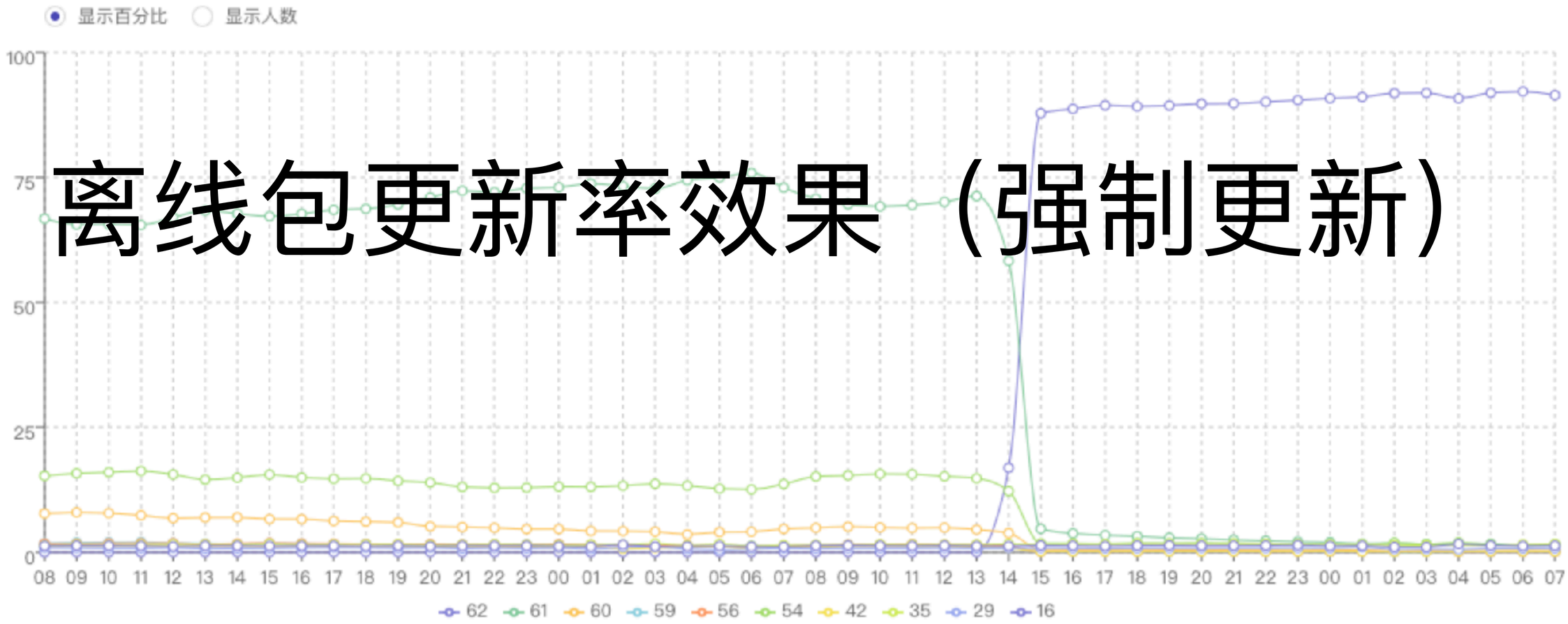
需要考虑的问题

- **保证资源的安全性，不被中间人恶意篡改**
  - 传输安全和存储安全，使用 RSA 加密
- **快速回滚**
  - 原来假回滚（重新发版） -> 真回滚
- **下线和强制更新**
  - 下线：针对当前包
  - 强制更新：将要下载的包
- **提高更新率**
  - 减少 qp 包的大小
  - 使用 HTTP2 协议
  - 使用差分包而不是全量包

http://ued.qunar.com/qp/

强制更新能在 2 小时内达到 90%

离线包更新率效果（强制更新）



普通更新通常7，8个小时后才能稳定在75%左右

离线包更新率效果（普通更新）

第一部分总结

- 去哪儿网三种前后端分离方案，以及各种方案的优缺点和适用情况
- 简要介绍静态资源离线包实现方式和部分细节

TABLE OF

CONTENTS 大纲

- 去哪儿网前后端分离方案和静态资源离线包
- Node.js 和 React SSR 实践
- 应用和性能监控方案

为什么没有大规模使用?

