

23 | 热更新：如何搭建一个热更新平台？

2022-05-20 蒋宏伟

《React Native 新架构实战课》

[课程介绍 >](#)



讲述：蒋宏伟

时长 21:50 大小 20.00M



你好，我是蒋宏伟。

在第 21 讲中，众文和惠姝两位老师站在客户端的角度，介绍了自研 React Native 热更新需要做什么。今天，我会站在服务端的角度，聊聊如何搭建一个热更新平台。

React Native 的热更新实际上包括两部分：一部分是客户端逻辑，另一部分就是热更新平台。客户端部分需要 iOS、Android 方向同学的配合，热更新平台部分需要 Web、Node.js，甚至是 Java 方向同学的配合。

当然，只用一讲的内容，是很难把热更新平台讲透的。因此，我会先从整体上为你讲解热更新的原理和业内常见方案，当业内方案满足不了你的业务需求时，这时你就可以选择进行自研了。这一讲中我也给你准备了两套自研方案，你既可以选择无成本的 CDN 方案，也可以选择高可靠性的版本方案。

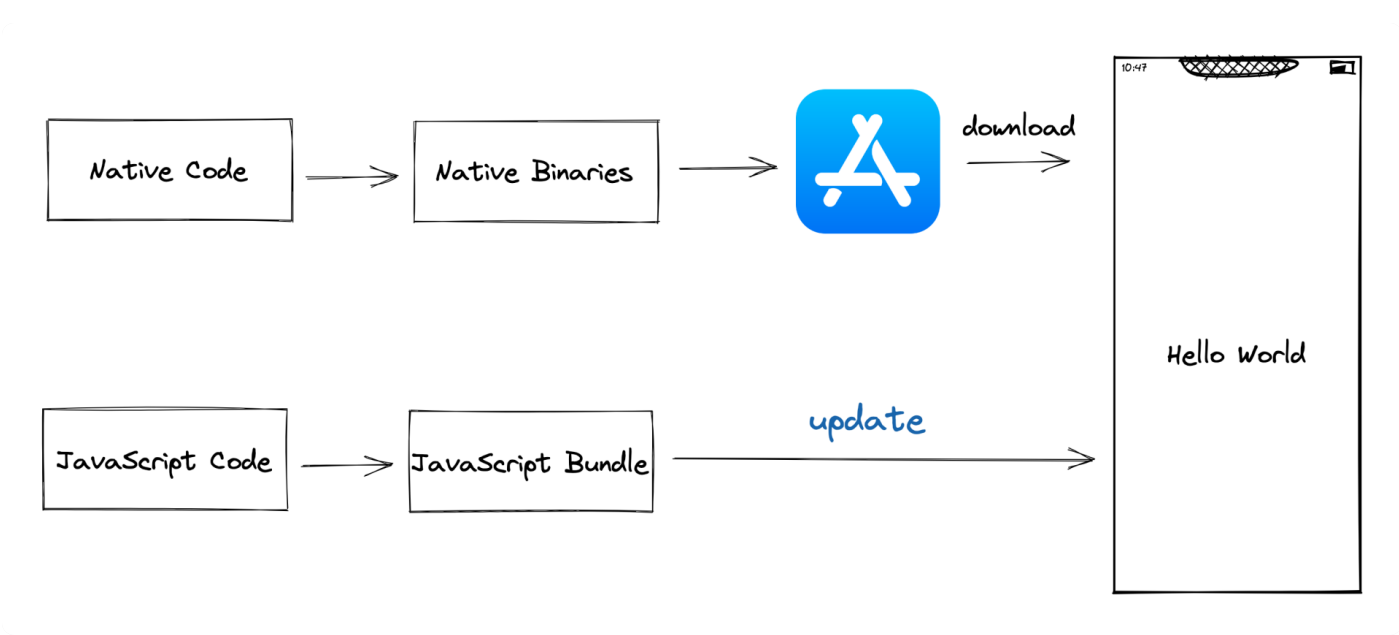
无论你的热更新需求是什么样的，总有一套方案是适合你的。

什么是热更新？

有些同学可能还对 **React Native** 热更新不太熟悉，我先简单介绍一下什么是热更新。

热更新，也叫做动态更新，它是一种类似 **Web** 的更新方式。相对于 **App** 的发版更新而言，热更新能提升业务的迭代效率。我们都知道，互联网业务讲究兵贵神速，如果业务能够通过热更新来快速发版和迭代，这就相当于在产品 and 用户之间搭建了一座能够随时通行的桥梁，代替了原来好几周才有一趟的渡轮。

