

有赞技术 Lv4

2018年05月22日 阅读 12577

[关注](#)

Node 在有赞的实践

一、概述

4月21日，有赞举办了第一届“有赞技术开发日”的活动，我作为分享讲师，分享了有赞最近一年在 Node 这一块的实践经验。但由于分享时间有限，我也只能把最重要的内容拿出来和大家分享，所以这个周末就花了几个小时时间，结合那次的分享，并完善了其中的一些内容，写了这篇文章，希望可以给大家带来新的启发。

二、Node 基础框架的迭代与演进

1. 从 Koa 到 阿童木 (Astroboy)

(1) Koa + 中间件

有赞最早的一个比较完整的 Node 项目是公司内部的一个管理系统，这个系统是用 Node 全栈开发的，主要包括一个给 HR 用的员工管理系统和给小伙伴用的 APP。就像大多数公司一样，我们第一个 Node 项目也是直接用 Koa，然后整合一些开源的中间件，这样就快速的把项目搭建起来了。

这个项目做了半年之后，我们把 Node 该踩的坑基本也都踩了一遍，所以我们就开始尝试在对外产品上使用 Node 了，我们第一个尝试改造的项目是公司的官网，这是最简单的一个项目，基本没什么大的风险。

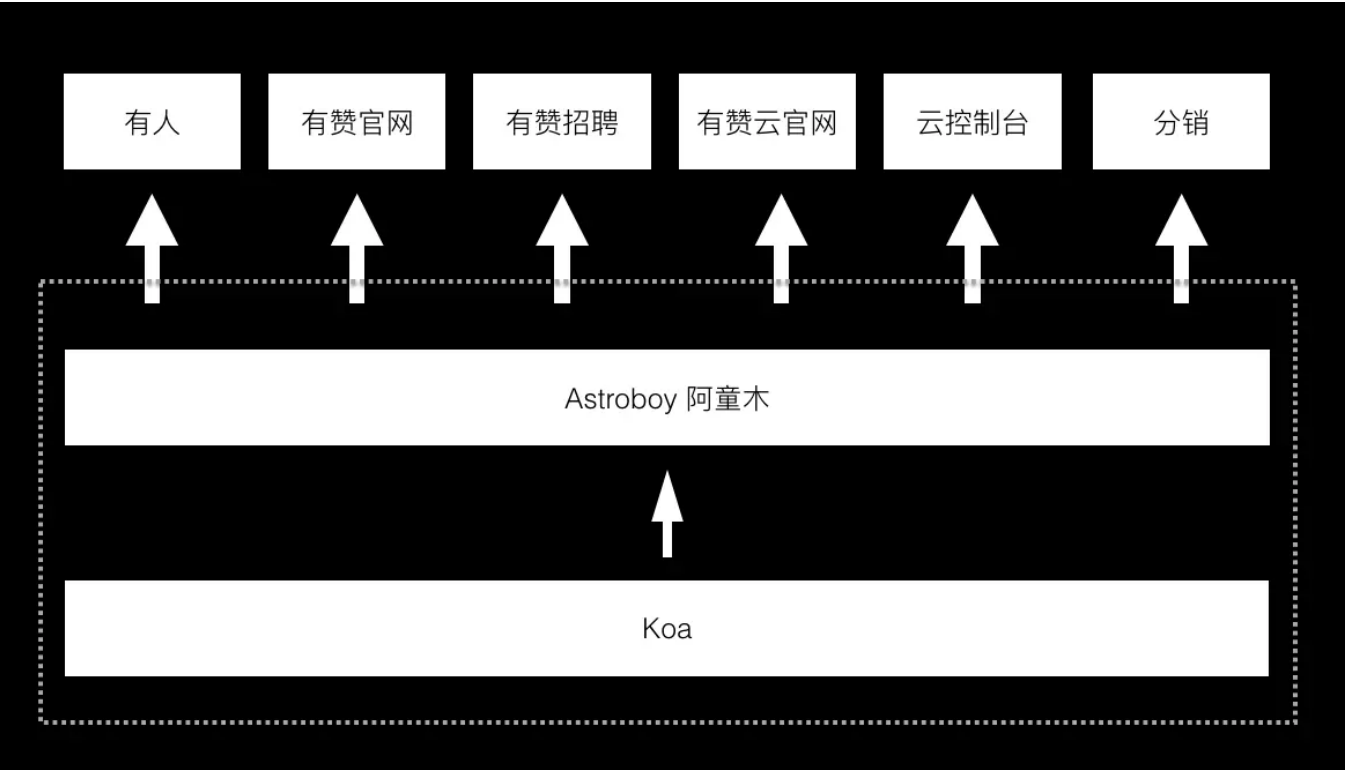
