51CTO WOT

World Of Tech 2024

WOT全球技术创新大会

2024

智启新纪
慧创万物

51CTO WOT

Unio 新一代 WebApp 架构：RSC/SSR 在支付宝的实践

刘奎 & 肖胜桃
蚂蚁集团 前端工程师

自我介绍

刘奎

GitHub 账号 Kuitos，开源微前端解决方案 qiankun 作者。
目前在蚂蚁主要负责微前端及 Web 应用架构的建设及探索。

肖胜桃

GitHub 账号 PeachScript，UI 组件库开发框架 dumi 作者，Umi 及 Ant Design 维护者。
目前在蚂蚁主要负责前端运行时性能解决方案及前端开发框架的建设。

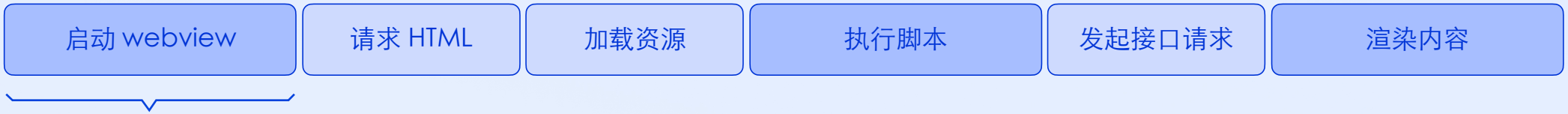
Unio 新一代 WebApp 架构：RSC/SSR 在支付宝的实践

快

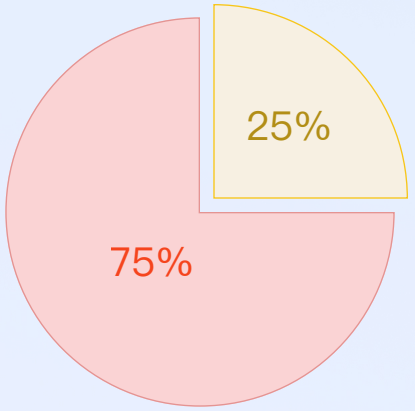
时间都花在哪里了

旗舰机

非旗舰机



部分机型耗时
需要近 1000ms



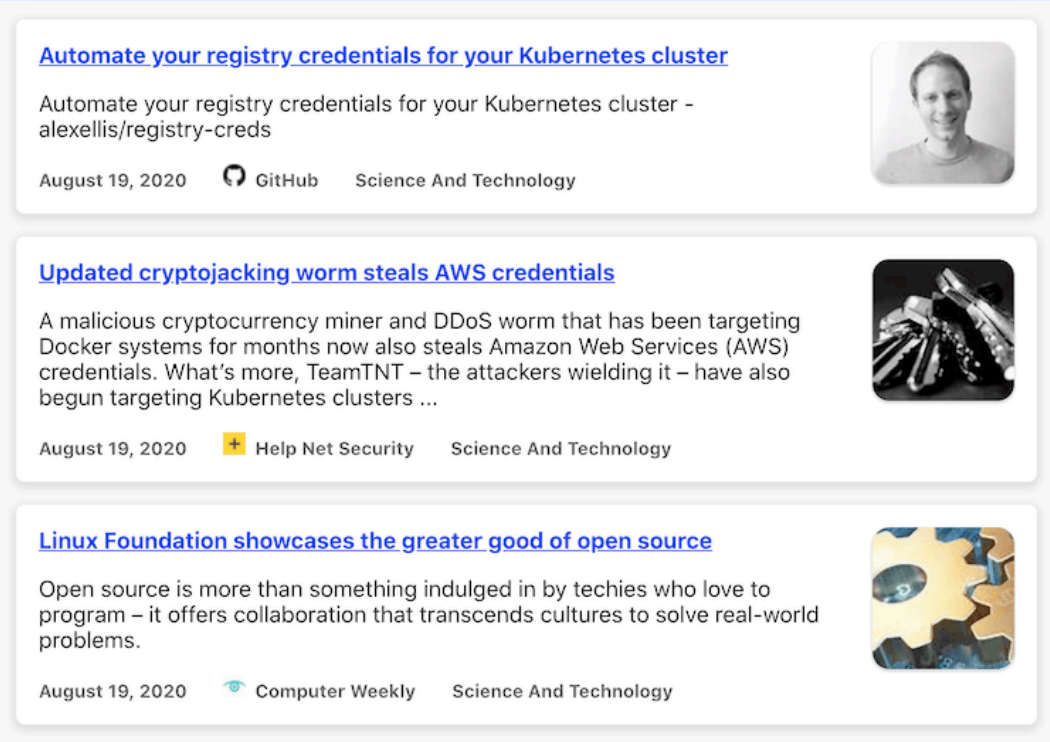
市场中的移动设备分布

旗舰机　非旗舰机

RSC/SSR 能给性能带来什么

React Server Component 是怎么工作的

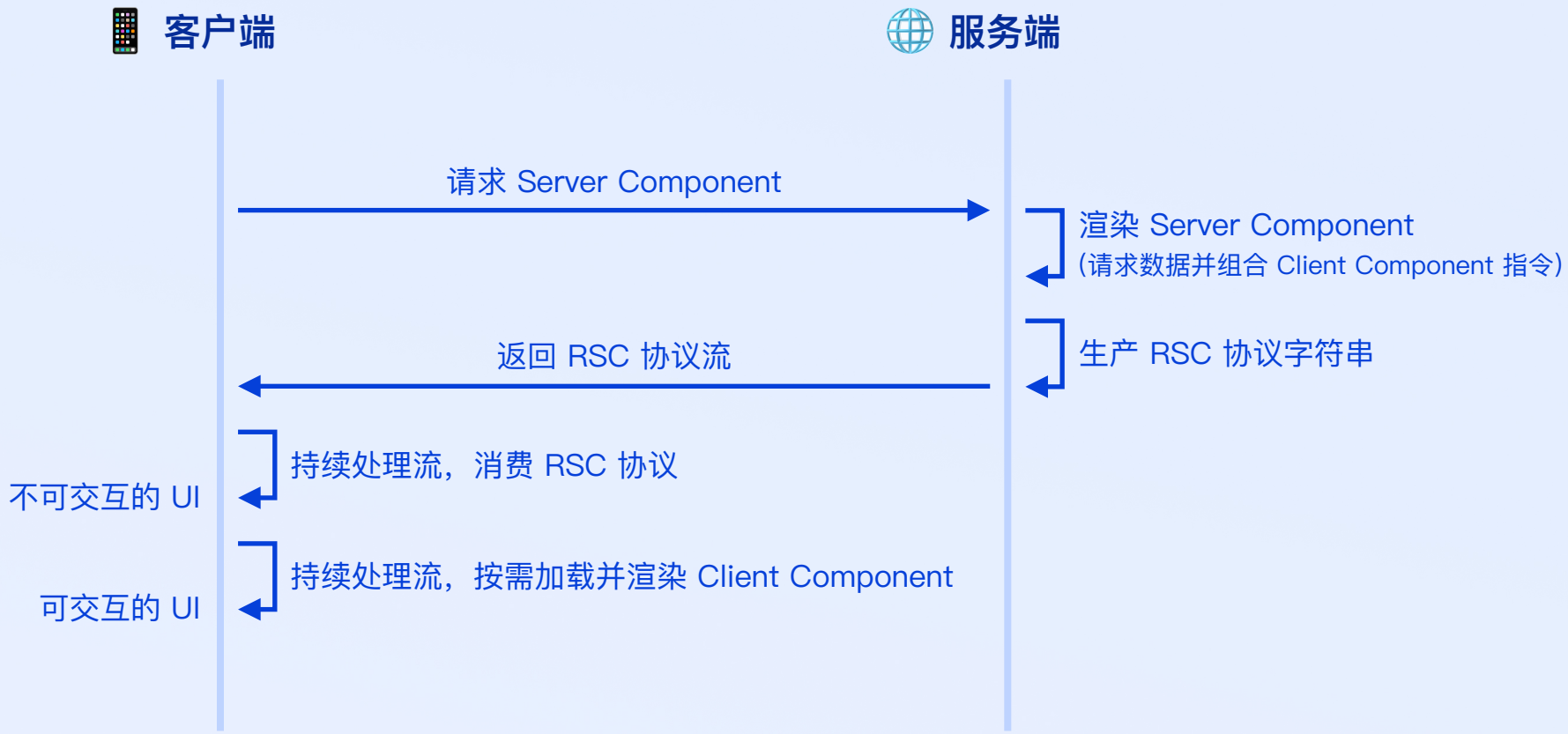
下载资源：组件 + service + moment.js + swr + ...
获取数据：[{ title: ... }, 冗余字段...]
渲染内容：




```html
<ul>
  <li>
    <a href="/posts/1">
      <h2>Automate your registry...</h2>
      <p>Automate your registry...</p>
      <div>
        <time datetime="2020-08-18T16:00:00.000Z">Auguest 19, 2020</time>
        <span>GitHub</span>
        <span>Science And Technology</span>
      </div>
      <img src="..." alt="..." />
    </a>
  </li>
  ...
</ul>
```