那么，热更新和发版更新有什么不同呢？为什么热更新比发版更新快这么多呢？我给你画了这两种更新方式的原理对比图，你可以先看下：



发版更新，指的是你把 **React Native App**，当作 **Android App** 和 **iOS App**，按照 **Android**、**iOS** 上架流程，通过各自的应用商店进行更新。通常每个 **Native App** 都会有一个自己的上架节奏，可能是两周，也可能是 4 周。此外，从提交应用商店到审核通过，也需要等上几天时间。甚至，即便新版本上架了，用户更新到最新版本也需要一个过程，可能需要一个月的时间，新版本才能覆盖到 **90%** 的用户。

所以说，如果你的 **React Native App** 选择发版更新，就会受到发版节奏、审核耗时和版本覆盖耗时的影响，这些都会导致业务迭代速度变慢。

不过，React Native 的热更新就可以绕过应用商店直接进行更新。只要你的集成热更新功能的 React Native App，在应用商店上架过一次之后，后续你的业务都可以走你自己的热更新流程，再也不用依赖应用商店发版。这样你的业务就能随时上线，随时更新了。

也因为这个原因，热更新在国内也是大受欢迎。

热更新方案

既然热更新这么好用，那怎么把热更新用起来呢？业内都有哪些方案呢？当前，业内的主流方案有四种：[🔗 Code Push](#)、[🔗 Pushy](#)、[🔗 Expo](#) 和自研。

但由于国内网络环境的原因，访问国外的云服务速度比较慢，所以我不太推荐你直接使用 Code Push 和 Expo。Code push 是微软 App Center 的服务之一，它底层用的是微软自家的 Azure 云服务；Expo 使用的是亚马逊的 AWS 和 Google Cloud 云服务。

如果你不嫌麻烦，可以基于 Code Push 或 Expo 自行搭建。基于 Code Push 自行搭建的方案我其实不太熟悉，不清楚基于 Code Push 自行搭建需要多少成本。这里我就引用群友“问题本人”的回答，他说：

Code Push 可以自己搭建一套，哪怕你不是后端，前端自己去搭建，耗费个三两天就可以了。

如果你想使用 Expo 替换云服务商的方案，可以参考一下[🔗 官方文档](#)。

如果你嫌自己搭建太麻烦，你也可以看看 React Native 中文网提供的[🔗 Pushy](#)热更新方案。它使用的是国内的阿里云服务，且有比前两者更省流量的增量更新方案，应该是国内目前市面上唯一可以直接使用的开源热更新方案了。

不过，我最熟悉的热更新方案，还是自研方案。所以接下来，我就基于自研方案，和你介绍一下热更新的原理，帮你搞清楚热更新究竟是怎么一回事，自研热更新平台又需要注意些什么。

热更新平台的原理

说到自研热更新平台，如果你没有接触过，可能会认为很难，但真实情况真是如此吗？

其实，自研热更新平台的难度主要体现在**如何大规模应用**上。对于规模小的应用来说，搭建一个自研热更新平台并不难。

一个自研热更新平台，主要包括这两个部分：

- 打包服务：**Bundle Server**；
- 静态资源服务：**Static Server**。

所谓的打包服务，是将 **React Native** 项目中的所有 **JavaScript** 代码打包成一个 **Bundle** 文件的服务。而所谓的静态资源服务，是将 **Bundle** 文件分发给客户端的服务；当客户端拿到 **Bundle** 代码后，执行 **Bundle** 文件，就能渲染 **React Native** 应用 / 页面了。

这个流程有没有让你想起启动项目时的 **npm start**？是的，理论上，你在本地通过 **npm start** 启动的 **Metro** 服务时，**Metro** 服务就同时具备了打包和静态资源下发两种功能，再配合框架默认的代码加载功能，也能完成热更新。

也就是说，你找台服务器把 **React Native** 代码放上去，运行 **npm start** 命令，然后在客户端配置对应的 **ip** 和端口号，并把相关的调试开关给关了，这时访问 **npm start** 服务客户端也能把 **React Native** 页面渲染出来。然后你再用 **npm start** 启动的 **Metro** 服务，就具备了“热更新平台”最基础的两个功能了。