- 一些前端开发，只关注浏览器端，服务器端开发关注很少，或者根本就不关注;
- 认为 Node.js 只适合开发一些工具类的功能，对于后端开发是个玩具;
- Node.js 的生态不如其他后端语言生态健全;
- 涉及到后端开发的知识面比较广，在没有这些基础知识或者经验积累的基础上，考虑问题比较片面，最终做出的系统问题比较多，容易被后端鄙视;
- 对于 Node.js 开发后端，对项目负责人要求比较高（项目的目录规范，开发规范，系统的安全性，稳定性，可靠性，扩展性，维护成本等）;
- 以往前端不需要 7 x 24 保持待命状态，但是接触后端后，需要接收报警短信，有时出现问题还需要马上随时随地解决;

Node.js 能解决哪些问题和痛点？

- 提高开发效率 （不用 Nginx ， 代理工具等等）
- 降低沟通成本（除接口格式外，不需要和后端交互）
- 前后端职责更新清晰 -> 后端生产数据，前端消费数据
- 可以使用 React SSR 技术 -> 首屏渲染、SEO 等
- RESTful API -> 自身使用或对外提供，不依赖后端

三年前所用框架的问题（基于express）

1. 如何确定项目目录划分的规范，命名规范（view or views）

2. 确定规范后，如何保证大家都认可，并且严格遵守

3. 如何保证系统的安全性，稳定性，可扩展性

4. 守护进程程序的选择（pm2 or supervisor）

5. 多环境运行规则（local / beta / prod）

6. 如何利用系统 cpu 多核，以及多进程之间的通信

新选择框 or

在文档说明、框架扩展、插件数量、部署流畅性以及开发体验上，最终选择的是 Egg.js。

插件开发

- egg-qversion
- egg-qconfig
- egg-qwatcher
- egg-accesslog
- egg-healthcheck
- egg-checkurl
- egg-swift

原来部署流程（service 方式）问题

1. 不能利用发布系统中相应的端口和目录字段，只能在 qunar_xx 服务中写死，不友好

2. 不能区分多环境策略，比如 beta 环境和 prod 环境配置规则不一样

3. 启动过程中出现错误，不方便定位问题，需要到机器上排查

4. 写系统服务需要了解 shell 命令和系统服务格式，对于前端开发同学，成本稍高

5. 除了端口、项目路径、运行环境，node.js 启动方式外，处理逻辑相似

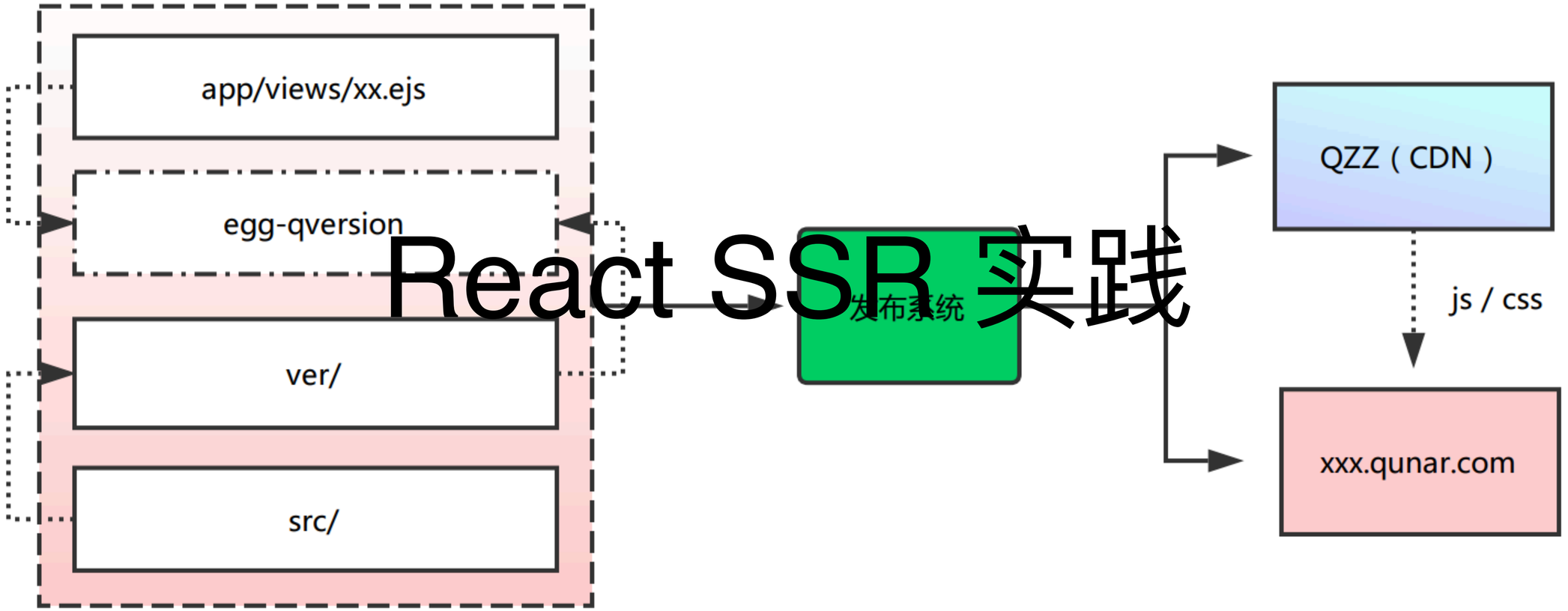
改进后的部署方式

- 在项目中建立 deploy_scripts 目录，新增 start.sh （名称可以随便命名）
- 在 start.sh 中填入 Node.js 启动逻辑，比如 node index.js (之前是 N 行，如今最多两行)
- 在发布系统选择 node 发布方式，填入端口号，发布路径，以及启动脚本名称（start.sh），停止脚本填入发布系统内置的 stop.sh （按照端口杀掉进程）

