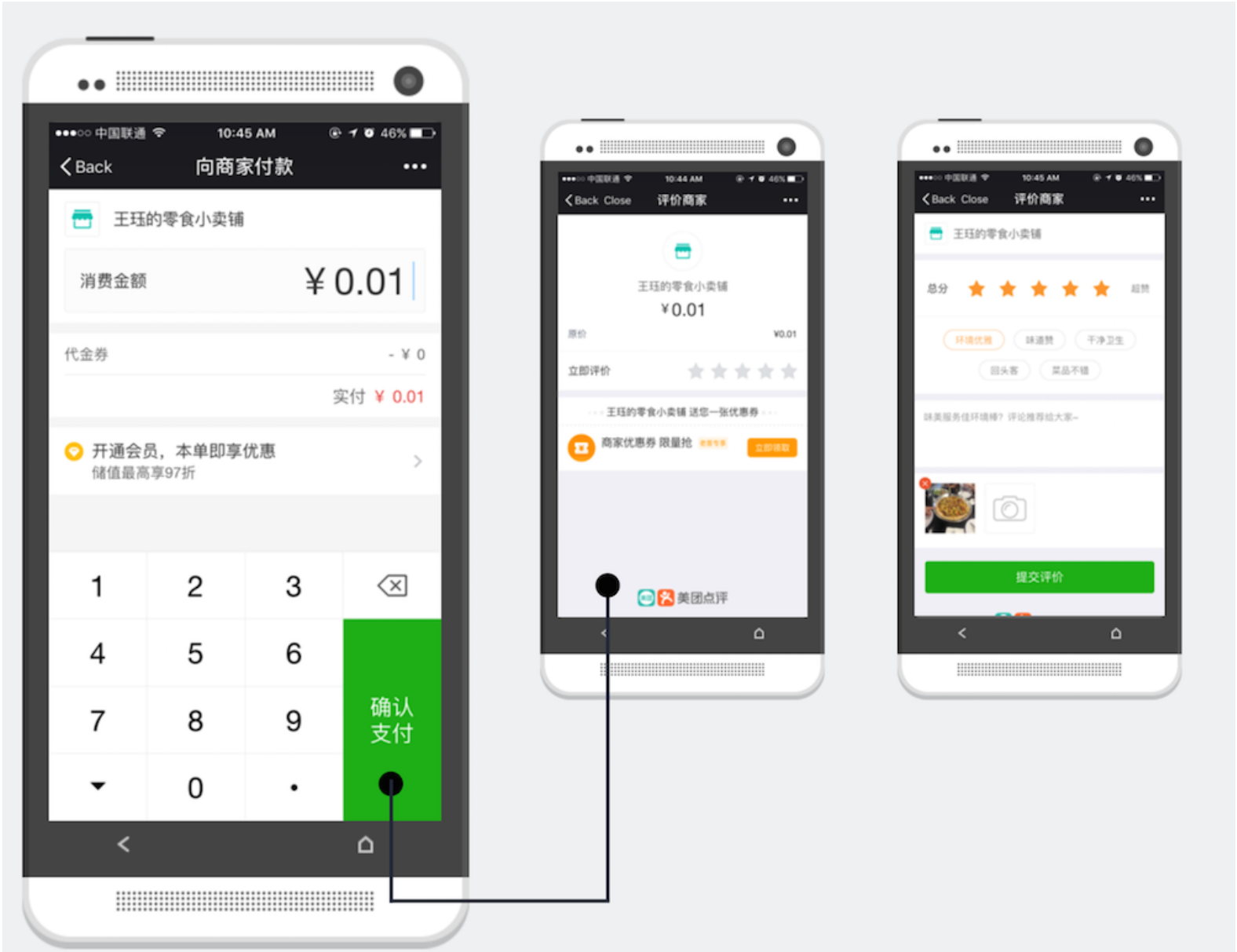
美团金融扫码付静态资源加载优化实践 (https://tech.meituan.com/2017/12/23/qrcodepayment-static-optimize.html)

2017年12月23日　作者: 孙辉 李罡　**文章链接** (https://tech.meituan.com/2017/12/23/qrcodepayment-static-optimize.html)　6375字　13分钟阅读

扫码付项目是美团金融智能支付团队面向 C 端消费者推出的一款 H5 融合支付类的产品，消费者在商家消费之后，可使用多种 App 进行扫码支付，同时可对商家进行评价，支持美团、大众点评、微信、支付宝、美团钱包等多种 App，目前业务日均 PV 千万级。如下图所示：



扫码付页面

扫码付页面

接入扫码付的商家大多数位于购物中心、写字楼等人口密集的室内空间。网络链路复杂、相对开阔的地区网络质量较差，为了减轻网络条件的影响，我们使用团队之前实现的模块加载器 ThunderJS。通过 **字符级增量更新** 减少文件传输大小，节省流量、提高

页面成功率和加载速度。其中增量计算能力由美团平台的静态资源托管方案 Build Service 支持。

我们曾经在 《 **美团智能支付背后的前端工程师**🔗🔗 **(https://juejin.im/post/58be3fac2f301e006c784733)** 》介绍过我们的前端服务架构，如下图：

架构图

架构图

ThunderJS（团队内部实现的一款 CMD 模块加载器） 属于其中非常重要的一环，集成在脚手架中为井喷的业务发展提供了基础。相比业界其它模块加载器，ThunderJS 定制加强了与静态资源托管（公司自研的Build Service）结合的能力，能够让我们对静态资源的加载进行针对性的优化，而在 C 端项目中，静态资源的加载优化是我们尤为重视的。