图片来源：https://rapidapi.com/blog/news-app-react

React Server Component 是怎么工作的

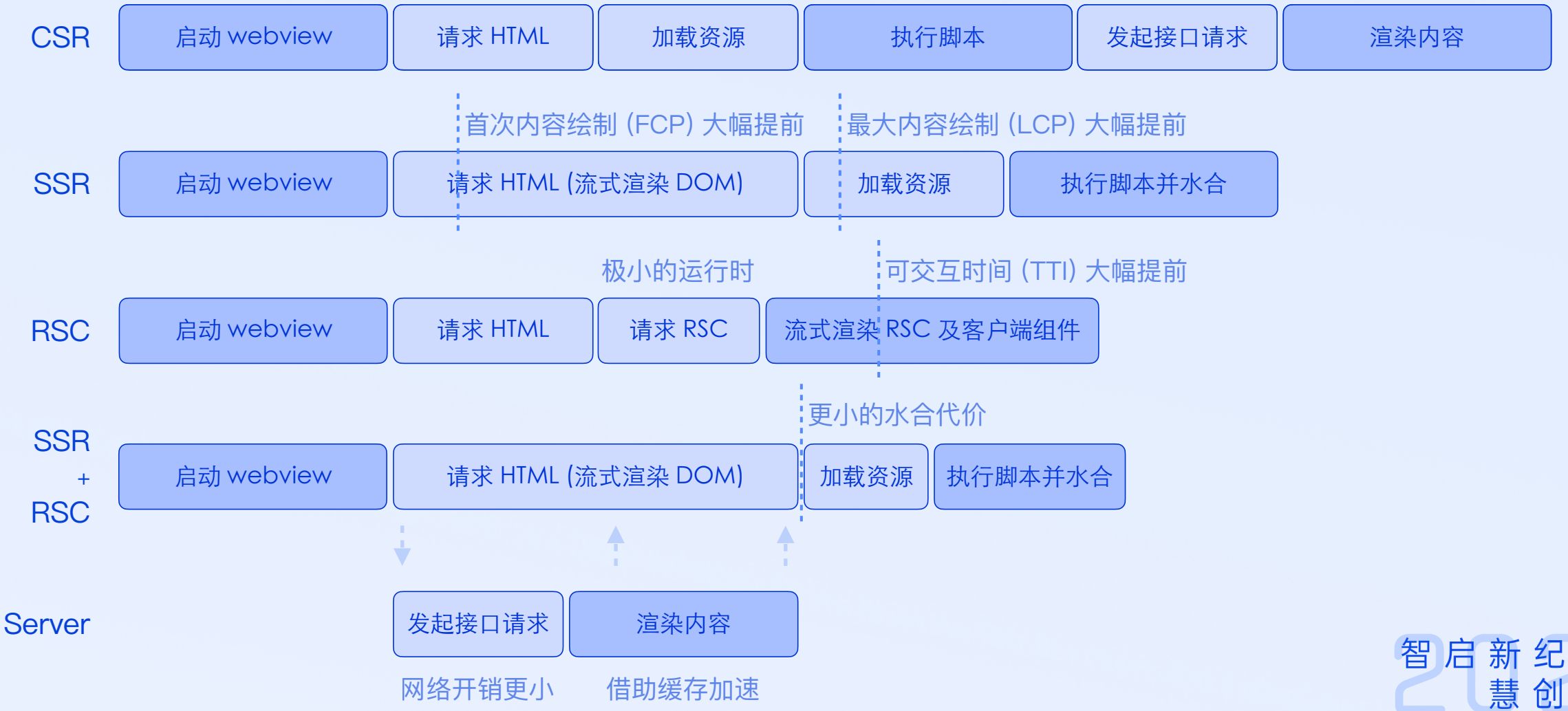


React Server Component 是怎么工作的

详细了解 RSC，可阅读：https://www.builder.io/blog/why-react-server-components

RSC/SSR 能给性能带来什么

RSC/SSR 能给性能带来什么



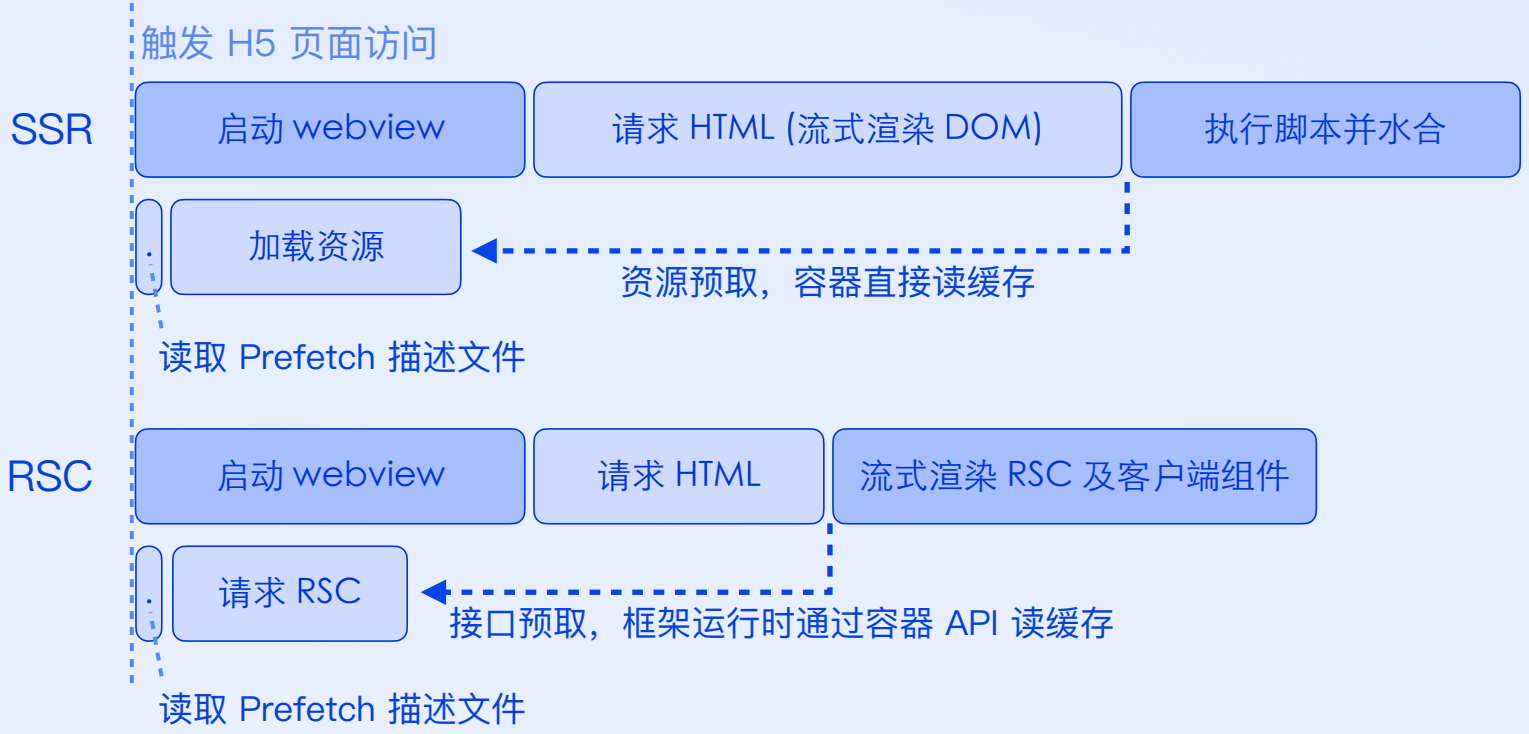
支付宝还做了些啥

支付宝的 RSC/SSR 实践

Prefetch　　自动 fallback　　构建优化　　...

支付宝的 RSC/SSR 实践

利用 Prefetch 实现抢跑



支付宝的 RSC/SSR 实践

利用 Prefetch 实现抢跑


```
{
  "offlineResources": {
    "sync": [
      // 同步加载的静态资源列表（高优）
      "https://example.com/xxx.js"
    ],
    "async": [
      // 异步加载的静态资源列表（低优）
      "https://example.com/yyy.js"
    ]
  },
  "dataPrefetch": [
    // 接口数据预取配置
    {
      "config": { "fetchType": "rpc" },
      "params": { ... }
    }
  ]
}
```

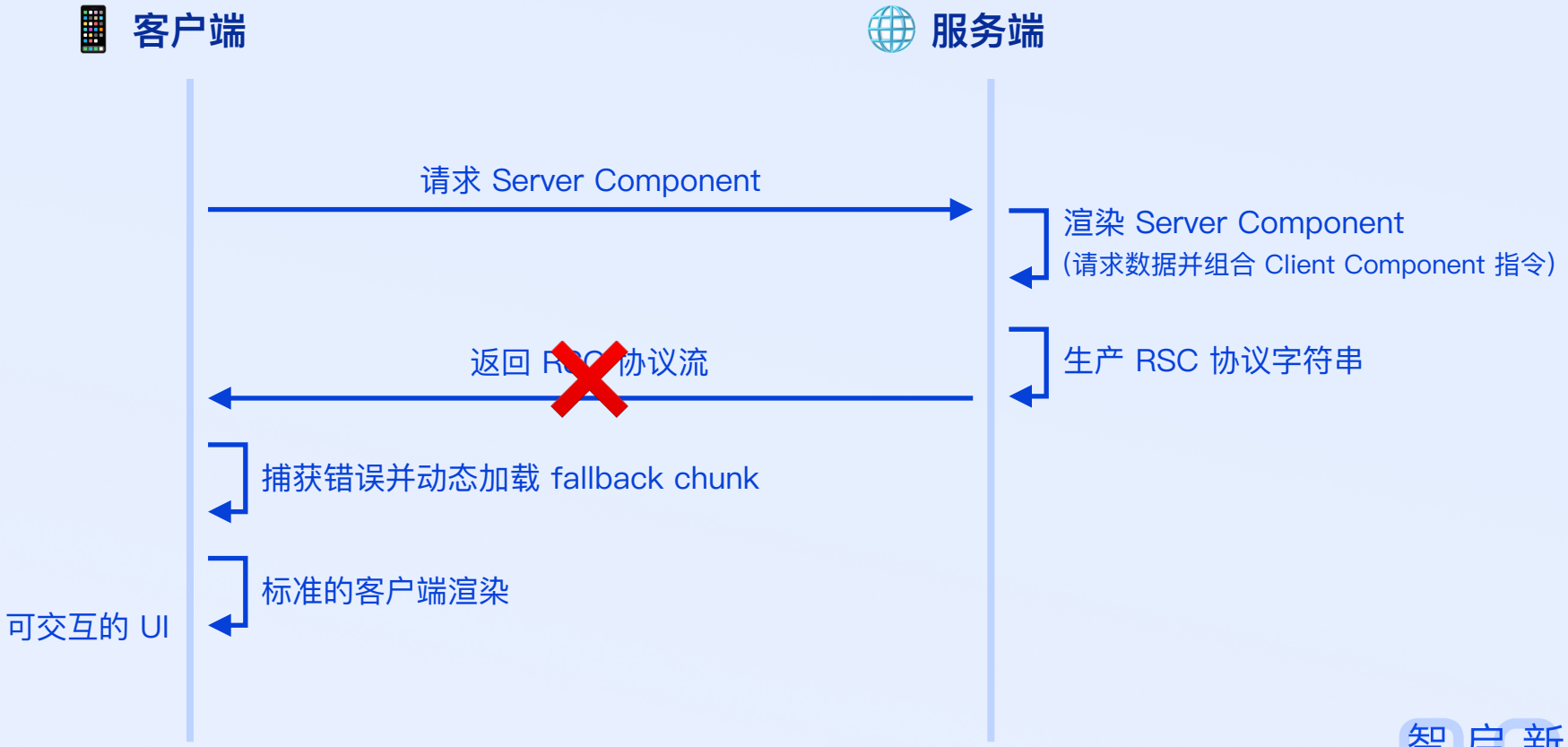
支付宝的 RSC/SSR 实践

利用 Prefetch 实现抢跑

最终效果：平均节约 200 ~ 500ms

支付宝的 RSC/SSR 实践

RSC 如何自动 fallback



支付宝的 RSC/SSR 实践

RSC 如何自动 fallback

```
import React from 'react';
import { ErrorBoundary } from 'react-error-boundary';

const RSCApp = () => { /* createFromReadableStream... */ };
const FallbackApp = React.lazy(async () => (await import('./app')).default);

export default function App() {
  return (
    <ErrorBoundary fallback={<FallbackApp />}>
      <RSCApp />
    </ErrorBoundary>
  );
};
```

