微前端在网盘的落地

张亚涛

自我介绍

- 姓名
  - 张亚涛
- 工作经历
  - 入行6年，前端5年
  - 百度-百度网盘-前端技术负责人
- 社区帐号
  - 微信
  - github
  - 思否
  - 掘金
  - 公众号

张亚涛

资深打杂 | 百度

头脑灵活，基础扎实，有项目经验

目录

1. 什么是微前端
2. 场景分析
3. 灵感来源
4. 微内核架构
5. 实现思路

开始出发

项目迭代就像搭积木

某一个积木拼错了

增加一个新功能

其他人也想使用这个部件

其他人的部件怎么添加到我的这个积木里面

要搬家了，这个大家伙怎么带过去

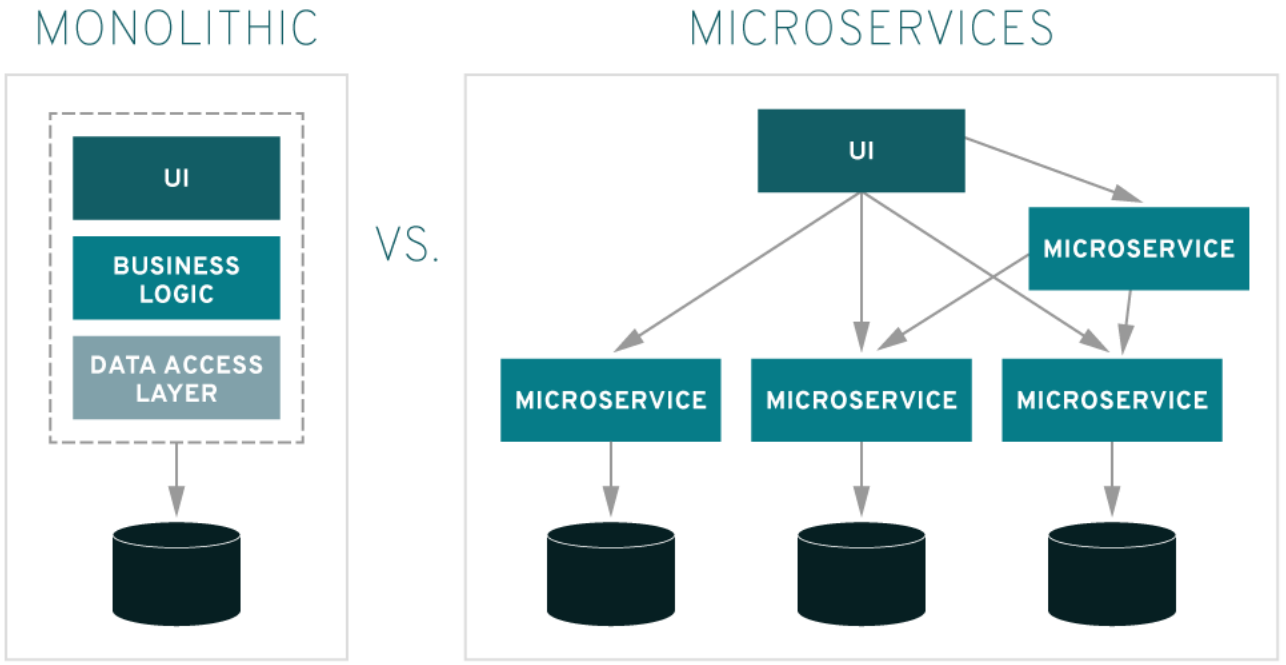
什么是『微前端』

- 背景：