start.sh 样例

```
1   # 必须先进入到目录中
2   echo "进入到项目目录 $deploy_dst"
3   cd $deploy_dst
4
5   # 安装依赖
6   echo "yarn安装依赖包 yarn install --production"
7   yarn install --production
8
9   # 设置运行环境
10  APP_ENV="${deploy_type}"
11
12  # 启动node
13  echo "start node, 执行命令: EGG_SERVER_ENV=$APP_ENV PORT=$app_port yarn run start"
14  NODE_ENV=production EGG_SERVER_ENV=$APP_ENV PORT=$app_port yarn run start
```

React SSR 实践


app/views/xx.ejs
egg-qversion
ver/
src/
发布系统
QZZ（CDN）
js / css
xxx.qunar.com

Action 写法

```
5  const configureStore = require('../../src/util/configureStore').default;
6  const StorePage = require('../../src/page/prize201806Store/home').default;
7  const { reducer: storePageReducer, actions: storePageActions } = require
   ('../../src/page/prize201806Store/model');
8
9  class Prize201806 extends Controller {
10     async store(ctx) {
11         const qconfig = (await co(ctx.helper.getQConfig
           ('prize201806-store.json'))) || {};
12
13         const store = configureStore(storePageReducer);
14
15         await store.dispatch(storePageActions.fetchAllStore(qconfig));
16         ctx.body = await ctx.helper.reactRender('prize201806/store.ejs',
           StorePage, store, '门店选择');
17     }
18 }
19
```

reactRender 方法

```
async reactRender(viewPath, mainComponent, store, title = '默认模板
Title') {
    if (!store) {
        throw '请为页面填入reduxStore! ';
    }
    const reactDOM = ReactDOMServer.renderToString(
        React.createElement(
            Provider,
            { store },
            React.createElement(mainComponent, { Sniff: this.ctx.Sniff |
            | this.getClientDeviceInfo() })
        )
    );
    return await this.ctx.renderView(viewPath, {
        title,
        reactDOM,
        initialState: JSON.stringify(store.getState())
    });
},
```

视图（xx.ejs）

```
5       <link rel="stylesheet" href="<%- ctx.app.config.qunarzzPath %>/<%-
        ctx.app.config.qunarzzResourceId %>/prd/page/prize201806Store/index@<%-
        ctx.loadVersion('self-server', 'page/prize201806Store/index.css')
        %>.css" />
6   </head>
7   <body>
8       <div class="yo-root" id="app"><%- reactDOM %></div>
9       <script>
10          window.__INITIAL_STATE__ = <%- initialState %>;
11      </script>
12      <script type="text/javascript" src="<%- ctx.app.config.qunarzzPath
        %>/<%- ctx.app.config.qunarzzResourceId %>/prd/manifest@<%-
        ctx.loadVersion('self-server', 'manifest.js') %>.js"
        crossorigin="anonymous"></script>
```

前端 （xx.js）

```
12  import { actions } from './model';
13
14  import './index.scss';
15
16  @autobind
17  class Home extends Component {
18      constructor(props) {…
26      }
27      onItemTap(item) {…
32      }
33      onPressGoTop() {…
38      }
39      onScroll(y) {…
41      }
42      filterStore(allData, inputValue) {…
54      }
55      renderContent(item, index) {…
69      }
70      render() {…
116     }
117 }
118
119 export default connect((state) => {
120     return {
121         store: state
122     };
123 }, actions)(handleServerSniff(Home));
```

React SSR 遇到的问题

- **CASE1** 共享代码如何处理请求

```
8
9   if (isEnvNode) {
10      axios.defaults.baseURL = `http://127.0.0.1:${process.env.PORT}`;
11  }
12
```

```
34      // 我的红包列表
35      async my(ctx) {
36          const store = configureStore(myPackReducer);
37          await store.dispatch(myPackActions.getList(ctx.helper.headerForAxios(), ctx));
38          ctx.body = await ctx.helper.reactRender('splitRedPack/my.ejs', RedPackList, store, '去哪儿旅
            行');
39      }
40  }
```