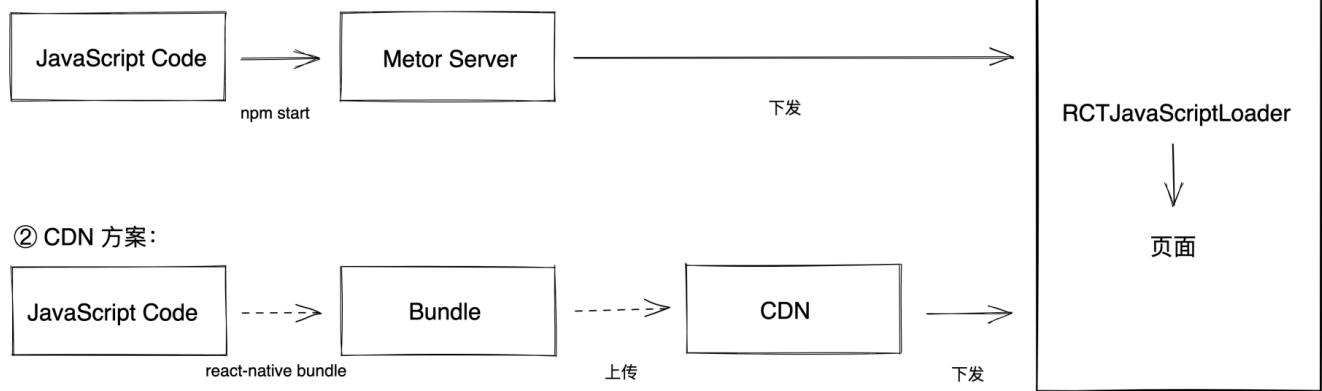
当然，这个最简单的热更新流程是不能跑在线上的，毕竟 **npm start** 的本意是用于调试的，它的首次加载耗时太长，扛不住高并发，而且服务可用性也是问题。

CDN 热更新方案

既然，**npm start** 的热更新方案不太靠谱，那有没有靠谱点的热更新方案呢？有，接下来我要给你介绍的 **CDN** 热更新方案就是既简单又实用。

我给你画了一张 **npm start** 方案和 **CDN** 方案的对比图，你看一下就能明白它们之间的区别了：

① npm start 方案：



在 npm start 方案中，Metor Server 提供了代码打包和 Bundle 下发的功能。而在 CDN 方案中，代码打包是通过 react-native bundle 命令提供的，Bundle 下发的能力是通过 CDN 提供的。

那么 CDN 方案到底是怎样的呢？我们简单看下它的操作流程。

首先，你可以通过 react-native bundle 命令，提前把 JavaScript 代码打包成一个 Bundle 文件，命令如下：

复制代码

```
1 npx react-native bundle --entry-file index.tsx --dev false --minify false --bun
```

通过上述命令打包出来的是 index.bundle 文件，本质是一个可执行的 JavaScript 文件。

如果你使用的是 Hermes，那么你还需把 JavaScript 文件转成相应的字节码文件。Hermes 提供了把 JavaScript 文件转成字节码文件的方案，你可以先按照 [🔗文档](#) 搭建 Hermes 环境，然后执行如下命令进行转换：

复制代码

```
1 hermes -emit-binary -out ./build/index.hbc ./build/index.bundle
```

转换完成后，你就有了一个 .hbc 的字节码包了，其中 .hbc 的意思是 Hermes Bytecode。

完成打包之后的第二步，就是将包上传到 CDN 上。这里你可以选择阿里云、腾讯云或者是你们公司内部提供的 OSS 和 CDN 服务。

我们以阿里云为例。在第一次使用时，你可以先 [将包文件上传到 OSS](#)，然后 [开启 CDN 加速](#)。这两步和你选择的 CDN 服务商有关系，你参考相关文档操作即可。

完成文件上传 CDN 这一步后，你会得到一个 CDN 地址，你可以用该地址来访问你的文件，例如：

```
1 https://static001.geekbang.org/resource/rn/index.bundle
```

 复制代码

拿到包地址后，热更新最后一步是，在客户端请求和加载该地址的 `.bundle` 文件或 `.hbc` 文件，这样就完成热更新的整个流程了。

以上就是第一次热更新的流程。我们第一次把包上传到 CDN 时，CDN 文件是新创建的，但如果代码再次更新了，要怎么更新 CDN 上的包文件呢？

更新版本的方案就是**用新包把老包给覆盖掉**。

在 CDN 方案中，CDN 地址是固定不变的，老包用的地址是 `/resource/rn/index.bundle`，那么新包用的地址也得是 `/resource/rn/index.bundle`。例如，你可以使用阿里云提供的 [CDN 刷新](#) 功能，用新包把老包给覆盖掉。