  - 前端应用越来越复杂

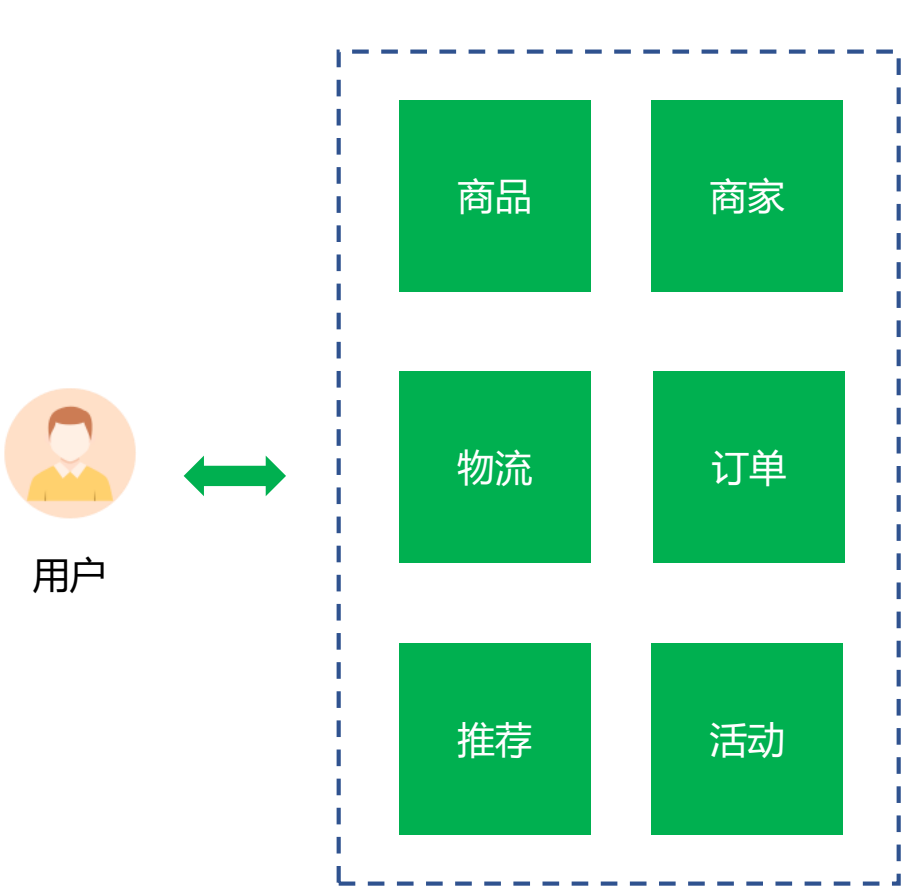
- 导致：
  1. 人力成本压力
  2. 维护成本高
  3. 迭代成本高
  4. 需求变更影响范围大
  5. 持续化投入产出比不足

- 所以：
  - 面对这种情况『其他人（后端）』是如何处理的?

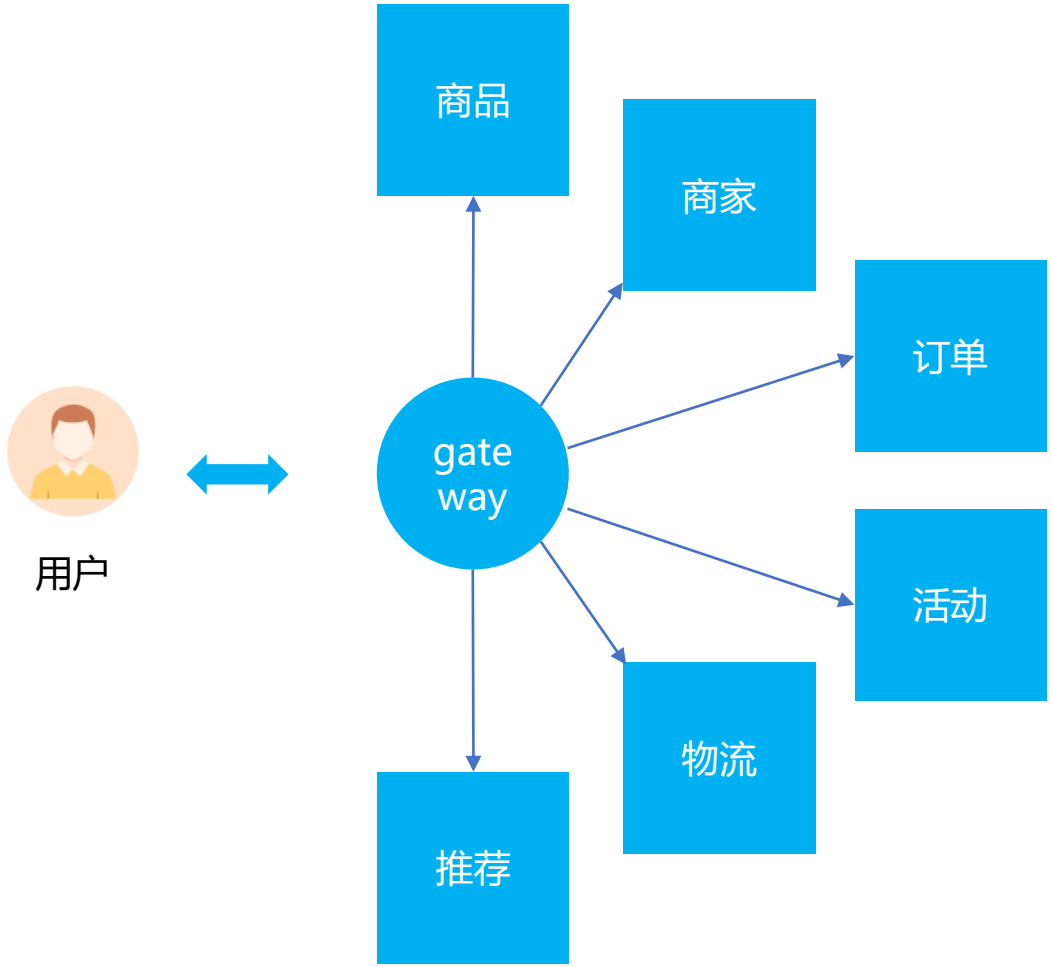
- 答案：
  - 重新洗牌（先拆后合）



什么是『微前端』



牵一发而动全身



事不关己，高高挂起

什么是『微前端』

- 优点：
  1. 隔离(服务、IDC)
  2. 弹性/扩展性（(如：扩容)
  3. 增强稳定性
  4. 降低成本（人力、上线、回归、需求)
  5. ...