React SSR 遇到的问题

- **CASE2** 共享代码如何处理错误

```
17      getList (headers, ctx) {
18          return dispatch => {
19              return axios({…
22              }).then(data => {…
30              }).catch(err => {
31                  const { status } = err;
32                  if (status === 10001) {
33                      const url = `/splitredpack/index.htm?activityId=$
                        {ajaxCommonData.activityId}`;
34                      if (ctx) {
35                          ctx.redirect(url);
36                      } else if (window && window.location) {
37                          window.location.href = url;
38                      }
39                  }
40              });
41          };
42      }
43  };
```

React SSR 遇到的问题

- **CASE3** 后端代码获得设备信息

```
function handleServerSniff(MainComponent) {
    class SniffComponent extends Component {
        constructor(props) {
            super(props);

            this.Sniff = isEnvNode ? this.props.Sniff : Sniff;
        }
        getChildContext() {
            return {
                Sniff: this.Sniff
            };
        }
        render() {
            return <MainComponent {...this.props} Sniff={this.Sniff} />;
        }
    }
    SniffComponent.childContextTypes = {
        Sniff: PropTypes.object.isRequired
    };
    return SniffComponent;
}
export default handleServerSniff;
```

第二部分总结

- 简要分析 Node.js 没有被广泛使用的原因
- Node.js 解决了哪些问题和痛点
- 出于什么考虑选择 Egg.js 以及开发一系列比较实用的插件
- Node.js 发布方式的演进，大家可以借鉴借鉴
- React SSR 实践思路，以及一些比较典型的问题