(2) 脚手架项目模板

第二个项目我们不可能再按照之前的方式，简单用 Koa 加上一堆中间件的方式来搭建项目了，因为已经有了之前的经验，所以我们就整理了下这一套方案，抽离出了一个项目模板，每个新项目只要把这个模板克隆下来，然后改一下配置，就可以快速搭建出一个新的项目来。

(3) 阿童木 1.0



项目多了之后，这种方式弊端很快就显现出来了，因为模板代码和业务代码是耦合在一起，如果要改模板生成的代码，只能每个项目手动更新，而随着时间的推移，越来越难保持同步了，每个项目的目录结构和代码风格可能也会变得非常不一样，所以，**解耦框架代码和业务代码就非常重要了**。所以我们就在脚手架模板的基础上抽离出了一个框架叫 Astroboy（阿童木），这个框架是在 Koa 的基础上封装的，这样，每个项目都基于这个框架开发，如果框架更新了，项目也只需要更改下框架的版本号。



(4) 阿童木 2.0

很多项目都开始用 Node 了，新的问题又出现了，因为每个产品的业务场景都不一样，对框架的需求也都不一样。例如某个中间件，产品 A 可能需要，而产品 B 可能根本不需要这个中间件，而这个时候的框架又不支持定制改造。所以对框架来说，又提出了新的挑战，所以在今年年初，对框架做了一次大的重构。