- 注意事项：
  1. 测试
  2. 部署
  3. 服务拆分标准
  4. 过于 分散/密集

『微前端』场景分析

- 名词解释：
  - 微前端就是后端微服务思想在前端的映射

- 问题：
  - 微前端如何在浏览器中落地?

主要的三种机制全不相同

应该怎么做哪?

『微前端』场景分析

iframe

缺点：

- 结构冗余（嵌套处理）
- 事件通讯繁琐
- 只能处理视图相关服务
- 操作反馈复杂
- 其他更多的缺点

single-spa.js

缺点：

- 只有app级别的隔离
- 没有统一的服务规范
- 使用了system.js
- 对业务侵入性太强

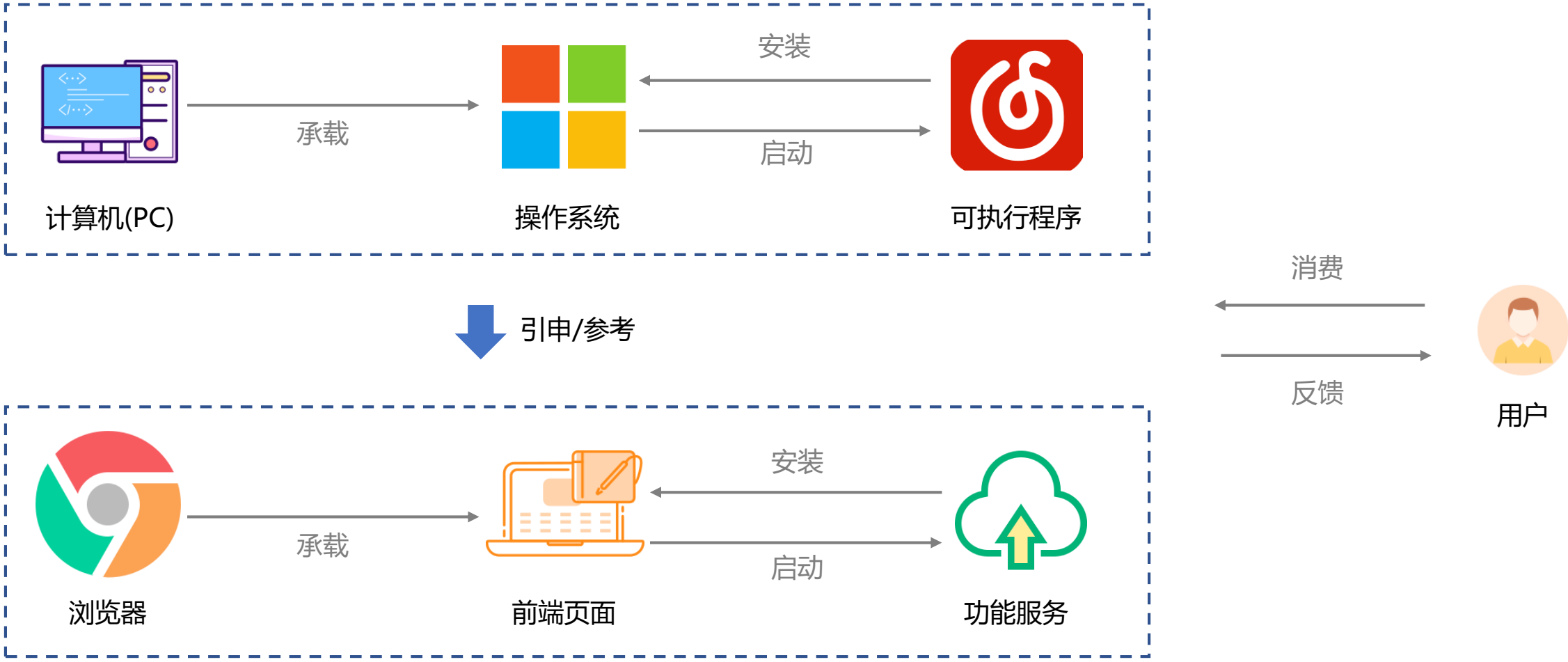
其他

目标：

吸收single-spa的优点

改正single-spa的缺点

『微前端』灵感来源

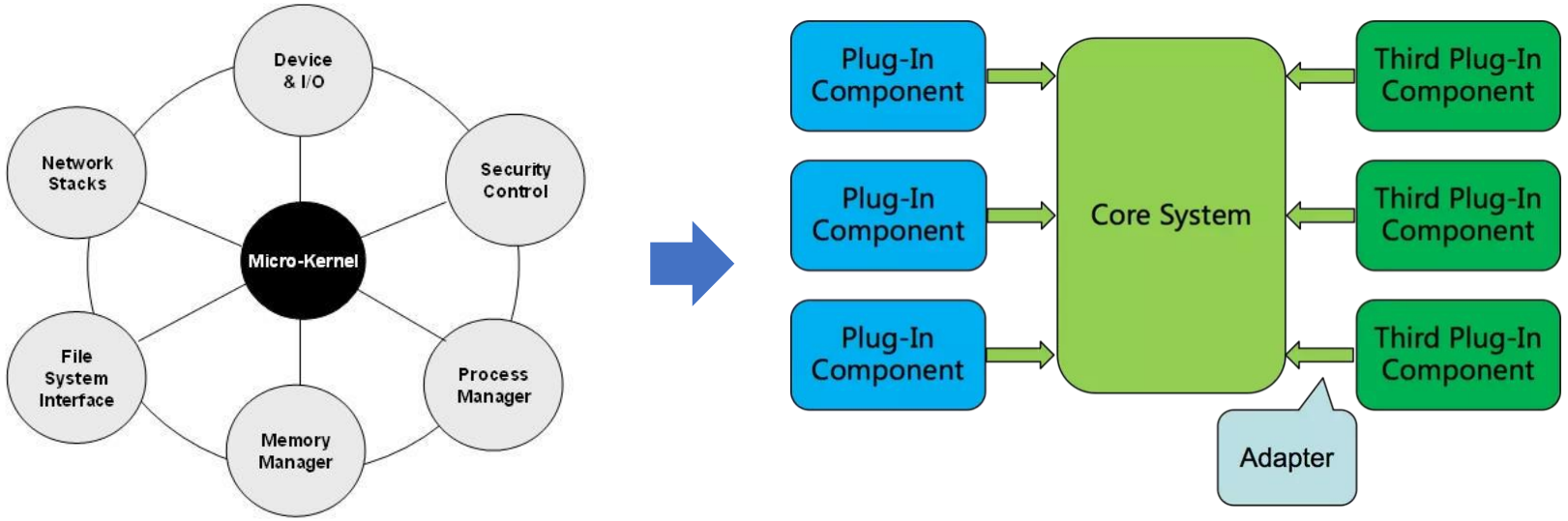


『微前端』核心——微内核架构