支付宝的 RSC/SSR 实践

RSC 如何自动 fallback

RSC 需要满足客户端/服务端同构

不能操作数据库还是真正的 RSC 吗?

不重要，我们的目的是快，能作为 BFF 已经足够了

支付宝的 RSC/SSR 实践

RSC/SSR 的构建优化

CSR — Client 构建

SSR — Client 构建 + Server 构建

RSC — Client 构建 + Server 构建

fallback chunk

Webpack 很难顶得住了

Mako

蚂蚁自研的高性能 Rust 构建器

了解更多：https://makojs.dev

真实 RSC 业务的冷启动耗时变化

(含构建之外的框架耗时)

510s → 34s

可以顺利在业务中落地了吗
RSC 还不行

RSC 落地的最大阻碍

Client/Server 的边界难以把控

存量项目很难改造

NPM 生态尚未就绪

维护、迭代成本更高

研发习惯及思维惯性

RSC 落地的最大阻碍

```tsx
import React from 'react';

export default function App() {
  return (
    <>
      <Tabs
        items={[
          {
            title: '热门',
            content: (
              <>
                <h1>商品名称</h1>
                <button
                  onClick={() => addToCart()}
                >
                  加购物车
                </button>
              </>
            ),
          },
          ...
        ]}
      />
    </>
  );
};
```

```tsx
'use server';

export default function App() { /*...*/ }
```

```tsx
'use client';

export default function Tabs() { /*...*/ }
```

```tsx
'use server';

export default function Item() { /*...*/ }
```

```tsx
'use client';

export default function ItemActionBar() { /*...*/ }
```

RSC 落地的最大阻碍

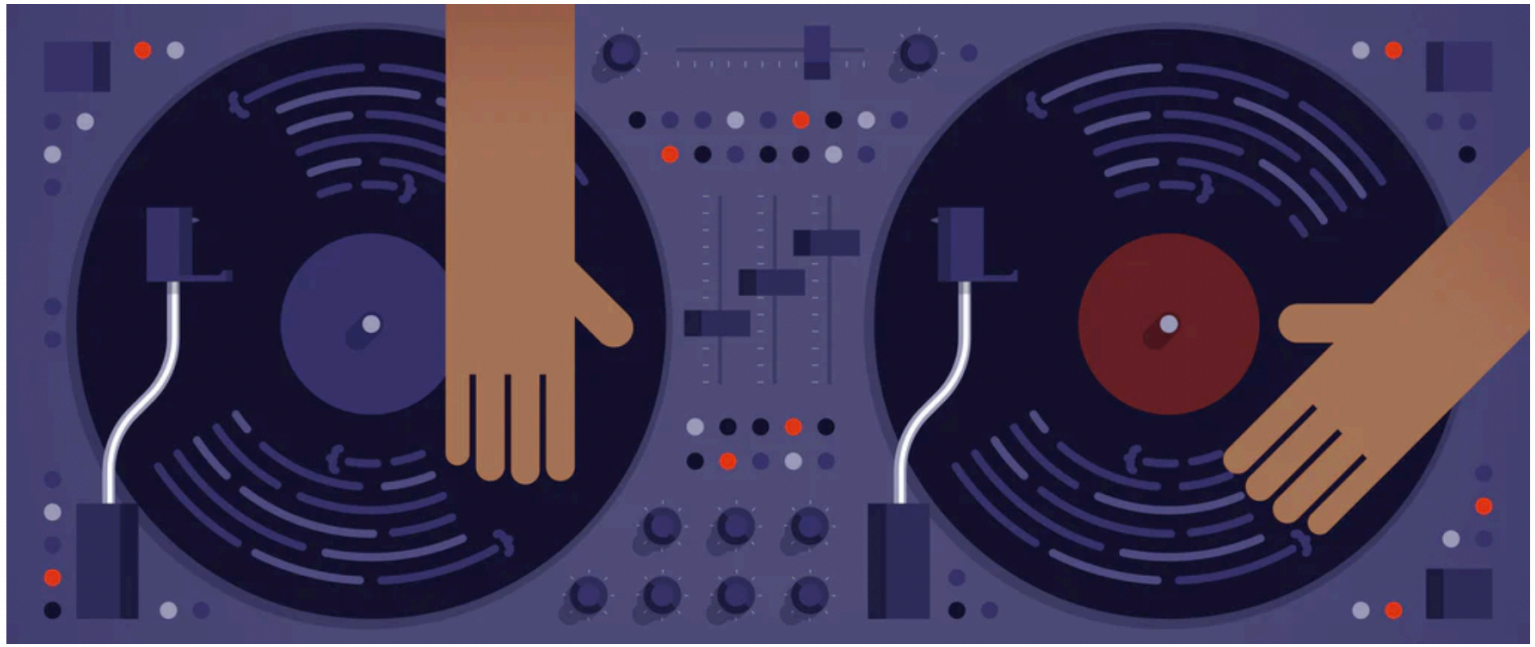
Client/S

NP

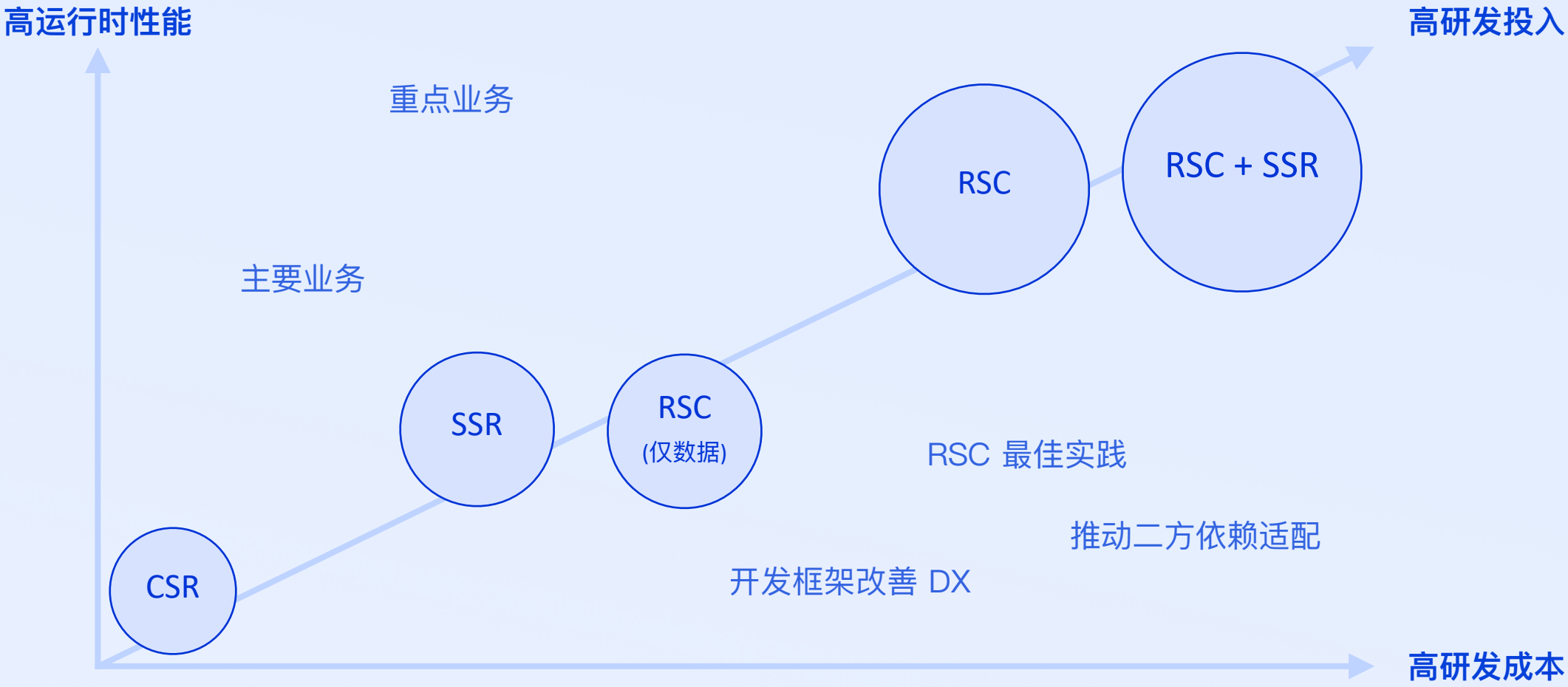
造

Mixing It Up: Remix Joins Shopify to Push the Web Forward

by Dion Almaer • Development
Oct 31, 2022 • 6 minute read



渐进式的选型方案


高运行时性能
高研发投入
重点业务
主要业务
RSC
RSC + SSR
SSR
RSC
(仅数据)
RSC 最佳实践
推动二方依赖适配
开发框架改善 DX
CSR
高研发成本

小结

❶ RSC/SSR 结合容器层的优化可以为业务应用的 FCP 和 TTI 带来明显提升

❷ 支付宝结合容器和框架实践 Prefetch、自动 fallback、本地构建优化等方案

❸ RSC/SSR 都不是『免费』的性能提升方案，存量应用要付出的成本会更高

❹ RSC 的接入门槛较高，靠渐进式方案匹配不同的业务是比强推更合适的路线

618 大促会场实录

CSR RSC

Nova 2 Plus (2017)

CSR RSC

Mate 60 (2023)

看上去旗舰机不太需要 RSC/SSR，如何实现规则化渲染?