CDN 方案非常适合小流量的业务。而且它解决了 `npm start` 方案存在的三个问题，包括首次加载耗时长的问题、扛不住高并发的的问题和服务可用性低的问题。

这是因为，热更新包是提前打好并放到 CDN 上的，无需像 `npm start` 一样，在首次加载时需要长时间的等待打包。其次，CDN 通过上千台分布在全国各地的机器，解决了高并发的的问题，即便同一秒钟有上万的用户并发访问，CDN 都完全能抗住。并且，你不需要使用 Java 或者 `node.js` 去搭建高可用的热更新后台，像阿里云、腾讯云这种大家都用的 CDN 服务，它们的可用性比我们自己搭建的服务可高太多了。

而且，类似阿里云这样的云服务提供商，还贴心地提供了 [🔗 自定义脚本发布静态资源](#) 的功能，这又进一步降低了部署的成本。

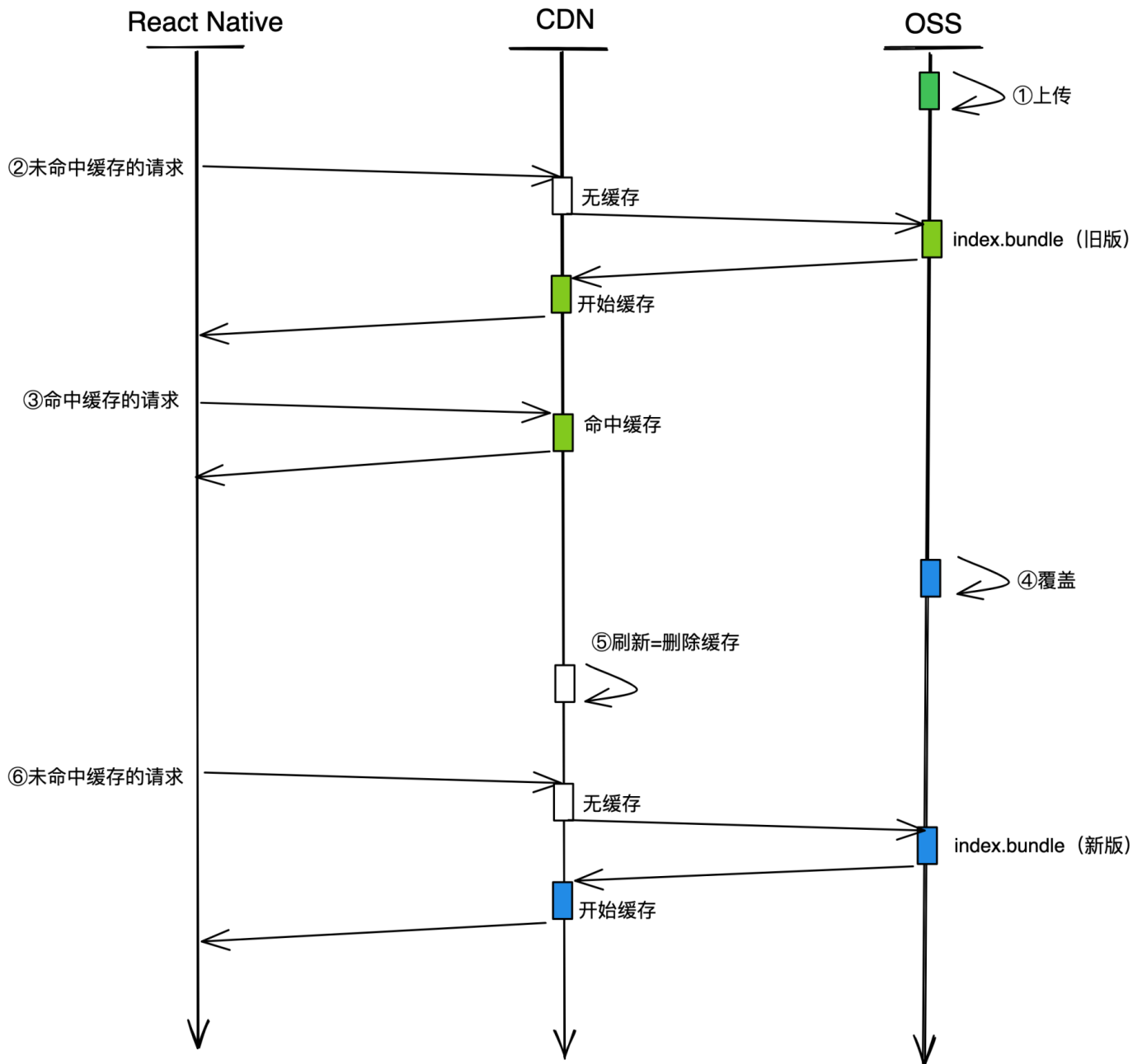
因此，如果你的业务流量不是很大，你完全可以依托于阿里云、腾讯云这样的 CDN 服务，零成本自建热更新平台。