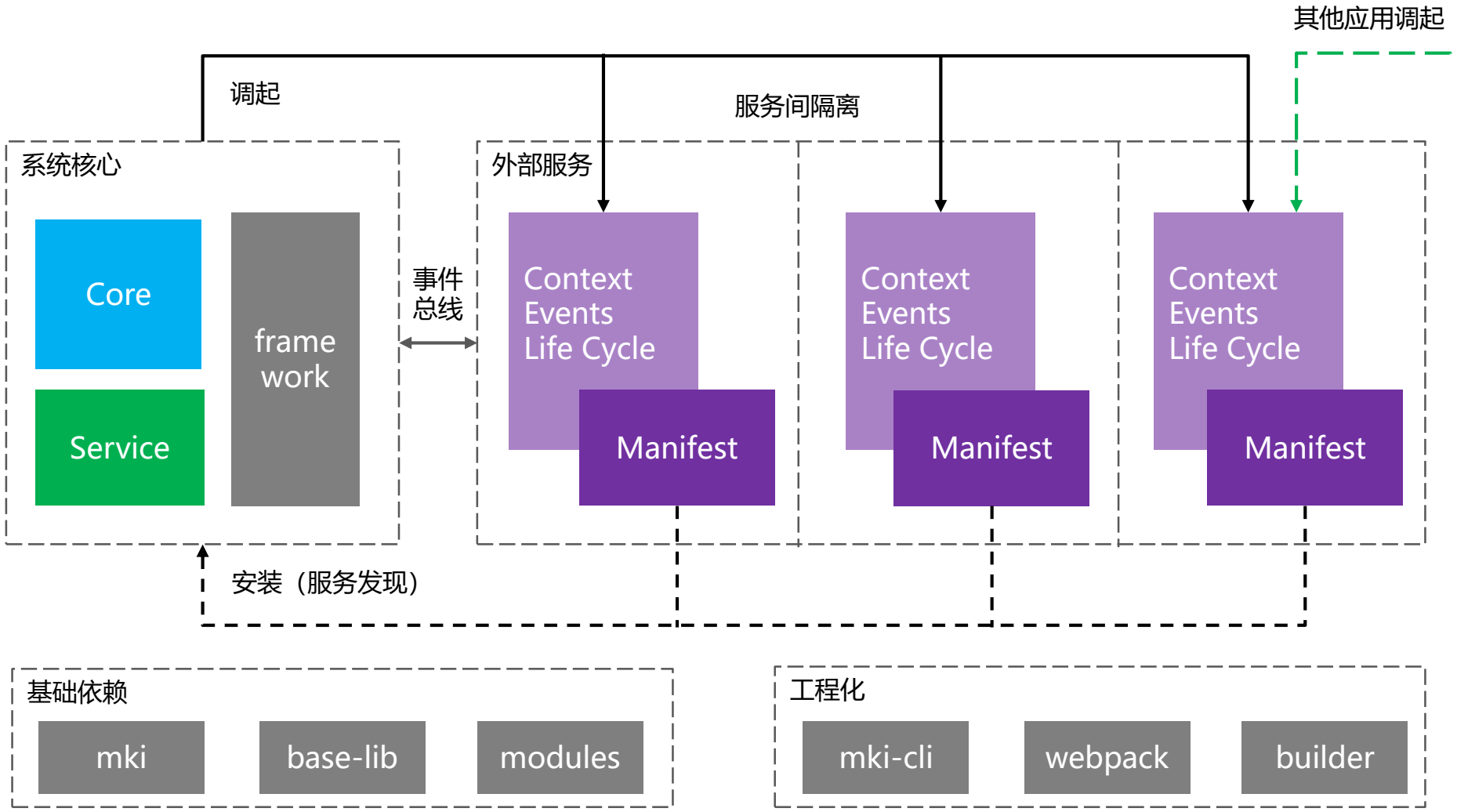
- 微内核架构
  - Windows、macOS操作系统架构

- 组成部分
  1. 系统核心
  2. 系统服务
  3. 插件系统（可执行程序）

- 为什么选择它?
  1. 服务间高度解耦
  2. 统一的插件标准
  3. 轻量级的事件机制
  4. 单一容器应用的最佳选择
  5. 对业务更低的侵入性
  6. 渐进式开发



微内核应用——前端系统


其他应用调起
调起
服务间隔离
系统核心
Core
Service
frame work
事件总线
外部服务
Context Events Life Cycle
Manifest
Context Events Life Cycle
Manifest
Context Events Life Cycle
Manifest
安装（服务发现）
基础依赖
mki
base-lib
modules
工程化
mki-cli
webpack
builder