TABLE OF

CONTENTS 大纲

- 去哪儿网前后端分离方案和静态资源离线包
- Node.js 和 React SSR 实践
- 应用和性能监控方案

性能监控

对框架本身没有做太多，因为 eggjs 本身已经经历过淘宝双十一的洗礼，相信在阿里内部对这块已经做的不少优化，所以简单使用的是公司级别的机器监控。

应用监控

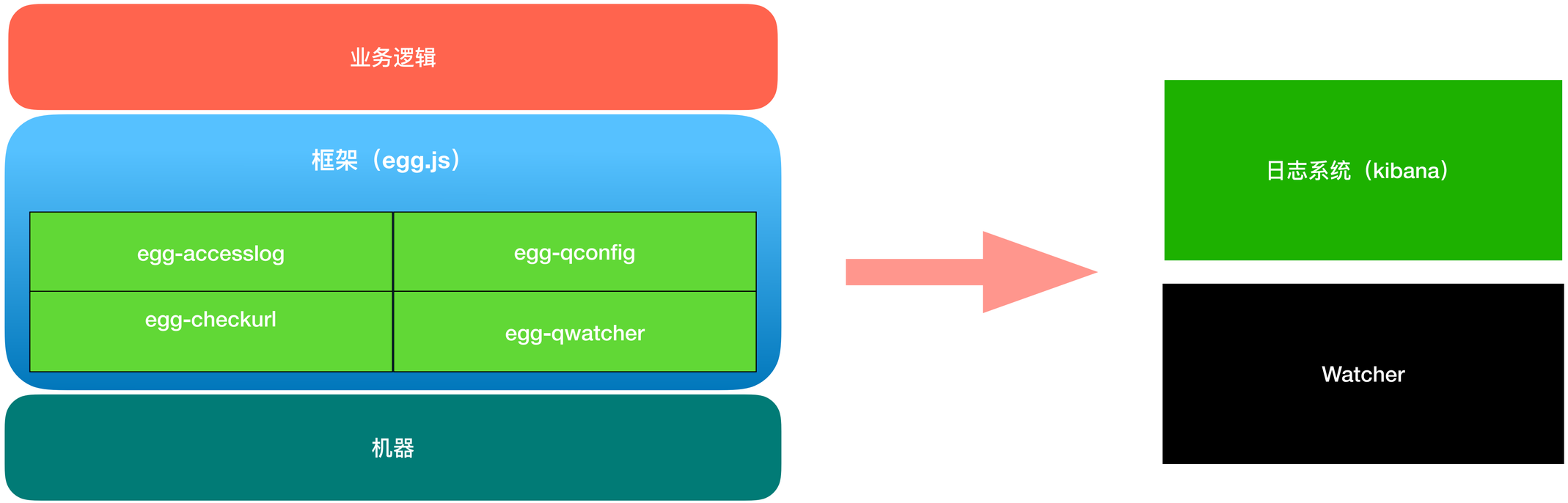
应用程序级别

应用错误数、接口消耗时长、
接口异常信息、accesslog 等