纯 CDN 方案的弊端

但是对于大流量业务，我并不推荐你用纯 CDN 方案，为什么呢？

因为纯 CDN 方案，会存在**几分钟的更新延迟**的问题。在小流量业务中，这种几分钟的更新延迟不是什么问题，但是对于大流量业务来说，如果线上出现了一个重大 BUG，需要等几分钟才能完全回滚，那么对用户或者收入的影响会很大。

为什么纯 CDN 方案会存在几分钟的更新延迟问题呢？我们来看下一下 CDN 方案的时序图：



上述时序图中，涉及 React Native App、CDN 边缘节点和 OSS 源站，以及 index.bundle 包文件的两个版本。旧版本包是绿色的，新版本包是蓝色的。

将旧版本更新为新版本的本质是：删除 CDN 中缓存的旧版资源，当 CDN 中没有缓存了，这时来自用户的请求才不会命中 CDN 中的缓存，而是到 OSS 上拉取最新的资源，返回给 React Native App。

然而，我们都知道 CDN 指的并不是某一台具体的机器，它指的是上千个分布在全国各地的节点网络。当我们使用 CDN 的刷新能力时，实际上是删除上千个节点中的缓存。一位负责 CDN

的同学告诉我，要把这上千个节点的缓存都删除干净的时间，最长可能需要个 5 分钟吧，而且还不敢保证 5 分钟的时效性。

因此，在这 5 分钟内，会存在三种情况：情况一，命中老版缓存；情况二，未命中缓存重新拉取新版资源；情况三，命中新版缓存；

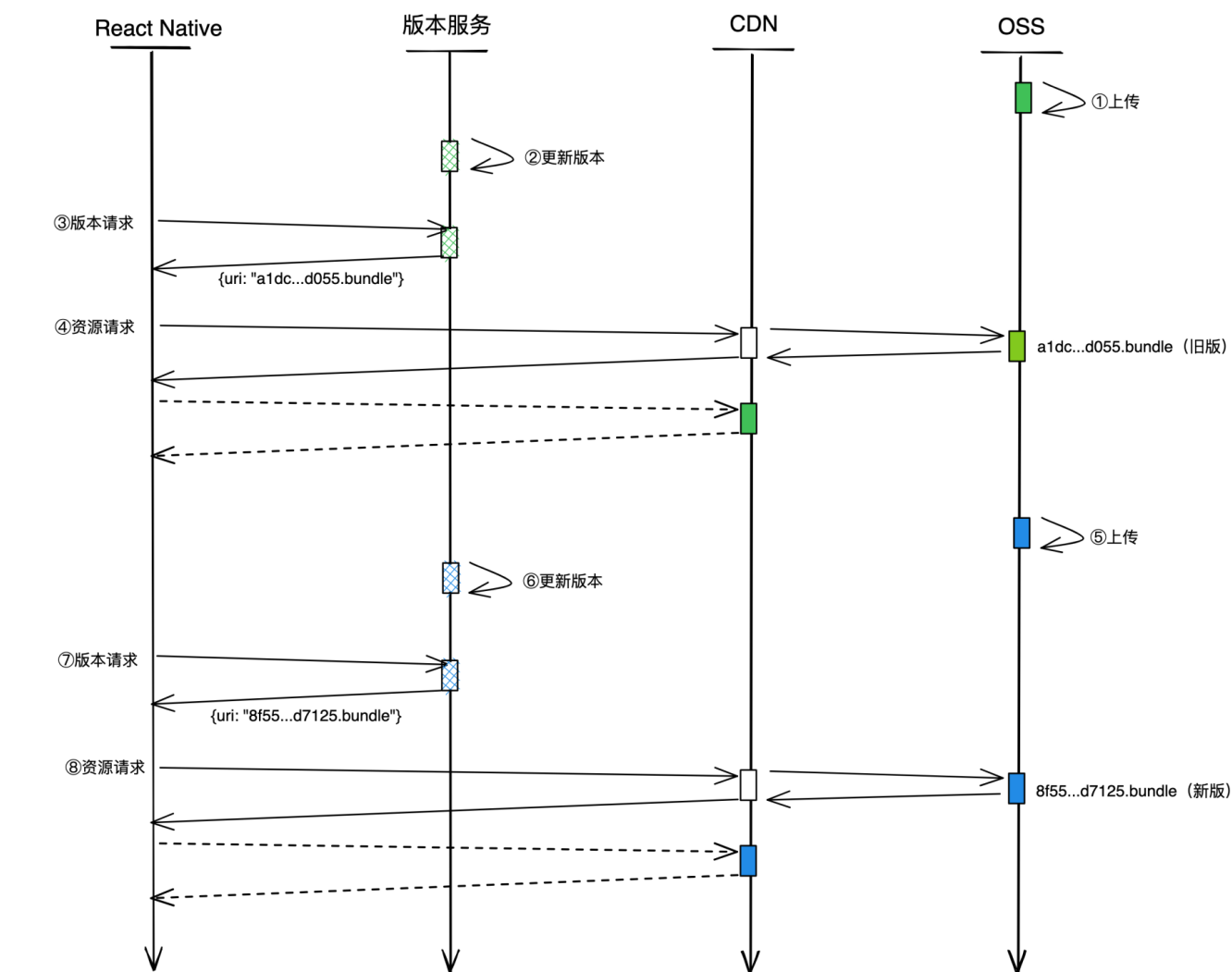
这也意味着，在你享用 **CDN** 的低成本热更新方案的同时，你的业务得能够接受 5 分钟的渐进式的更新延迟，或者说 5 分钟的渐进式的修复 **BUG** 的延迟。