百度网盘

网盘 动态 分享 找资源 更多

zhang384579631 | 客户端下载 会员中心

SVIP专享，精品好书免费领

全部文件

图片

文档

视频

种子

音乐

其它

我的分享

回收站

347.4G/5128G 扩容

时光轴 智能分类 最近上传

通过人物，事物，地点搜索你的图片

地点

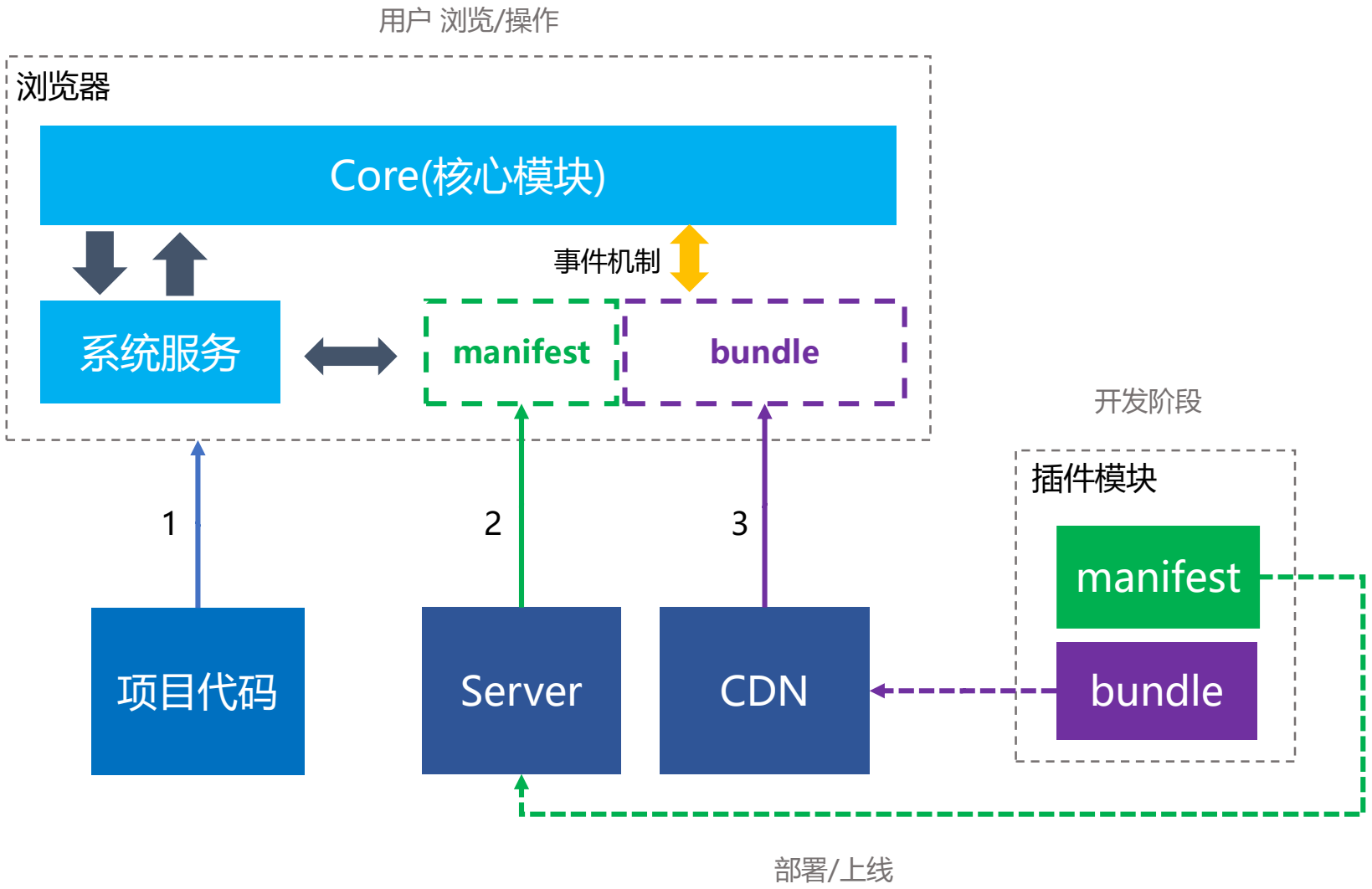
日本

事物

我的分享 2018-02-26 17:12

主要功能点及关系

- 核心模块
- 系统服务
- 插件模块
- 注册和调起
- 入口管理
- 测试和部署



功能服务(插件)

调起插件

```
ctx.call('helloworld@com.baidu.pan', {file:[{},{}]});
```

目录结构

```
./
└── helloworld
    ├── assets
    │   ├── imgs
    │   │   └── home.png
    │   └── style.css
    ├── index.js
    └── manifest.json
```

插件逻辑

```
'use strict';

// 将要被加载
module.exports.beforemounted = () => {};
// 加载完成
module.exports.mounted = () => {};
// 卸载
module.exports.unmounted = () => {};
// 入口
module.exports.start = (ctx) => {
    // 全局事件，关注订阅模式