RSC/SSR 应用部署后要依赖哪些云上的基础设施?

RSC/SSR 应用在云上的调用链路是怎样的?

51CTO WOT

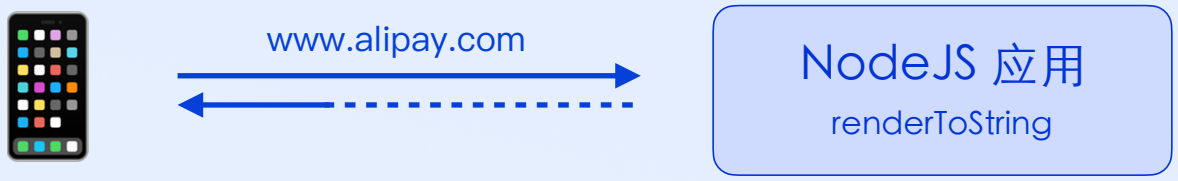
有请我的搭档

刘奎

智启新纪
慧创万物
2024

支付宝服务端渲染
云上链路的演进

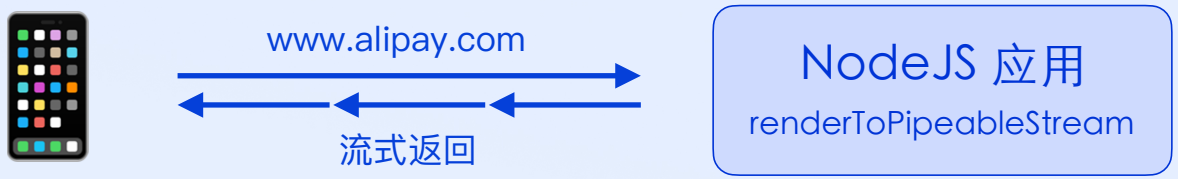
传统 SSR



阻塞式 SSR，延时过高

传统 SSR

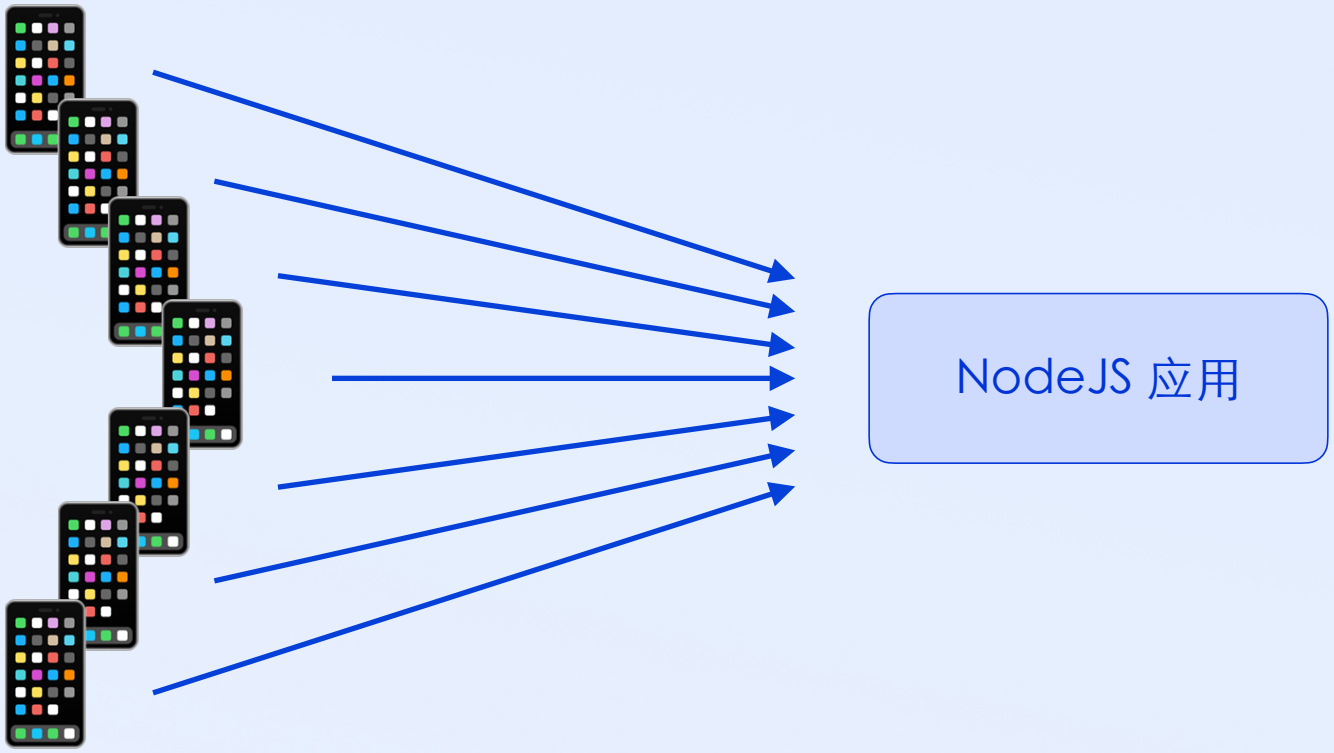
流式渲染



流式非阻塞，首字节响应（TTFB）性能提升

传统 SSR

应用型架构的容量问题



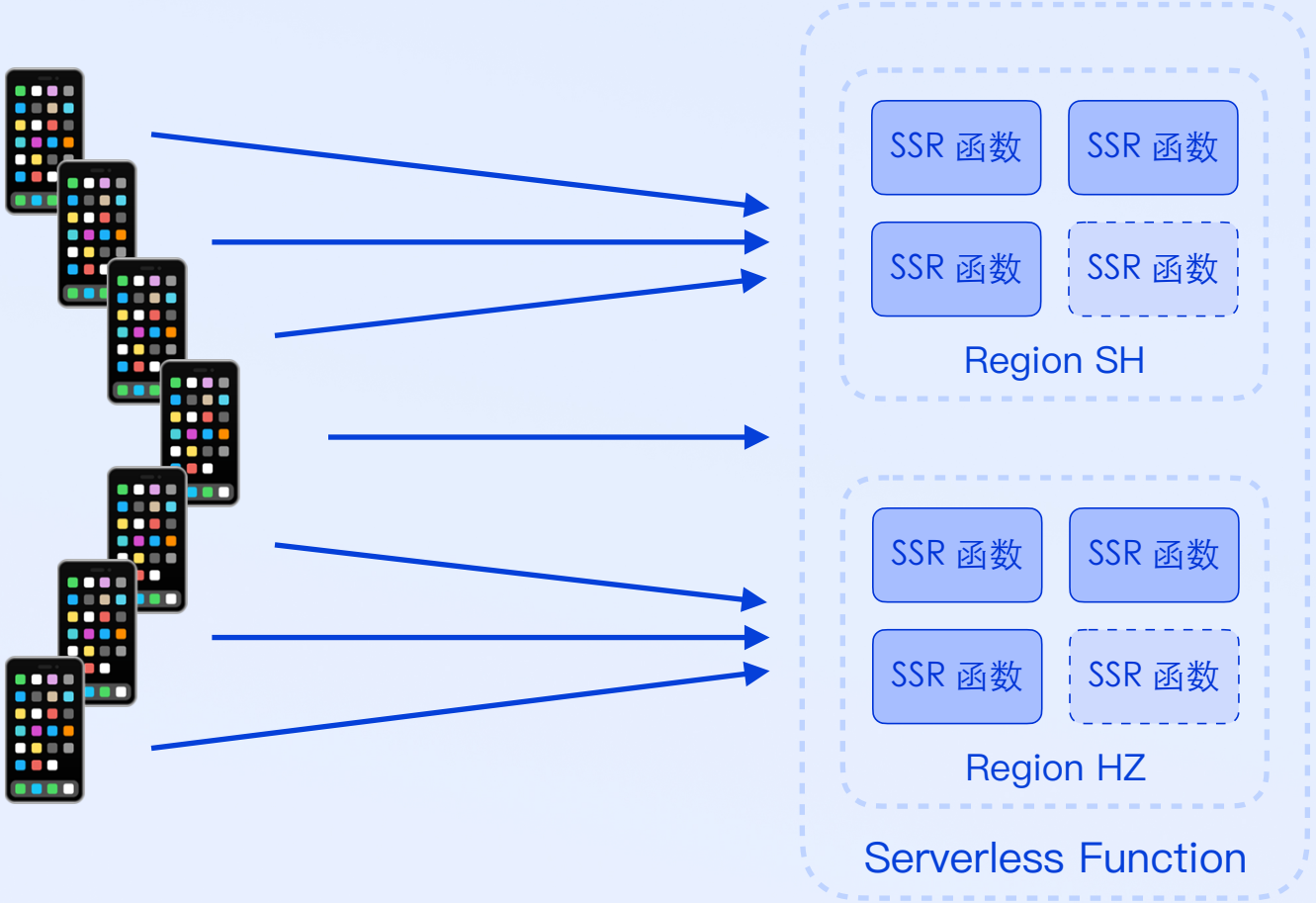
- 日常资源闲置，大促时又不够用

引入函数架构解决容量问题

按需分配计算资源

引入函数架构解决容量问题

按需分配计算资源

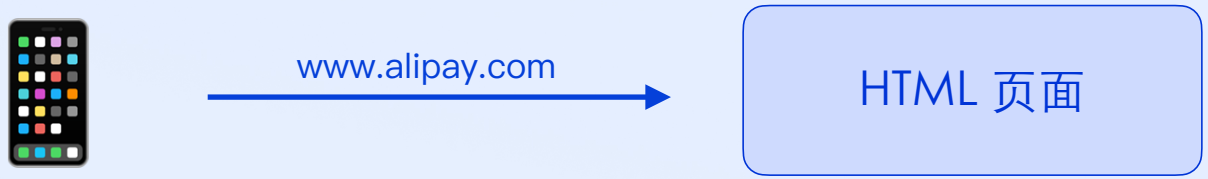


客户端/函数 架构

Client/Serverless Function

传统 C/S 架构

相比 CSR，SSR 的预期效果没有达成



传统 C/S 架构

相比 CSR，SSR 的预期效果没有达成

在 SSR 的链路中引入 CDN?