版本方案

CDN 方案有最长 5 分钟延迟，那有没有解决办法呢？

解决方案就是**多发一次版本请求**。解决思路是这样的，既然上千个节点 **CDN** 更新有延迟，那么就自己搭建一个版本服务，资源依旧上传 **CDN**，但用版本服务来控制更新。我们不妨把该方案叫做版本方案。

版本方案的时序图如下：



增加一个版本服务后，你可以看到整体流程发生了一些变化。纯 CDN 方案的更新方式采用的是覆盖更新，版本服务方案采用的是**告知更新**。

所谓的“告知更新”是什么？我们一步一步把它讲清楚。

第一步，上传 Bundle 到源站，也就是 OSS。先在将本地打包好 Bundle 文件，并将文件命名为“MD5”.bundle 上传到 OSS 源站。此时理论上，只要 Bundle 内容发生了变化，那么生成 MD5 值就是不一样的，用 MD5 作为文件的命名能保证文件的唯一性：

复制代码

```
1 // ①以 MD5 作为文件名
2 a1dc...d055.bundle
```

第二步，正式发版上线。当你要正式上线时，点击上线按钮，告诉版本服务最新的线上 Bundle 的名字，比如 a1dc...d055.bundle，这时版本服务会在内部通过 mysql 或 redis 把线

上最新文件名给记录下来：

 复制代码

```
1 // ②记录线上版本
2 online = "a1dc...d055.bundle"
```

第三步，React Native App 发起版本请求。由于只有一个版本服务，不会存在 CDN 上千个节点在某一时刻不同步的问题，版本服务会直接把最新的 Bundle 名字告诉 React Native 应用：

 复制代码

```
1 // ③下发版本名字或 CDN 地址
2 {uri: "a1dc...d055.bundle"}
```

第四步，React Native 发起 CDN 资源请求。资源请求会先询问某个 CDN 的边缘节点，如果该边缘节点没有缓存，则会去源站拉取 a1dc...d055.bundle；如果该边缘节点有缓存，则直接返回：

 复制代码

```
1 // ④请求 CDN 资源
2 https://static001.geekbang.org/resource/rn/a1dc...d055.bundle
```