    ctx.message.emit('eventName', {});
    // 点对点事件模式
    ctx.message.p2p.emit('target', {});
    // 调起其他服务
    ctx.call('other@com.baidu.pan', {});
    // 系统服务
    ctx.$service.http.get('/index', {params: {name: 'test'}}).then(data => {
        console.log(data);
    });
};
```

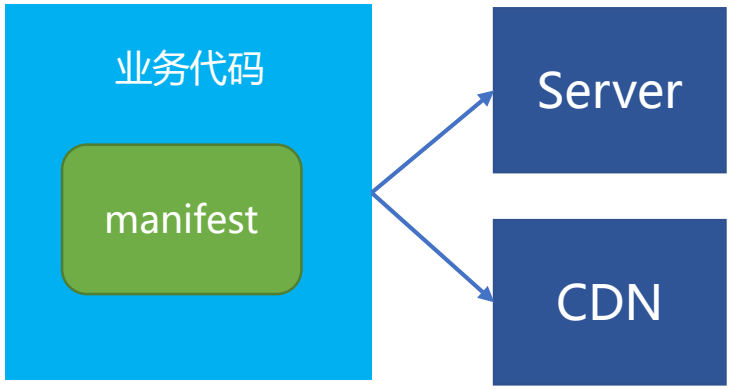
Manifest签名

```
{
  "name": "helloworld",
  "group": "com.baidu.pan",
  "preload": true,
  "arguments": {
    "file": {
      "type": "Object",
      "required": false
    }
  },
  "dependencies": [
    "other"
  ],
  "menu": {
    "area": "middle",
    "text": "你好",
    "index": 1,
    "keycode": ""
  },
  "button": {
    "area": "toolList",
    "index": 1,
    "text": "你好"
  },
  "entrance": "index.js"
}
```