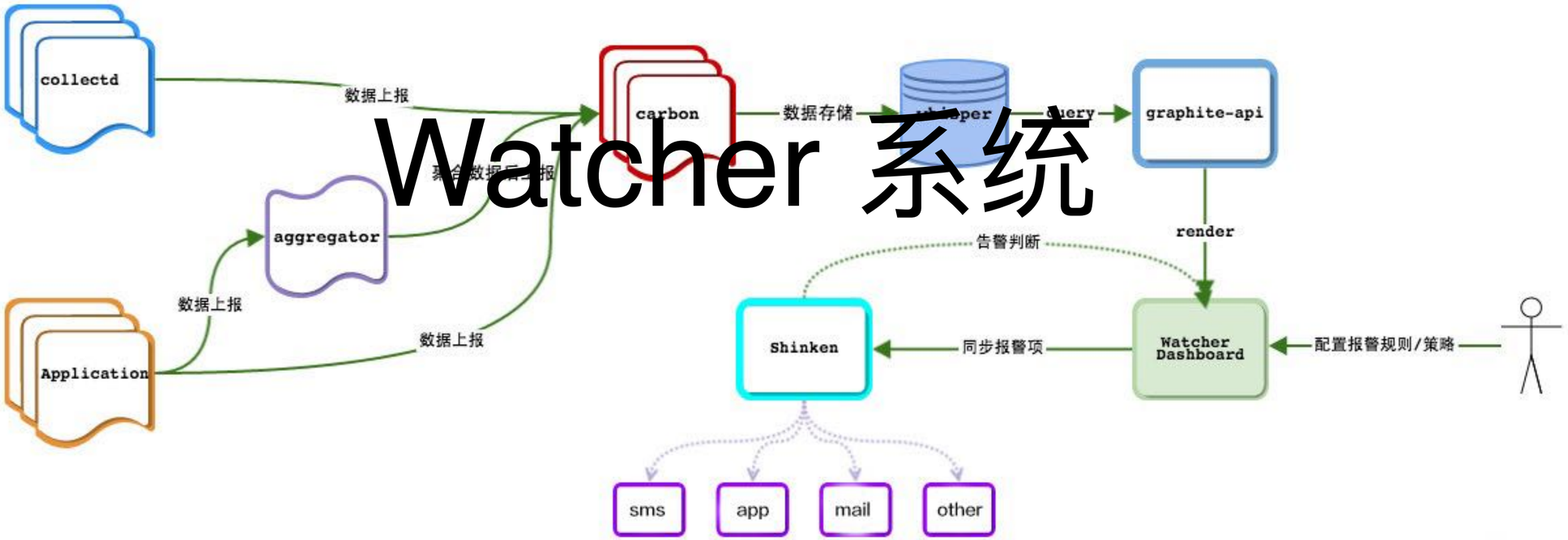
前端页面级别

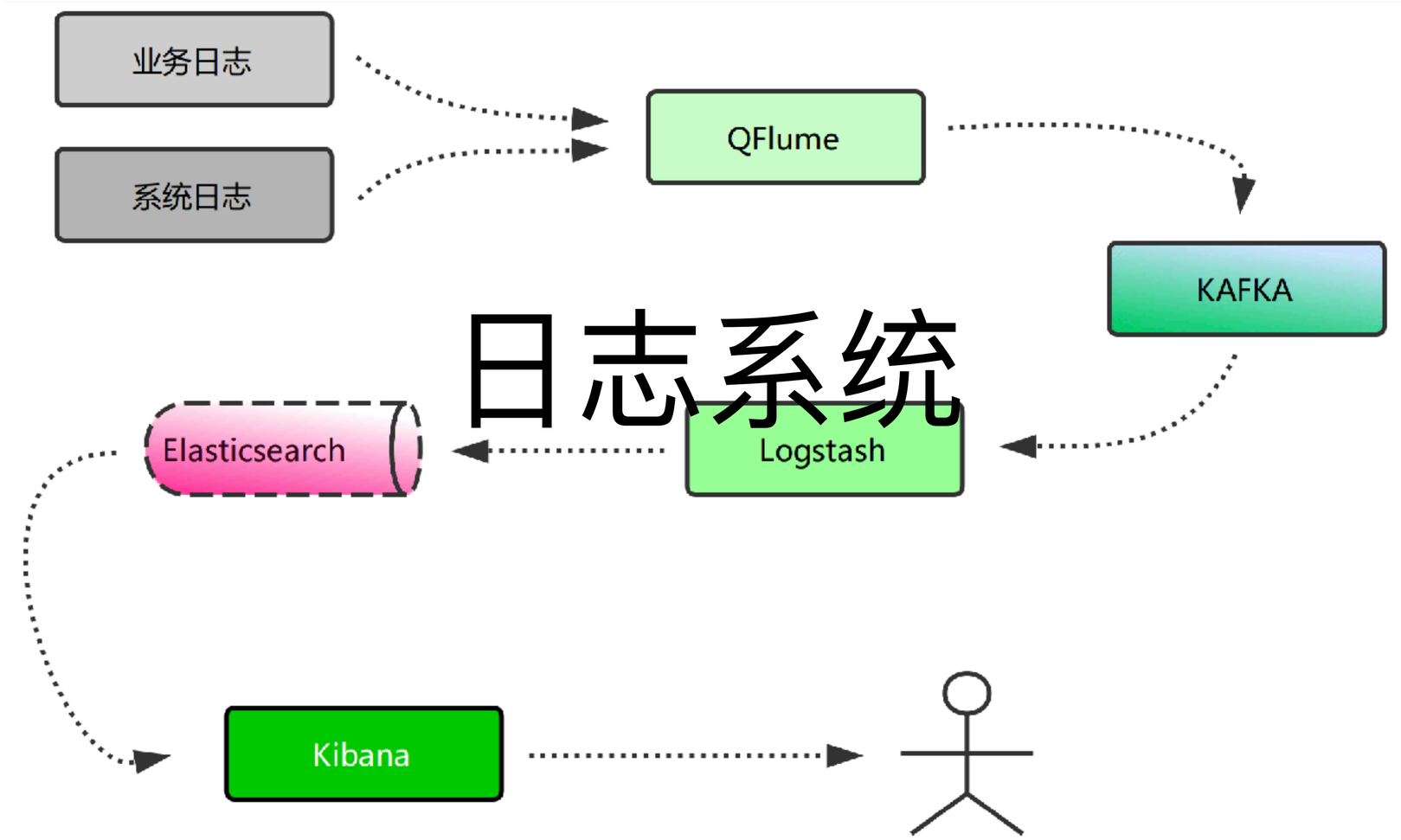
脚本全局错误、静态资源
文件加载错误、异步接口
错误、页面渲染时长等

应用监控

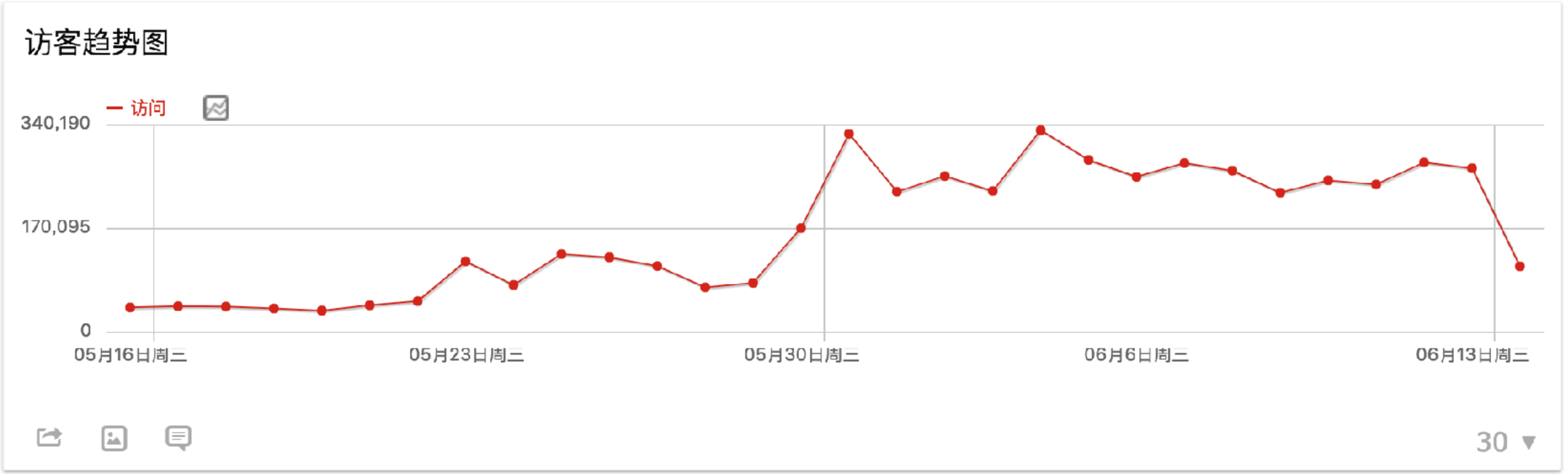

业务逻辑
框架（egg.js）
egg-accesslog
egg-qconfig
egg-checkurl
egg-qwatcher
机器
日志系统（kibana）
Watcher