传统 CDN 的特点与局限

特点：**全球化**内容分发网络

局限：纯静态，**不具备编程能力**

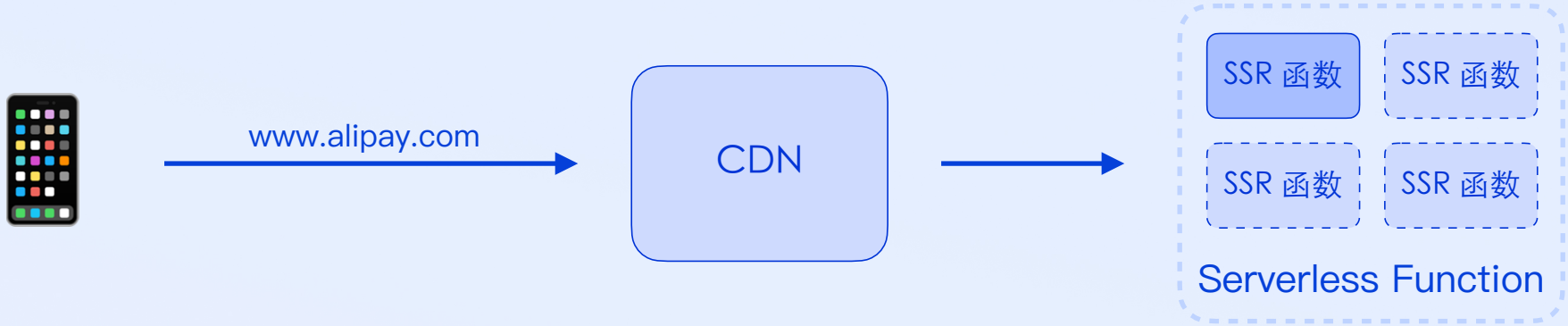
现代 CDN 边缘网络

加入边缘程序

特点：跟随 CDN 节点一起部署

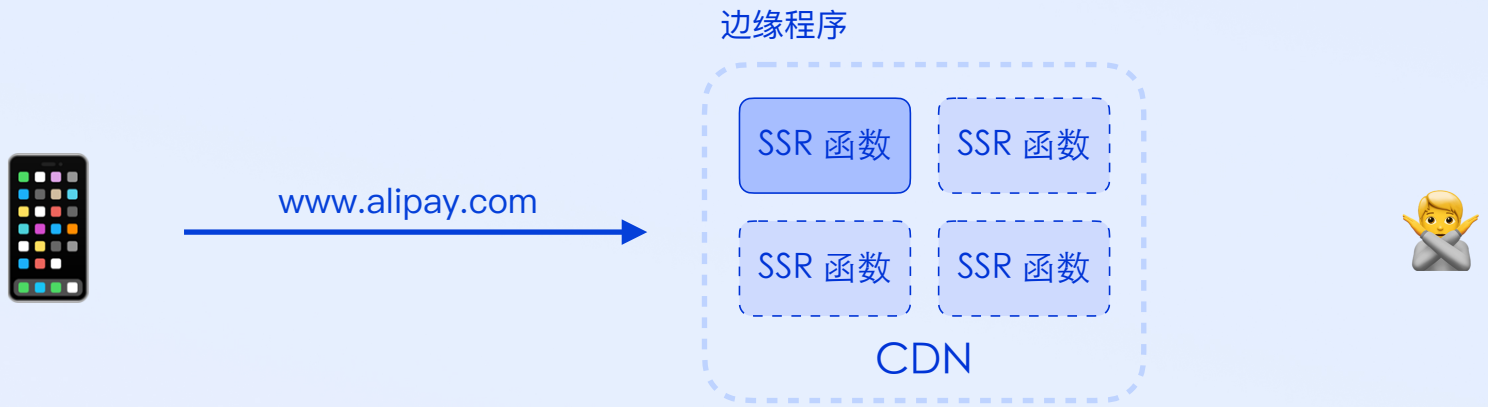
特点：使 CDN 具备可编程能力

如何在 SSR 链路引入 CDN?



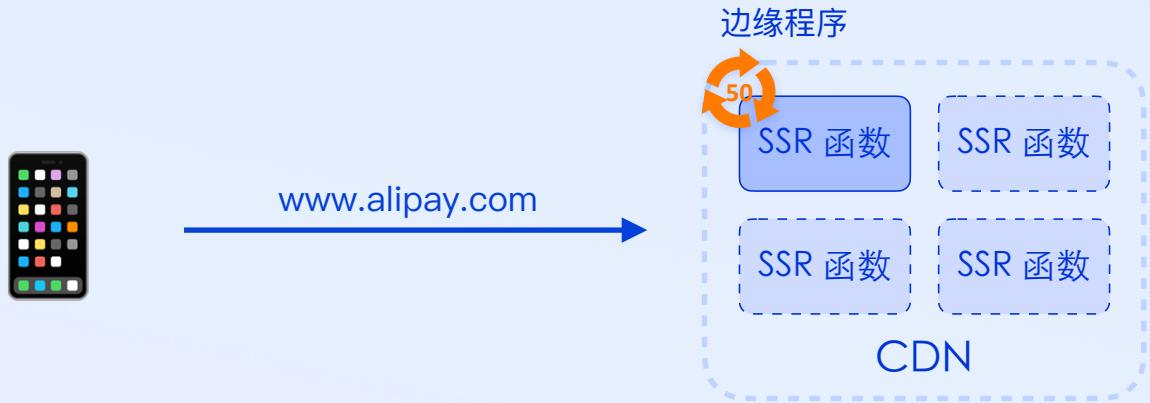
如何在 SSR 链路引入 CDN?

加入边缘程序



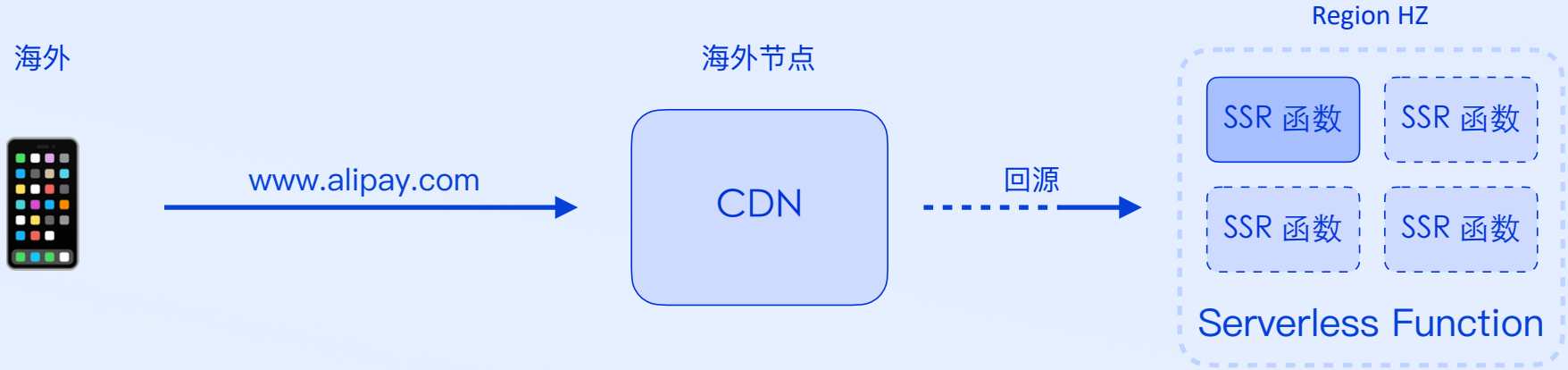
如何在 SSR 链路引入 CDN?

边缘程序的局限



边缘环境的计算资源有限
而 SSR 是 CPU 密集型任务

如何在 SSR 链路引入 CDN?



SSR 渲染结果是千人千面不可缓存的
无法利用 CDN 静态加速

如何在 SSR 链路引入 CDN?

一个典型 HTML 的特点

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" href="https://cdn.com/app.css">
</head>
<body>
  <div id="root"></div>
  <script src="https://cdn.com/app.js"></script>
</body>
</html>
```

拆分

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel="stylesheet" href="https://cdn.com/app.css">
</head>
<body>
```