然后的第⑤⑥⑦⑧步，其实和①②③④步是一样的，唯一不同的是 Bundle 文件的名字变了。整个流程如下：

 复制代码

```
1 // ⑤新包的 MD5 名字
2 a1dc...d055.bundle
3 // ⑥更新最新包名
4 online = "a1dc...d055.bundle"
5 // ⑦下发新版本名字或 CDN 地址
6 {uri: "a1dc...d055.bundle"}
7 // ⑧请求新版 CDN 资源
8 https://static001.geekbang.org/resource/rn/8f55...d7125.bundle
```

由于①②③④步的包名和⑤⑥⑦⑧步的 **CDN** 包名不一样，新包和旧包是同时存在于 **CDN** 上的，所以也就不存在要把上千个 **CDN** 节点的资源在同一时刻进行更新的问题了。

不过，虽然版本方案解决了更新延迟的问题，但它也不是没有弊端的。

首先，页面加载耗时会增加 **200ms** 左右，这是因为更新的流程多了一个版本请求。一个版本请求平均耗时大概 **200ms**，所以页面整体耗时也就增加了大概 **200ms**。其次，热更新平台的开发复杂度会高很多，你得提供高可用的热更新服务，并且能够抗住业务的高并发。

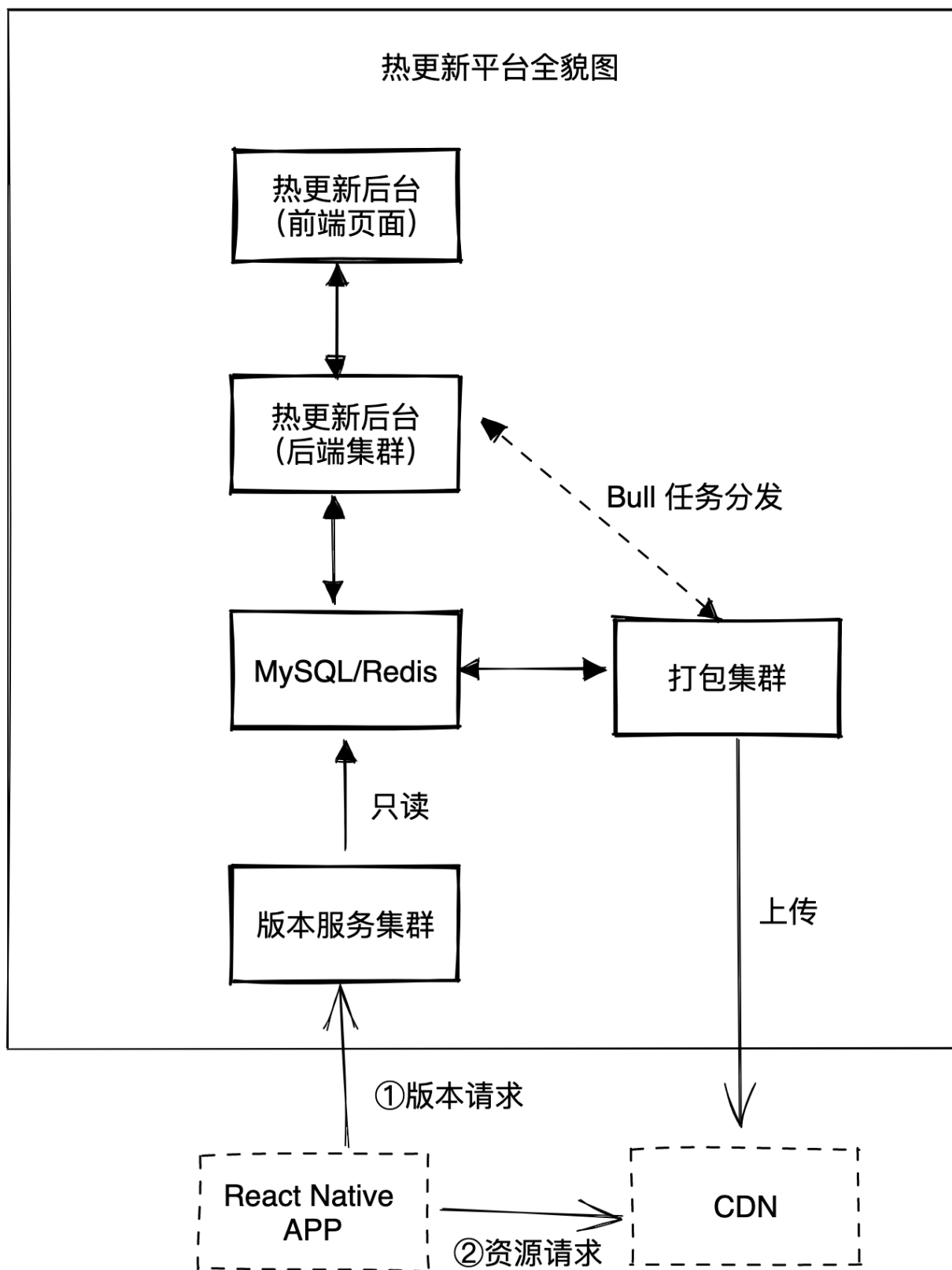
热更新平台全貌

无论是 **CDN** 热更新方案，还是版本方案，它们都不是完整的自研热更新方案，只是如何更新资源的一个切面，这个切面只解决了最基础的资源更新问题。

从我这些年来负责热更新平台的经验来看，自研热更新平台的核心难点在于，它需要你对前端、**node.js**、**React Native**，甚至 **Java** 都有所了解，特别是偏后端领域的知识。因为你需要解决以下问题，包括：

- 如何支持多人的并行打包；
- 如何支持多人的并行测试；
- 如何保障 **CDN** 资源更新成功；
- 如何保障版本服务的高并发、高可用。

为了解决以上问题，我也根据我的经验，帮你设计了一张热更新平台的全貌图：



你可以看到，热更新平台整体上包括以下几个部分：

- 热更新平台后台服务：一般两台机器就行，它提供打包、测试、上线、权限管理和相应的前端页面展示等能力。

- **MySQL、Redis**: MySQL 提供持久化存储能力, Redis 用于缓存用来抗高并发, 这里推荐用成熟的相关服务就行, 不要自行搭建。
- **打包集群**: 独立集群, 至少两台机器, 具体看情况而定, 用于支持多人的并行打包。把它独立出来的原因是, 打包是非常消耗 CPU、内存资源的任务, 和其他服务混在一起容易导致其他服务卡顿。