Watcher 系统

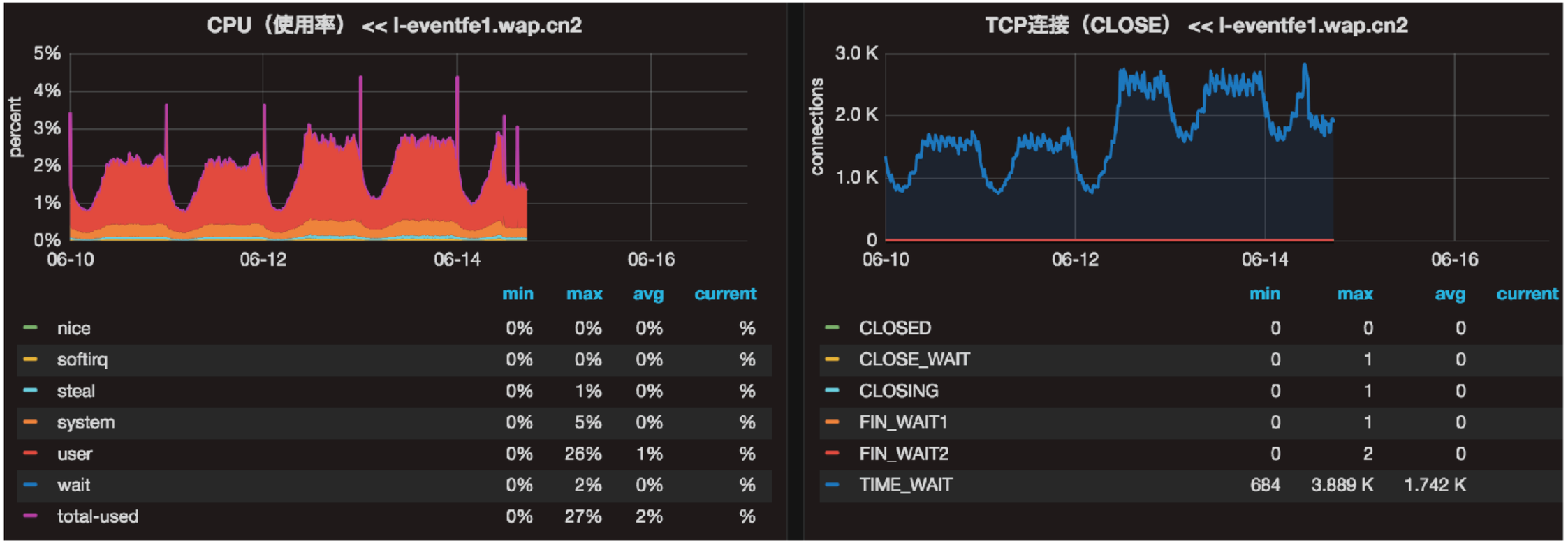

collectd
数据上报
carbon
数据存储
graphite-api
聚合数据后上报
aggregator
render
告警判断
数据上报
Application
数据上报
Shinken
同步报警项
Watcher
Dashboard
配置报警规则/策略
sms
app
mail
other