纯静态可缓存

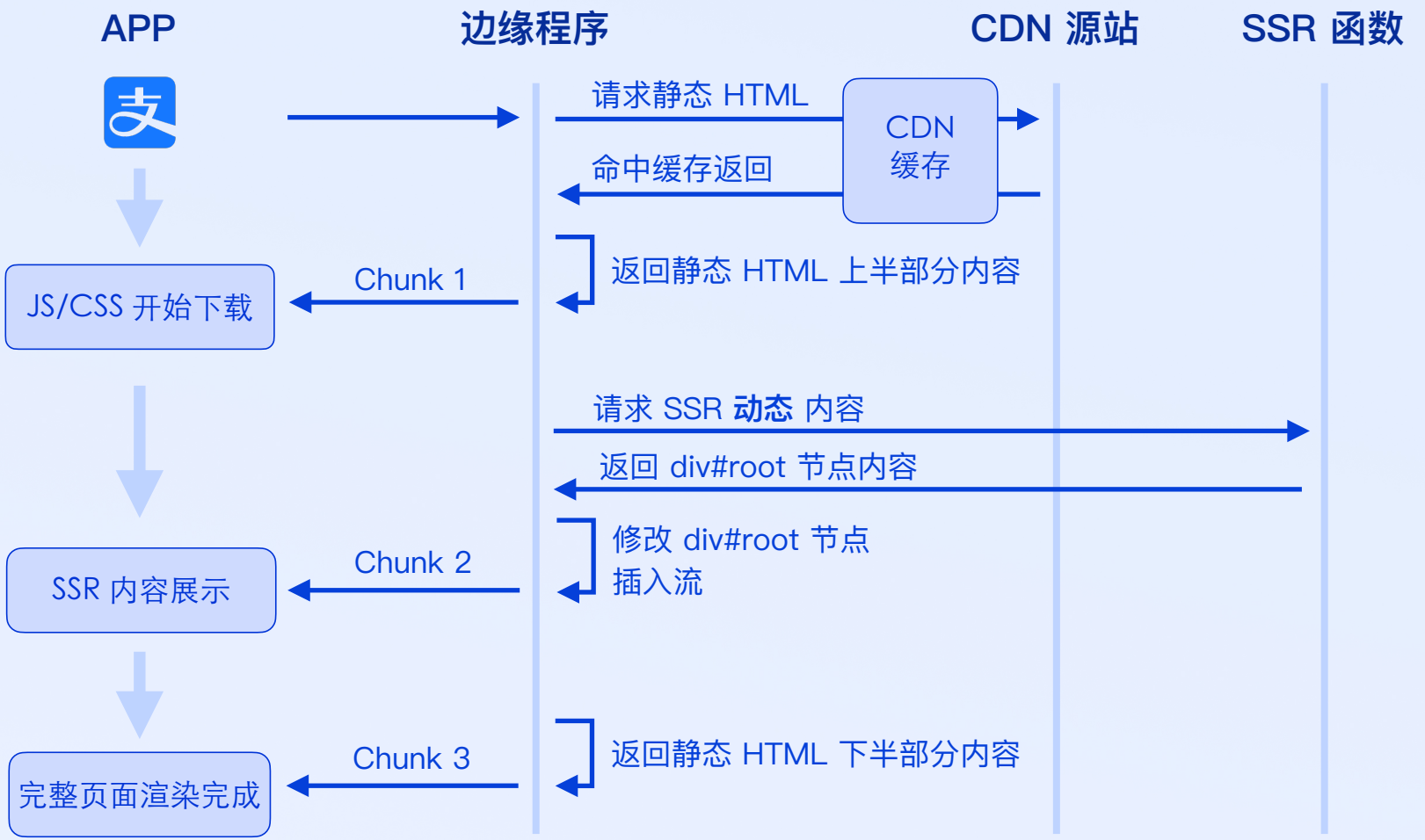
```html
<div id="root"></div>
```

需要替换为 SSR 动态内容

```html
  <script src="https://cdn.com/app.js"></script>
</body>
</html>
```