- **版本服务集群**: 独立集群, 用于提供能支持高并发的版本服务。如果你有 node.js 抗高并发的经验, 你可以用 node.js 来做, 或者你可以找 Java 同学帮忙一起负责实现。

当然, 有了热更新全貌图之后, 你还需要把它搭建起来。搭建热更新平台, 我推荐的关键技术栈有:

- [NestJS](#): 它是一个 Node.js 框架, 它能提供高效、可靠和可扩展的后端服务。
- [Bull](#): 它是一个任务队列库, 它能帮你解决热更新平台后台集群如何向打包集群发布打包任务的难题, 让你的平台可以支持多人并行打包。

我知道, 仅仅凭借热更新平台全貌图和核心技术, 很难让你复刻出一个热更新平台来。但有了这些关键的设计思路, 至少能够保证你在自研热更新平台时, 不会在主要的设计方向上跑偏。

总结

要实现 React Native 热更新功能, 有四种思路 [Code Push](#)、[Pushy](#)、[Expo](#) 和自研。

如果你选择自研 React Native 热更新功能, 这就需要 React Native 热更新平台和 Native 热更新模块的紧密配合了。

自研 React Native 热更新功能, 并不一定需要搭建一个热更新平台, 你也可以采用纯 CDN 方案, 比如你可以利用阿里云提供的静态资源部署能力和对应的 CDN 服务。

如果决心要完全自研一个热更新平台, 那么你最好找一个对后端技术比较了解同学一起来设计, 我也为你提供了一个热更新平台的全貌图, 你可以用它来帮你进行整体设计。

你或许发现了, 学到这一讲时, 所谓的 React Native 知识, 并不是局限在 React Native 本身, 它还包括了客户端知识、前端知识和后端知识。虽然在一个大团队中大家各自分工协作,

负责好自己擅长的技术领域即可，但也需要一个各方面都懂一些人同学，能把大家给组织起来。

能牵头把自研热更新平台落地，是一个非常考验技术架构能力和产品设计能力事情，如果你有机会能够主导，相信会对你的技术能力会有极大提高。


作业

很多人担心使用 **React Native** 热更新会导致苹果审核不通过，你有没有遇到过这类问题呢？在热更新技术方案的设计上，又应该如何规避审核风险呢？

欢迎在评论区分享，咱们下一讲见。

分享给需要的人，Ta订阅超级会员，你最高得 50 元

Ta单独购买本课程，你将得 20 元

 生成海报并分享

 赞 0  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

[上一篇](#) 22 | 自定义组件：如何满足业务的个性化需求？

精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。