这次重构在阿童木 1.0 的基础上，加入了很多新特性，主要有以下几点：

- 基于 Koa2 开发，性能表现优异
- 提供基于 Astroboy 定制上层框架的能力
- 高度可扩展的插件机制
- 渐进式开发

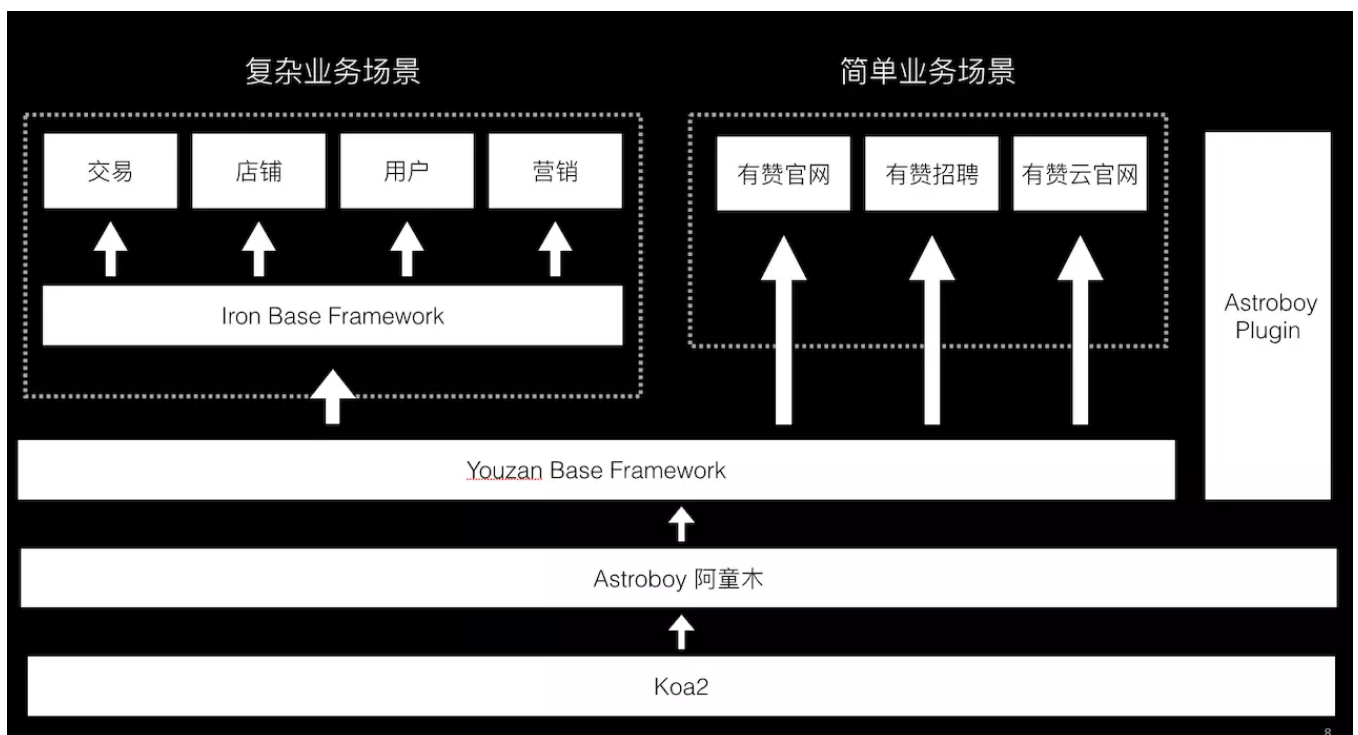
首先提供基于 Astroboy 定制上层框架的能力，如下图所示，Youzan Base Framework 是在阿童木的基础上定制的一个有赞最基础的 Node Web 框架，这一层主要集成了一些有赞最基础的服务，像：

- 天网系统接入，这是有赞内部的一个日志及业务监控系统



- 健康检查，运维监控系统每隔5秒钟，都会检查系统服务可用性
- 全链路监控，对于一次 HTTP 请求，一般都会调用多个后端接口，相应的后端接口也会再去调用其他接口，所以整个调用过程实际上是一棵树状的结构，如果碰到性能问题，找出其中性能瓶颈问题就非常重要了，全链路监控就是为了解决这个问题。
- Dubbo 服务调用接入，关于这一点，查看下面关于服务化的介绍。

有了 Youzan Base Framework 后，我们就需要在上面开发业务了，这个分两种业务场景：对于一些简单单一的业务，直接继承 Youzan Base Framework 开发就可以了；而如果是一些复杂的业务，就可以先在 Youzan Base Framework 的基础上，定制出一个业务框架，像我们有赞原先有一个超大的 PHP 项目（我们叫 Iron），那么服务化拆分后，Node 就承担了原先 PHP 的部分，所以我们新先定制了一个业务级的框架叫 Iron Base Framework，然后再按照业务模块（交易、店铺、用户、营销）拆分成多个子项目。



其次是支持插件化，关于这一点，可查看下面关于插件的说明。

2. 框架的几个核心概念

以上介绍了有赞 Node 基础框架迭代和演变的过程，下面主要介绍下阿童木2.0 框架的几个核心概念

(1) 应用 Application

应用 Application 的概念很好理解，在这里应用就可以理解成一个项目，它是从框架继承下来，并且实例化之后的一个实例，应用也是由一个一个插件构成的。



(2) 框架 Framework

Astroboy 框架是在 Koa2 的基础上封装的，关于框架的概念，这里就不再做过多的介绍了。

(3) 插件 Plugin

插件化是软件设计中一个很重要的思想，很多软件像 Eclipse 都支持这样的特性，插件化可以让我们的系统解耦，每个模块做到独立开发，而模块之间又不会相互影响，这样的特性对于大型项目来说是非常重要的。