纯静态可缓存

结合边缘程序、CDN 缓存、函数 的 SSR

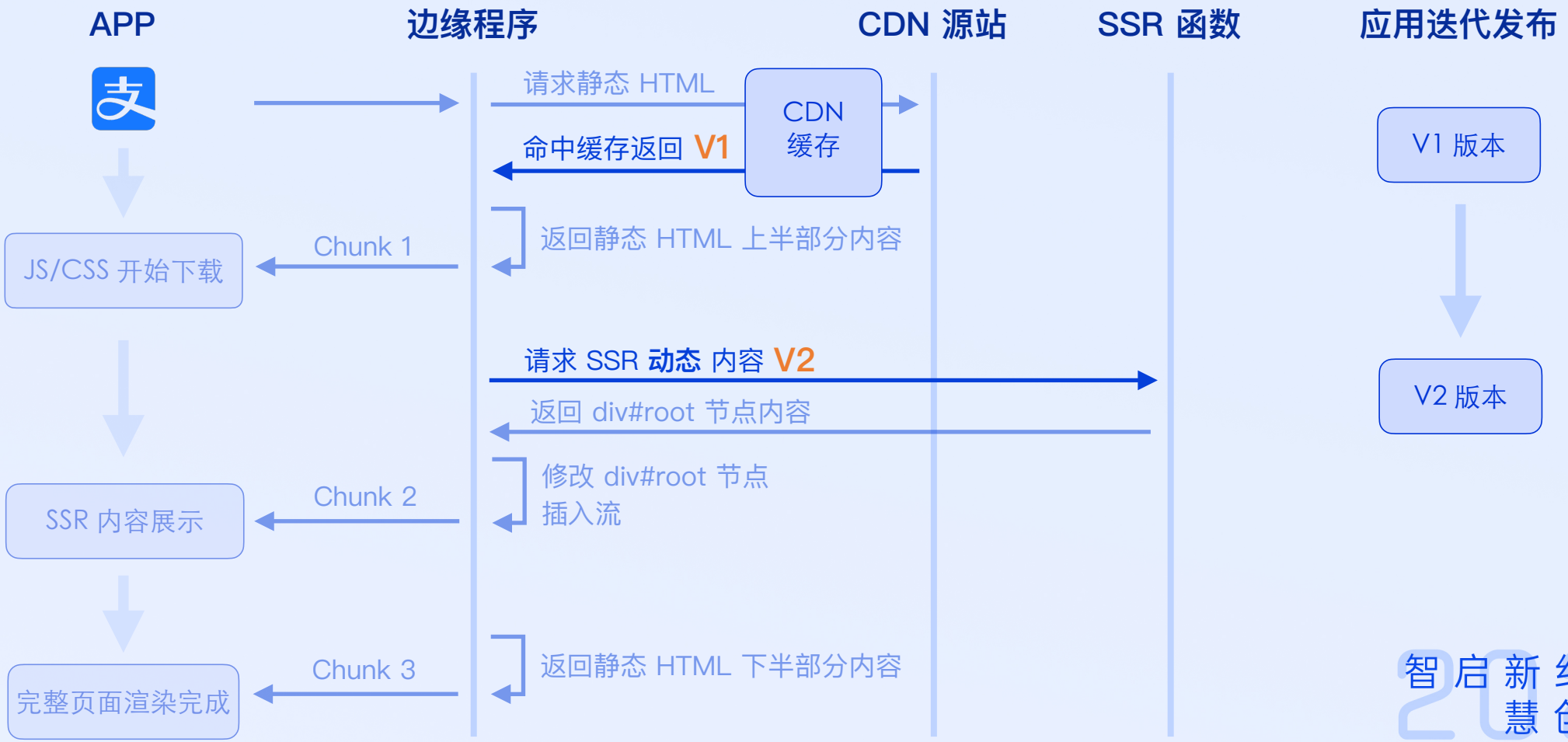


CDN 全球缓存加速

SSR 的动态渲染性

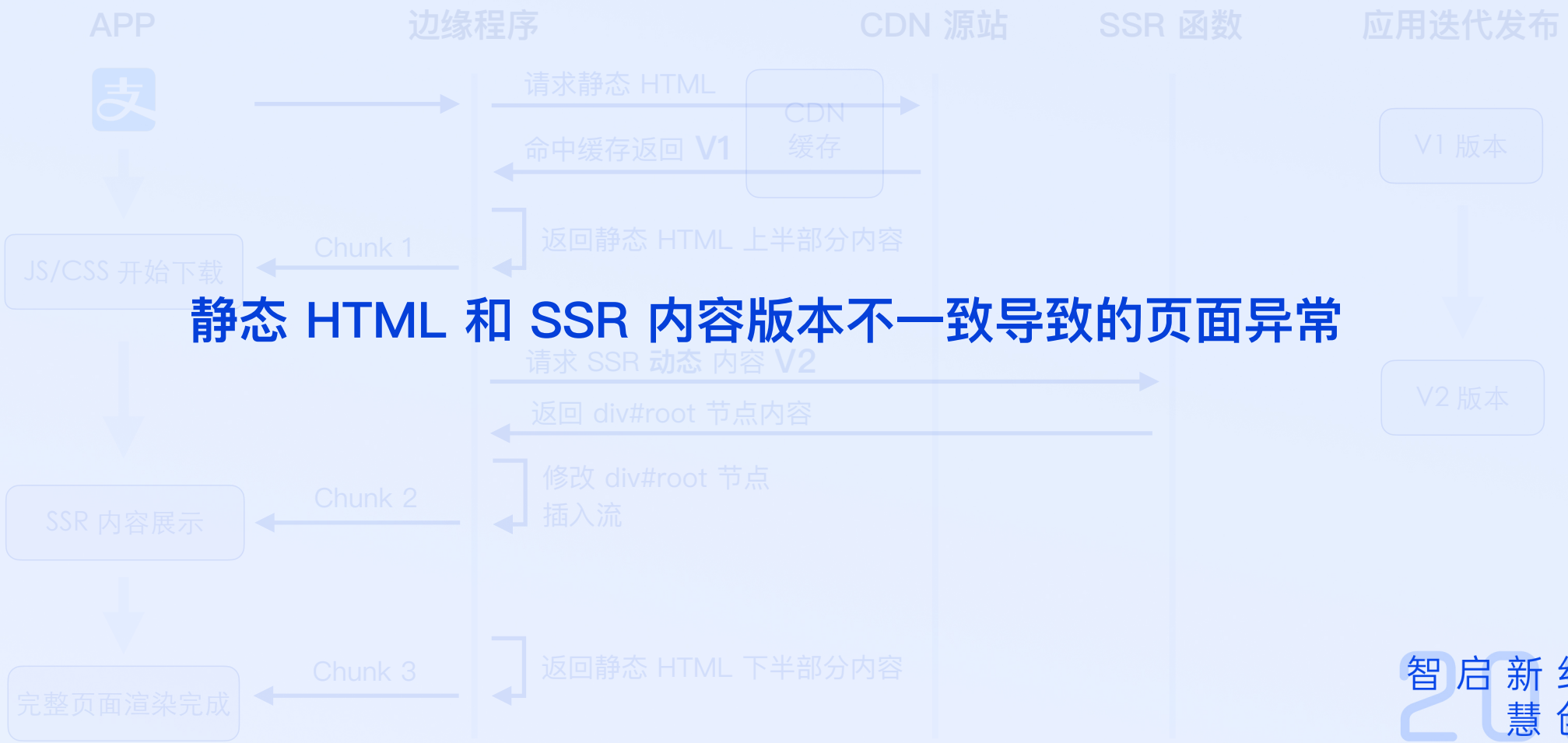
缓存并不是免费的

CDN 缓存与函数版本的一致性问题



缓存并不是免费的

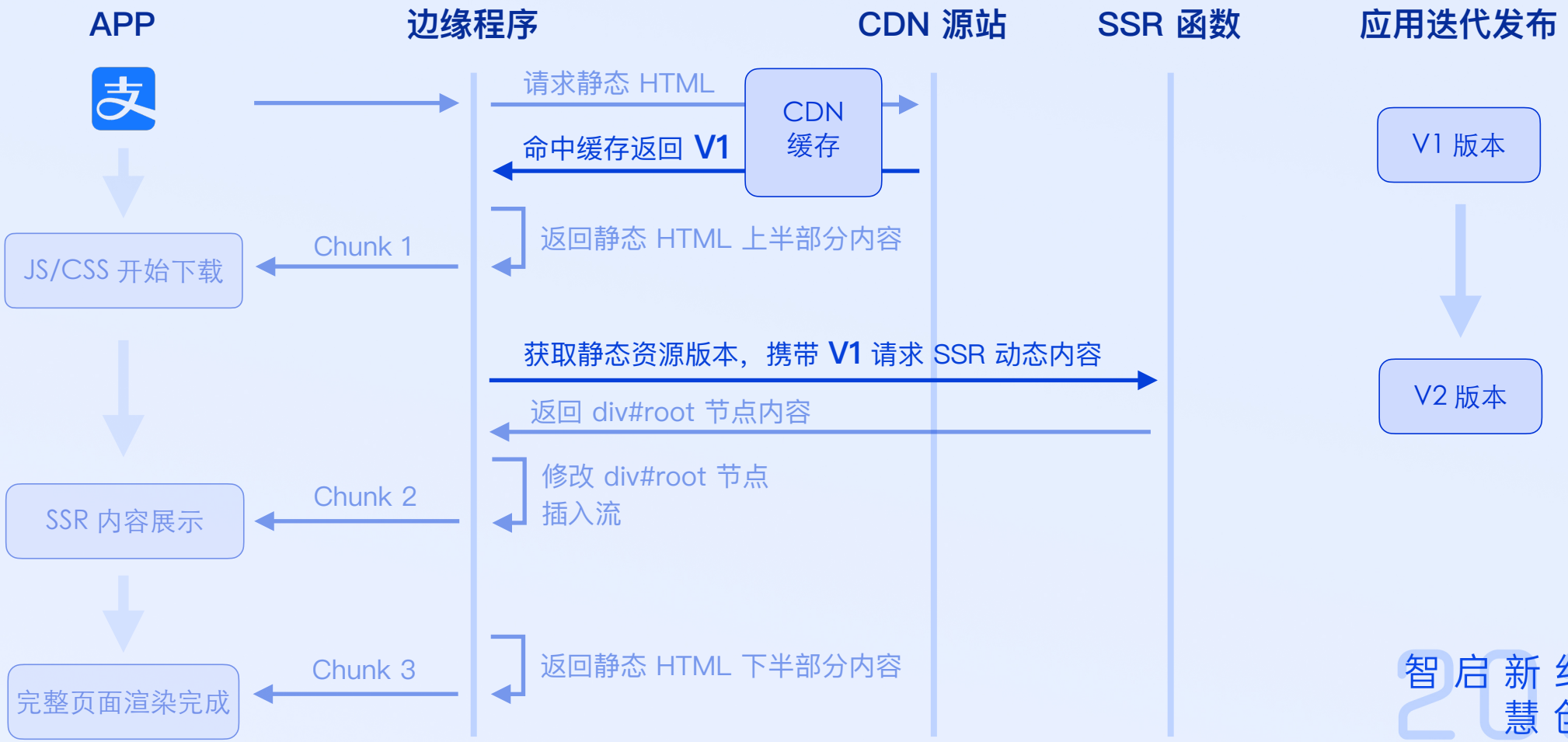
CDN 缓存与函数版本的一致性问题



静态 HTML 和 SSR 内容版本不一致导致的页面异常

缓存并不是免费的

CDN 缓存与函数版本的一致性问题



结合 CDN 的 SSR 的特点

TTFB 足够快?

基于 CDN 的全球化静态缓存

FCP 足够快?

取决于应用的静态 HTML 是否有骨架屏

SSR 足够快?

每次都需要执行完整渲染存在浪费

支付宝内典型页面

按照内容是否可缓存进行拆分

不含用户信息，千人一面
CDN 可缓存

含用户信息，千人千面
纯动态 CDN 不可缓存

不含用户信息，千人一面
CDN 可缓存

动静分离 SSR

基于 Suspense 的动静边界描述

支付宝内典型页面

静态可缓存部分

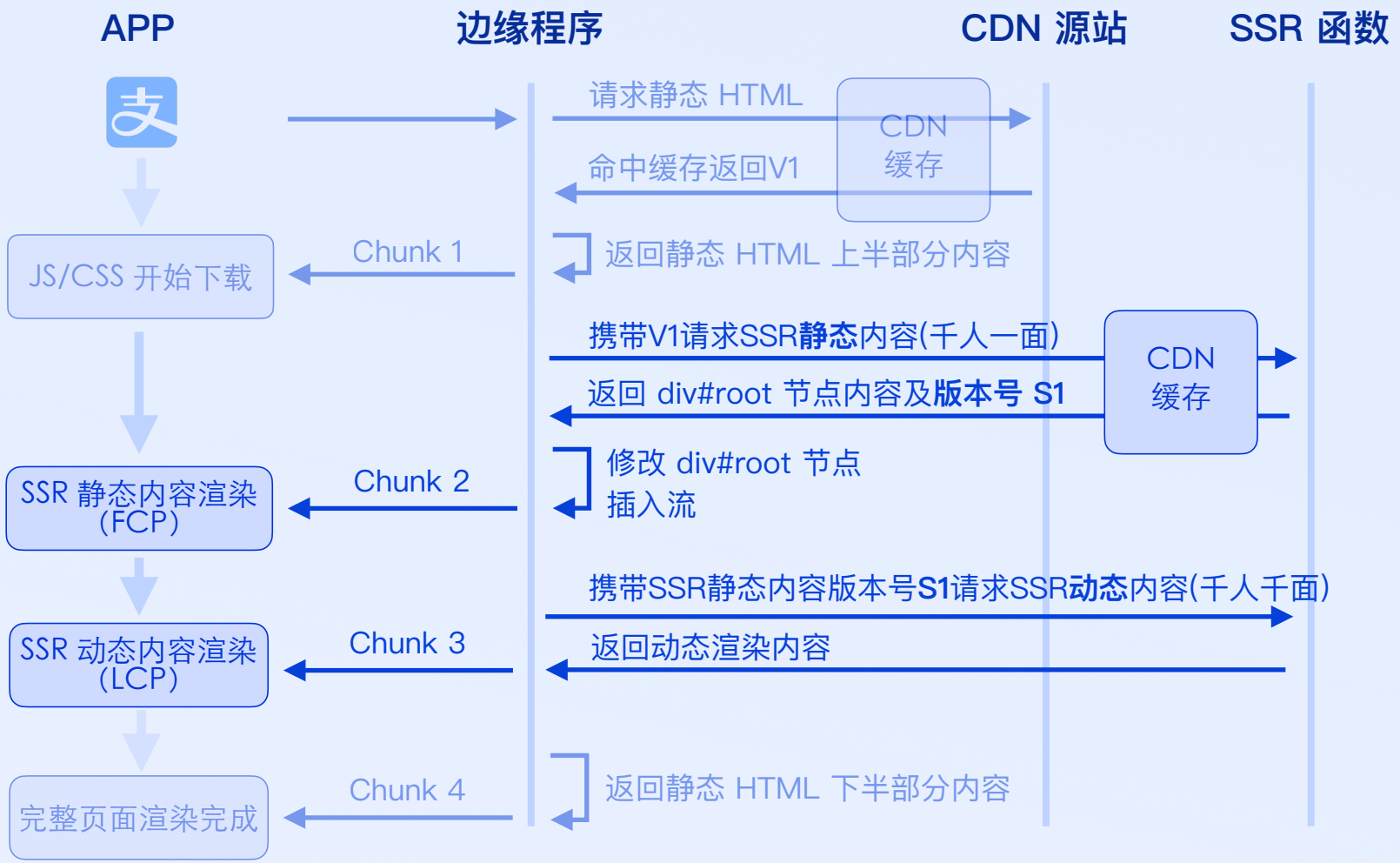
动态不可缓存部分

静态可缓存部分

```
export default function Page() {
  return (
    <main>
      <header>
        <Banners list={serviceProviderList}/>
      </header>
      <Suspense fallback={<InsuranceSkeleton />}>
        <Insurance />
      </Suspense>
      <footer>
        <Statements />
      </footer>
    </main>
  );
}
```

动静分离 SSR（PPR）

基于 Suspense 的动静边界描述



动静分离 SSR

Partial Prerendering（PPR）

TTFB 足够快? 

基于 CDN 的全球化静态缓存

FCP 足够快? 

取决于应用的静态 HTML 是否有骨架屏

SSR 足够快? 

每次都需要执行完整渲染存在浪费

SSR/RSC 性能之外的诉求

看上去旗舰机不太需要 RSC/SSR，如何实现规则化渲染?