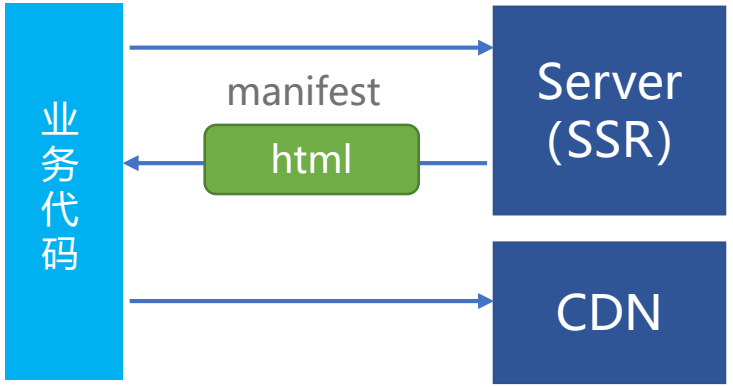
服务发现

- 三种方式
  - 硬编码
  - 渲染直出
  - 接口获取

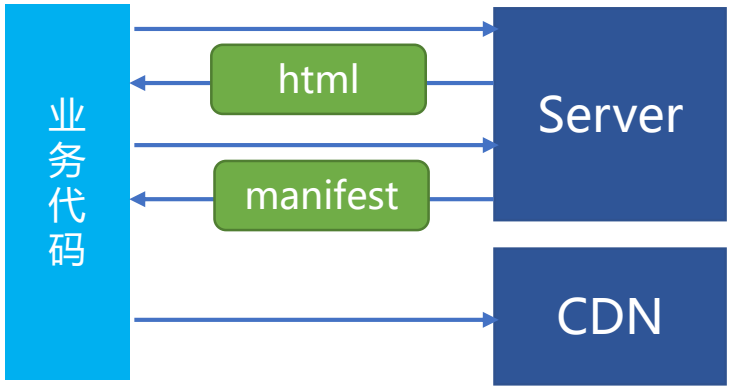
硬编码



渲染直出

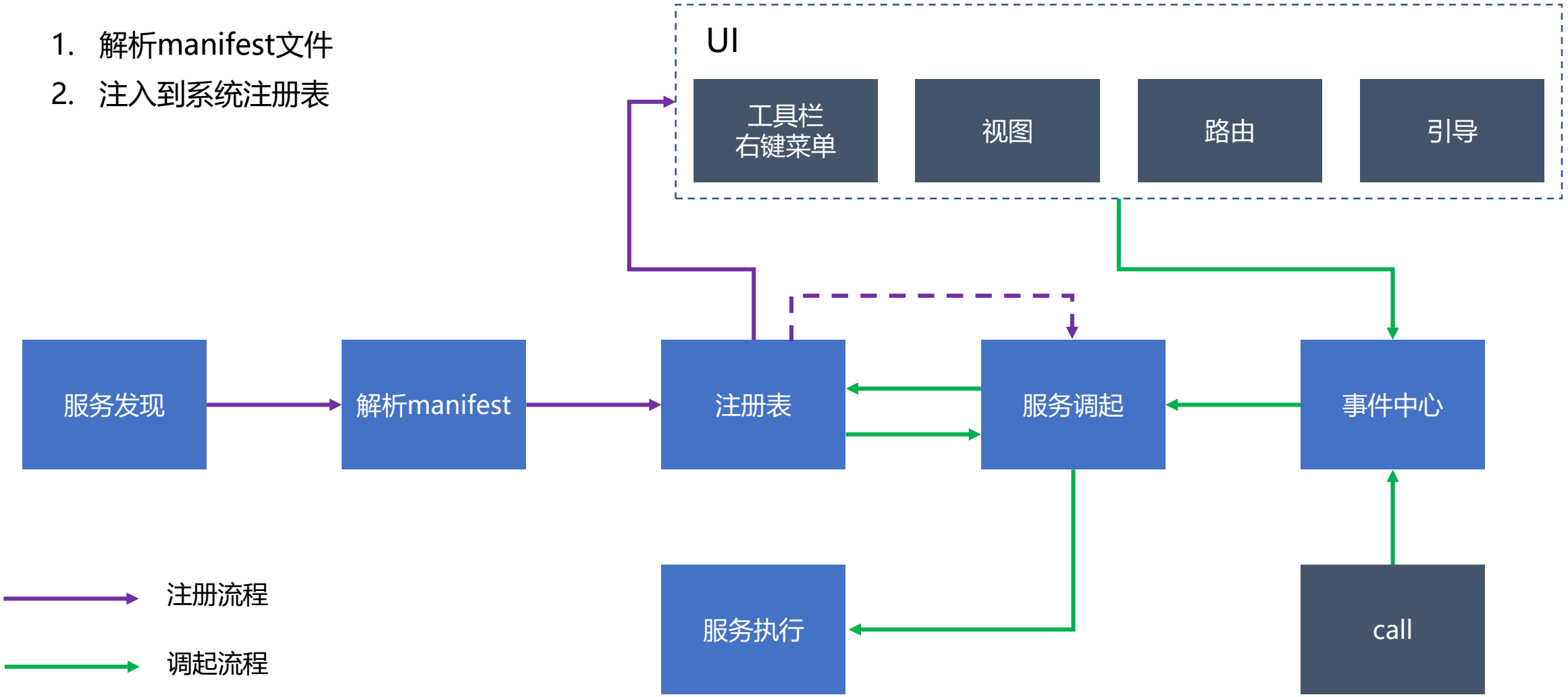


接口获取



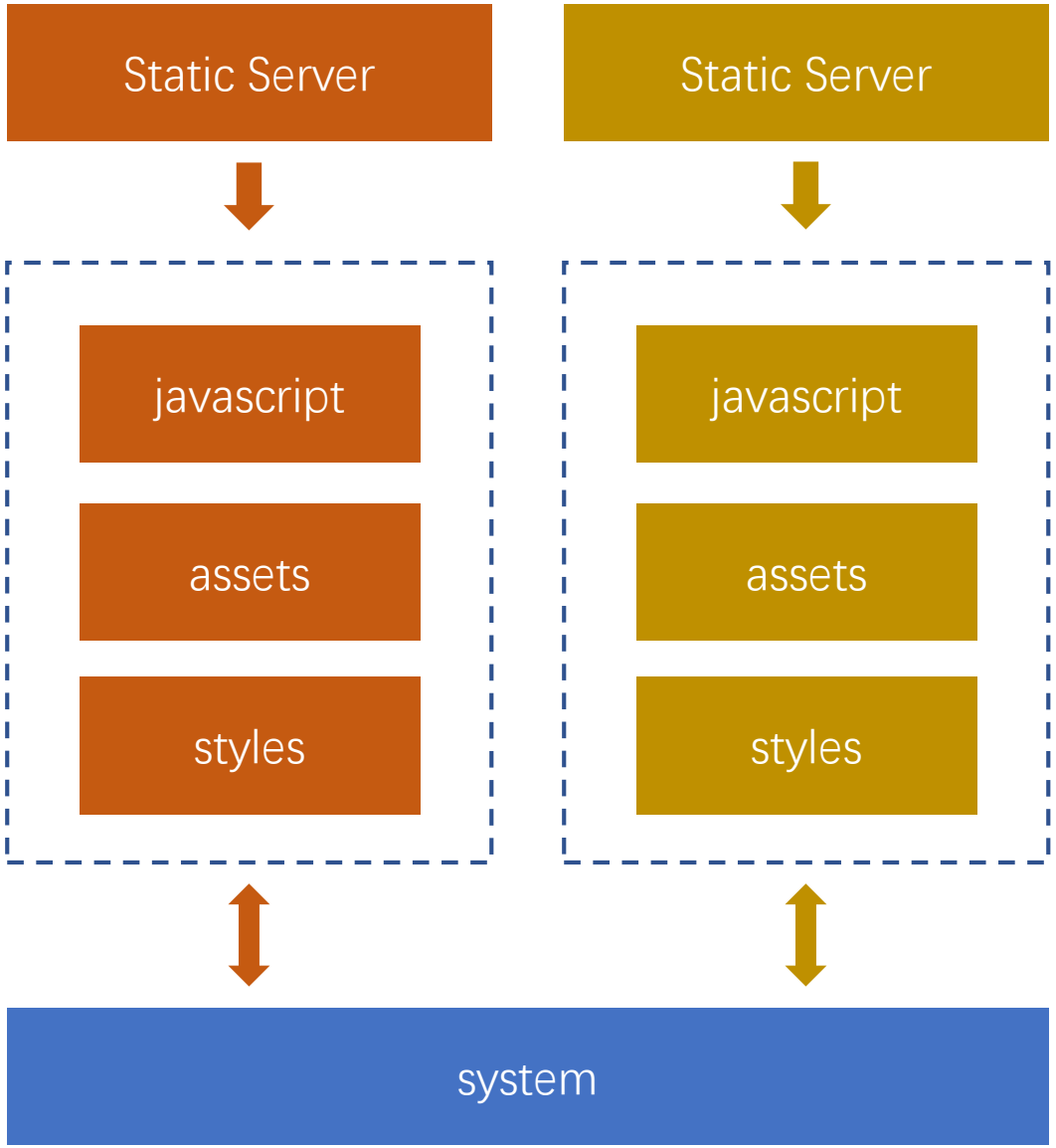
服务注册

1. 解析manifest文件
2. 注入到系统注册表



服务隔离

- 样式
  - CSS样式前缀（BEM）
  - name-block__element--modifier
- 业务逻辑
  - 只能通过ctx和外部交互
  - 在编译时通过eslint进行过滤
  - 服务可自行编译产出bundle
- IDC/CDN
  - 可部署在不同的CDN或服务器上



收益

1. 系统更加稳定
2. 迭代效率更快
3. 职责更加清晰
4. 功能随时上下线
5. 业务迁移能力大幅度提升
6. 同时支持vue、san、jquery等不同技术栈
7. 完全解决了single-spa的缺点
8. …

最后

- 说明：
  - 仅作简单的思路分享，所以没有文中并没有列出比较复杂的部分，真正落地实现你就会发现并不简单。

- 感谢：
  - 在场的各位