插件化是 Astroboy 框架中最核心的一个实现，它是服务（Service）、中间件（Middleware）和工具函数库（Lib）等的载体，它本质上还是 NPM 包，只不过是在 NPM 包的基础上，做了更深层次的抽象。基于 Astroboy 的应用，就是由一个一个的 Plugin 组成的，Plugin 就是我们手中的积木，通过 Astroboy 的框架引擎把这些积木组织在一起，就形成了系统。

那么插件跟普通的 NPM 包有什么区别呢？

插件约定了目录结构，这样每个插件看起来都是类似的，这对于团队的协作是非常重要的，如果每个模块看起来都不一样，那么团队的协作成本就会很高。应用启动后，插件的代码是自动注入到整个应用的，只需要在插件的配置文件里面开启这个插件即可。

一个插件可以包含哪些信息？

- 插件元数据，包括插件名称、版本、描述等；
- 服务（Service）、中间件（Middleware）以及工具函数库（Lib）等；
- Koa 内置对象的扩展，包括 Context、Application、Request 以及 Response 等；

插件的管理

- 安装插件，通过 npm install 命令即可，例如：npm install [<@scope>/]@
- 启用插件，安装插件后还需要启用插件，插件才会真正生效。启用插件也很简单，只需要配置 plugin.default.js 即可，如果不同环境插件配置不一样，也只需修改相应* 环境的配置（plugin.\${env}.js）即可，这里 env 表示 Node 运行时的环境变量，例如：development、test、production 等。如下代码所示：

```
'astroboy-cookie': {  
  enable: true,  
  path: path.resolve(__dirname, '../plugins/astroboy-cookie')  
}
```



enable 设置成 true 就可以开启这个插件，path 表示插件的绝对路径，这种一般适合于还在快速迭代中的插件，如果插件已经很稳定了，你就可以把这个插件打包发布成一个 NPM 包，然后通过 package 声明你的插件即可，如下代码所示：

```
'astroboy-cookie': {  
  enable: true,  
  package: 'astroboy-cookie'  
}
```

- 禁用插件，禁用插件就更加简单了，只需将 enable 设置成 false 即可。

三、Node 接入有赞服务化体系的历程

1. 为什么要做服务化？

随着公司业务的发展，网站应用的规模不断扩大，垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的分布式服务框架(RPC)是关键，所以在这个时候，分布式服务架构就势在必行了。

2. 技术栈的选择

在介绍技术栈选择之前，先讲一下公司的一些技术背景。

在公司成立初期，为了能够快速开发，把产品快速做出来推出市场，所以我们选择用 PHP 语言，我想这也是大多数创业公司的选择。而随着业务的发展，PHP 越来越难处理复杂的业务。

所以等到了一定时候，我们开始做服务化拆分，那么首先考虑的就是底层技术的选择，我们从下面几点考虑：

- 第一个是这门技术的生态是否足够完善，也就是相关的开源软件、工具是否成熟；
- 第二个是否能够快速招到你需要的人才。

3. 服务化拆分之后，每一层职责分别是什么？

对于 Node 层，我们的定位是一层很薄的中间层，Node 这一层不会过多地处理业务逻辑，业务逻辑全部都交给 Java 来处理，它只负责下面三件事情：

- 模板渲染：模板渲染说的就是 HTML 模板的渲染；



- 业务编排：对于一个稍微复杂一点的页面，通常需要聚合多个接口返回的数据才能显示完整的页面，所以在这种情况下，Node 就需要聚合多个接口的返回结果，然后将合并后的数据返回给前端。
- 接口转发：Java 的服务是不会直接暴露到公网提供给前端使用的，所以在这种情况下，Node 需要承担接口转发的角色。

而对于 Java 这一层，就需要承担业务逻辑以及缓存等复杂的操作，这里就不做过多的介绍了。

4. Node 如何调用 Java 接口？

那么服务化拆分之后，首先要解决的一个问题是：Node 如何调用 Java 提供的接口。首先，我们想到的就是 HTTP 的方式，这里说明一下，我们公司采用的分布式服务化框架是阿里开源的 Dubbo 框架，而 Dubbo 框架本身是支持通过添加注解的方式生成 Restful API 的，所以在初期，我们就是采用这个现成的方案。