业务日志
系统日志
QFlume
KAFKA
日志系统
Elasticsearch
Logstash
Kibana

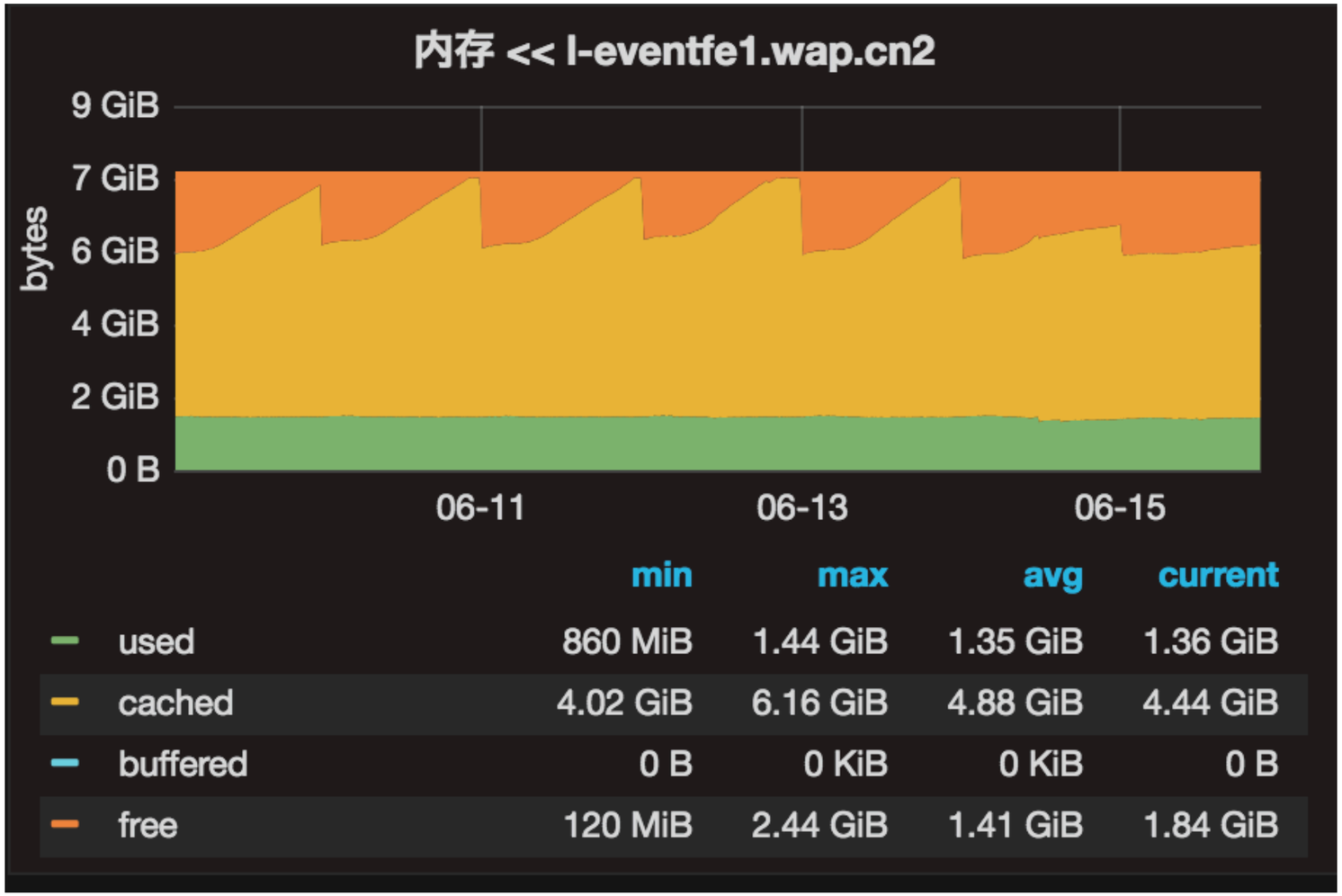
举个栗子（3台4C-8G虚机）



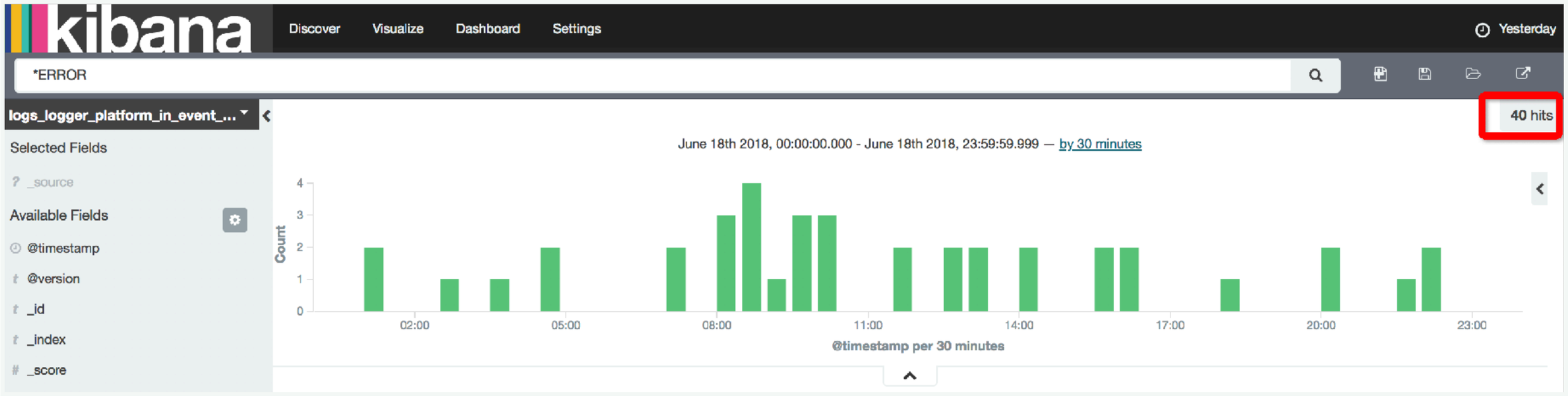
举个栗子（3台4C-8G虚机）



举个栗子（3台4C-8G虚机）



举个栗子（3台4C-8G虚机）



YAPI 平台

https://yapi.ymfe.org/index.html https://github.com/YMFE/yapi

极客邦企业培训与咨询

帮助企业与技术人成长

精品课程

Excellent Course

- ☑《互联网大规模分布式架构设计与实践》
- ☑《基于大数据的企业运营与精准营销》
- ☑《大数据和人工智能在金融领域的应用》
- ☑《区块链应用与开发技术高级培训》
- ☑《通往卓越管理的阶梯》

扫码关注官方微信服务号
了解更多课程详细信息

Geekbang
极客邦科技

THANKS