基于边缘程序的规则渲染

支付宝上终端环境的多样性：

1. 高端机型性能强劲，CSR 就很快
2. 千元机性能堪忧
3. 不同版本支付宝差异，又无法强制升级
4. 业务对 A/B Test 的诉求

基于运行时的动态渲染决策

基于边缘程序的规则渲染

渲染规则描述

```
{
  "version": "v1",
  "adaptiveRoutes": {
    "index.html": {
      "rules": [
        {
          "match": {
            "os": { "name": "ios", "version": "<13" },
            "alipay": { "version": "<12.0.1" }
          },
          "target": "index.ssr.html"
        },
        {
          "match": {
            "os": { "name": "android", "version": "<13" },
            "alipay": { "version": "<12.0.0.1" }
          },
          "target": "index.ssr.html"
        },
        {
          "match": {
            "query": {
```

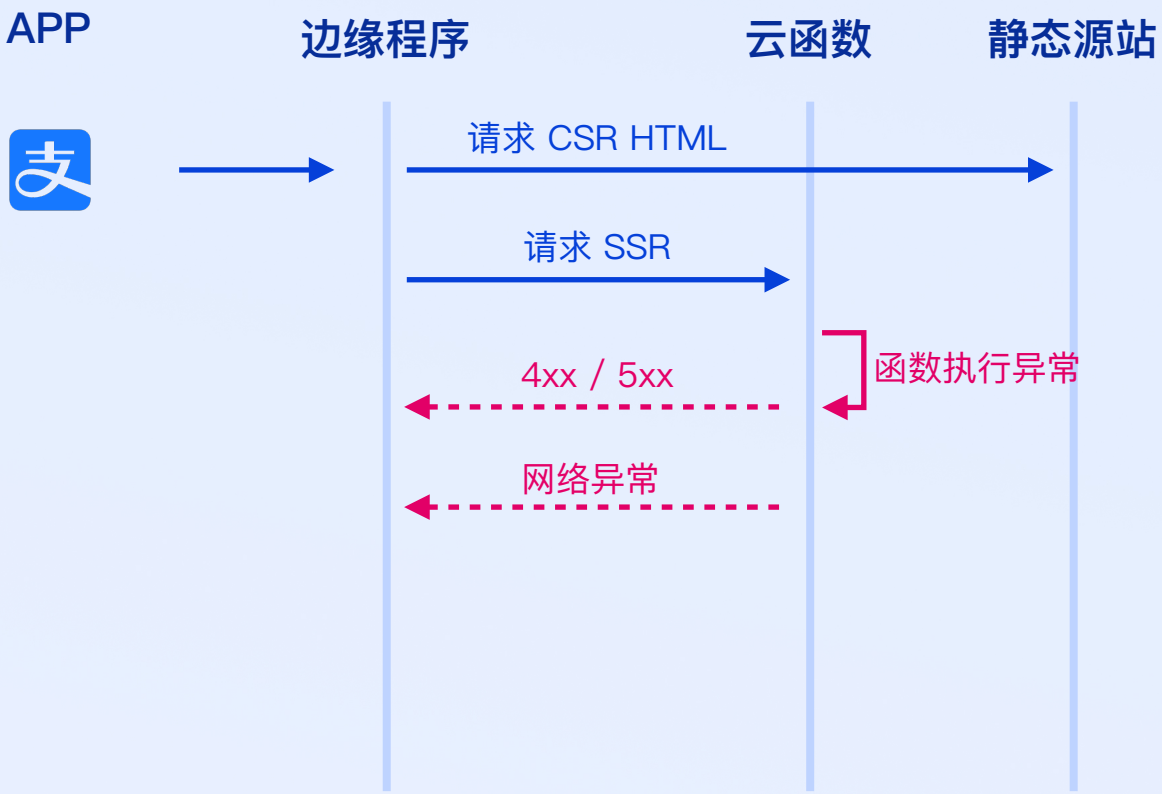
可以顺利在业务中落地了吗

业务在意什么?

👨‍💻 SSR/RSC 会不会变成另外一个运维负担?

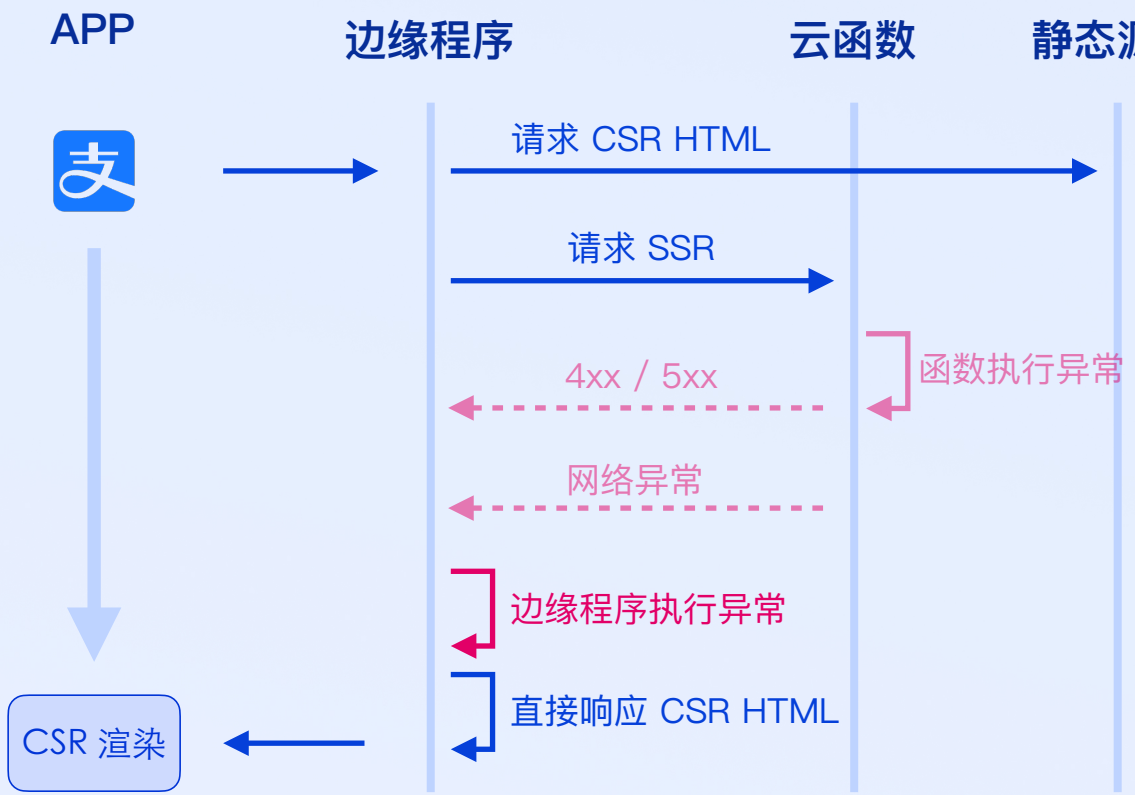
👨‍💻 我是前端，我不想五福期间值班!

服务端渲染可用性保障之容灾



云服务不稳定

服务端渲染可用性保障之容灾



只要 CSR 还能运行
SSR/RSC 服务就不怕挂！

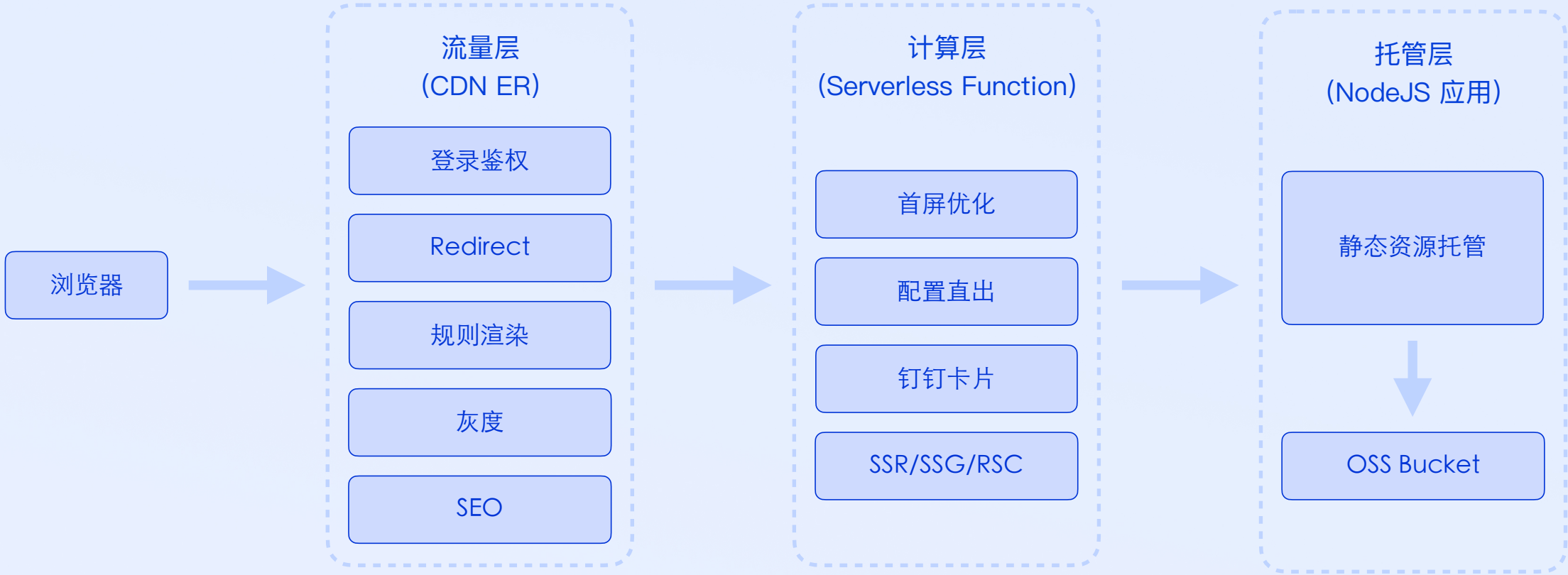
小结

1. 尽可能利用 CDN 缓存来提升云上 SSR/RSC 渲染性能
2. 但缓存并不是没有代价的，分布式缓存的数据一致性保障需要额外注意
3. 我们可以通过动静分离的方式，结合边缘程序极大提升 SSR 的首屏性能
4. 可用性非常重要，在支付宝内需要保证 SSR/RSC 链路上任意节点出现异常，都能自动 fallback 回 CSR

One more thing...

蚂蚁前端是如何支持 SSR/RSC 这类新的渲染场景拓展的?

Unio – 蚂蚁 Web 应用统一架构



落地数据

SSR FCP 提升

1. 支付宝外提升 40%
2. 支付宝内提升 30%

RSC LCP 提升

1. 支付宝内10% ~ 30%

落地业务

1. 春促、双十二、618
2. 五福
2. 消费圈、会员

谢 谢 观 看

THANKS