而随着应用数目的增加，这种方式的弊端也逐渐显现出来，主要有下面几点：

- 如果某个接口需要暴露给 Node 使用，就需要手动再去添加额外的注解。
- 每增加一个应用，运维都需要针对每个应用配置域名，不同的环境又需要配置不同的域名，所以随着应用数的增加，应用域名的管理越来越难维护。
- 相应的，node 也需要维护一份很长的域名配置文件。
- 由于 Java 是直接提供 HTTP 接口，所以性能上相对 RPC 的方式会低一点。

所以，我们就调研了下，看其他公司在使用 Dubbo 框架时，Node 是如何调用 Java 的？如下图所示：

首先，Java 应用服务启动的时候，会往服务注册中心注册服务，这里的 service 注册中心可能是 ETCD 或者 Zookeeper，然后，Node 应用在启动的时候，会先从 service 注册中心拉取服务列表，接着 Node 跟 Java 服务建立一条 TCP 长链接，除此之外，Node 还需要负责 Hession 协议解析以及负载均衡等。

不难发现，这种方式 Node 的职责就比较重，而且对 Node 开发的要求会很高。所以，我们对这种方式做了改进，如下图所示：

我们在 Node 和 Java 之间添加了一层中间代理层 Tether，Tether 是用 Go 语言写的一个本地代理，Tether 会对外暴露一个 HTTP 的服务，对 Node 来说，只需要通过 HTTP 方式调用本地的服务即可，其他服务化相关的服务发现、协议解析、负载均衡、长链建立维护都交由 Tether 来处理。这样，Node 这一层就非常轻量了，那么，最终实现出来，Node 是怎么调用 Java 服务的呢？如下代码所示：

javascript

```
const Service = require('../base/BaseService');

class GoodsService extends Service {
  /**
   * 根据商品 alias 获取商品详情
   * @param {String} alias 商品 alias
   */
  async getGoodsDetailByAlias(alias) {
    const result = this.invoke(
      'com.youzan.ic.service.GoodsService',
      'getGoodsDetailByAlias',
      [alias]
    );
    return result;
  }
}

module.exports = GoodsService;
```

对 Node 来说，调用 Java 服务它只需要关注三个点：



- 服务名：服务名是由 Java 的包名 + 类名组成，例如上面的 `com.youzan.ic.service.GoodsService`
- 方法名：Java 类对外暴露的方法，例如上面代码所示的根据商品 `alias` 查询商品详情的一个方法 `getGoodsDetailByAlias`
- 参数：参数就是传递给 Java 的参数列表

最后，总结下这种方式都有哪些优点：

- 第一个是使用简单，对前端开发非常友好，只需要通过 HTTP 方式调用本地的 Tether 服务即可；
- 第二个是多语言接入成本低，后期如果有其他语言（Python、Ruby）也需要接入整个服务化体系，也像 Node 一样，它们都只需要调用本地 Tether 暴露的 HTTP 服务即可，没有额外的开发成本了。
- 第三个是后期更方便做协议层的优化，因为这种方式 Tether 其实就是一个代理，后期如果需要做协议层性能上的优化，那只需要优化 Tether 的性能就可以了。

那么，看到这里，有人可能又会想，这里 Node 也是通过 HTTP 方式调用 Java 的，性能上是不是也存在问题呢？所以这里我们就做了一些优化，如下代码所示：

```
const Agent = require('agentkeepalive');

module.exports = new Agent({
  maxSockets: 100,
  maxFreeSockets: 10,
  timeout: 60000,
  freeSocketKeepAliveTimeout: 30000,
});
```

这里，我们引用了一个 `agentkeepalive` 包，在 HTTP 早期，每个 HTTP 请求都要求打开一个 TCP Socket 连接，并且使用一次之后就断开这个 TCP 连接，使用 `keep-alive` 可以改善这种状态，即在一次 TCP 连接中可以持续发送多份数据而不会断开连接。所以通过使用 `keep-alive` 机制，就可以减少 TCP 连接建立次数。

四、参考资料

<https://github.com/apache/incubator-dubbo> <https://github.com/QianmiOpen/dubbo2.js>

<https://github.com/QianmiOpen/dubbo-node-client> <https://github.com/p412726700/node-zookeeper-dubbo>

<https://zh.wikipedia.org/wiki/HTTP%E6%8C%81%E4%B9%85%E8%BF%9E%E6%8E%A5>



关注下面的标签，发现更多相似文章

- Node.js
- Java
- 前端
- koa

有赞技术 Lv4

技术 @ 杭州有赞科技有限公司

获得点赞 4,149 · 获得阅读 116,145

关注

安装掘金浏览器插件

打开新标签页发现好内容，掘金、GitHub、Dribbble、ProductHunt 等站点内容轻松获取。快来安装掘金浏览器插件获取高质量内容吧！

评论

输入评论...

Ve 前端工程师 @ undefin...
跟千米的dubbo2很像

4月前

  回复

海子就是我

安利一波阿里云服务器99一年

promotion.aliyun.com/ntms/act/group...

1年前



回复

小九喇嘛 **Lv1** 全干工程师 @ 北京

想问下你们用的什么数据库，以及操作数据库用的库

1年前



回复

KK1472878527810 Node 开发工程师 ...

回复 小九喇嘛 **Lv1**: 对外产品Node只做中间层，不会直接操作数据库，内部用的 sequelize

1年前

饭妖精 **Lv2** 前端开发

Hession 协议解析是不是写错了？应该是Hessian？谷歌不到Hession

1年前



回复

KK1472878527810 Node 开发工程师 ...

回复 饭妖精 **Lv2**: 这个确实写错了 👍

1年前

饭妖精 **Lv2** 前端开发

回复 饭妖精 **Lv2**: 想问下中间代理层 Tether是开源的吗？有没有地址

1年前

[加载更多](#)

胡衍生同志

链路太长了。

1年前



回复

KK1472878527810 Node 开发工程师 ...

回复 胡衍生同志: 链路其实并不长，在 Node 和 Java 中间再加一层 Tether，其实是可以带来不少好处的，就像文章中说：

第一个是使用简单，对前端开发非常友好，只需要通过 HTTP 方式调用本地的 Tether 服务即可；

第二个是多语言接入成本低，后期如果有其他语言（Python、Ruby）也需要接入整...

[展开](#)

1年前

蚂蚁兄弟



早期的http请求一次tcp链接是会断，现在基本都是keep-alive长链接，还有必要加入agentkeepalive吗？？

1年前



回复

[查看更多 >](#)

相关推荐

专栏 · ConardLi · 1天前 · 前端 / 算法

前端该如何准备数据结构和算法？

954 53

专栏 · 张亚涛 · 1天前 · 前端

前端物料中台建设

88 16

专栏 · HollisChuang · 23小时前 · Java

既然synchronized是"万能"的，为什么还需要volatile呢？

116 18

专栏 · 云中桥 · 1天前 · JavaScript / 前端

「中高级前端」从多线程到Event Loop全面梳理

337 51

专栏 · 全栈者 · 1天前 · 前端

面向前端工程师的Nodejs入门手册(一)

39 2

专栏 · _鸣啦啦啦火车笛 · 3天前 · JavaScript / 前端

深入 JavaScript 设计模式，从此有了优化代码的理论依据

520 35

专栏 · 腾讯IVWEB团队 · 3天前 · 前端

写给前端工程师的Flutter教程

471 22

专栏 · 阿里云前端 · 17小时前 · 前端


《阿里云前端技术周刊》第十八期


10



专栏 · 寻找海蓝96 · 5天前 · 面试 / 前端

20W字囊括上百个前端面试题的项目开源了

 761

 52