扫码付项目中也使用了ThunderJS，随着业务规模的持续增长，ThunderJS 的方案也在不断优化，本文主要介绍基于 ThunderJS 和 Build Service 的产品优化方案，希望为大家优化项目的静态资源加载提供更多思路。

最初的方案

ThunderJS 工作流程

ThunderJS流程图

ThunderJS流程图

ThunderJS 将页面的 JS 资源及版本信息存储在 LocalStorage 中。页面加载时通过线上版本和本地版本来判断是否需要更新，如果需要则会尝试进行 Diff 合并请求并 Patch 到本地资源。不需要更新则直接执行 LocalStorage 中缓存的数据，并且在合并请求失败的情况下会逐一加载单文件。

是否需要更新

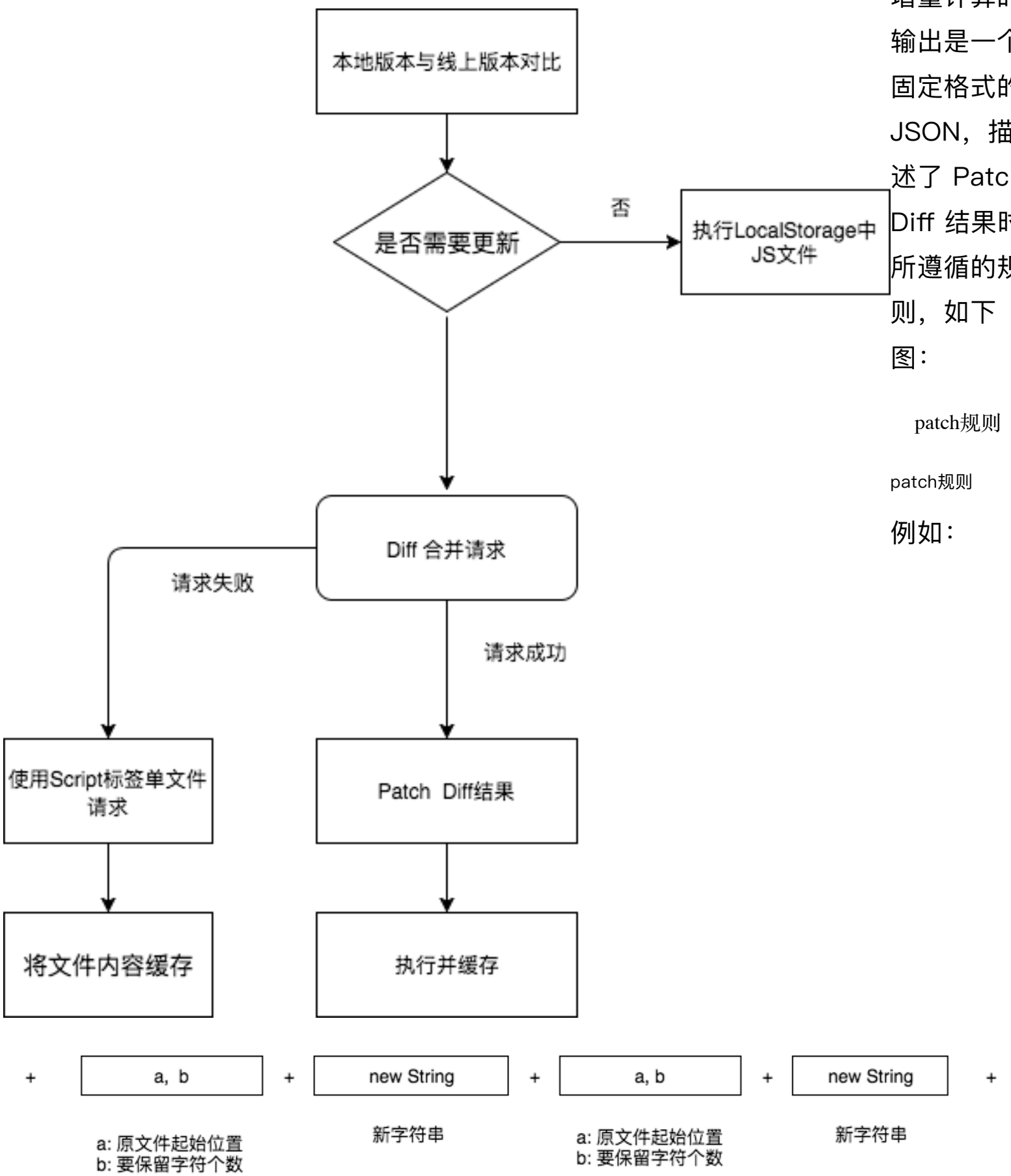
判断**是否需要更新**的具体原则如下：

- 该文件名在线上版本和本地版本中都存在。
- 该文件的版本在线上和本地中一致。
- 该文件存在于LocalStorage中。

Diff 合并请求 与 Patch Diff 结果

流程图中的 **Diff 合并请求** 是指在一次请求中输出多个文件的增量计算结果，请求合并是一种常用的 Web 资源优化策略，拼接多个相同媒体类型的资源经由单个请求输出，可减少页面实际发起的网络请求数。请求合并需要 Web 资源加载器配合。

增量计算的输出是一个固定格式的JSON，描述了 Patch Diff 结果时所遵循的规则，如下图：



patch规则

patch规则

例如：

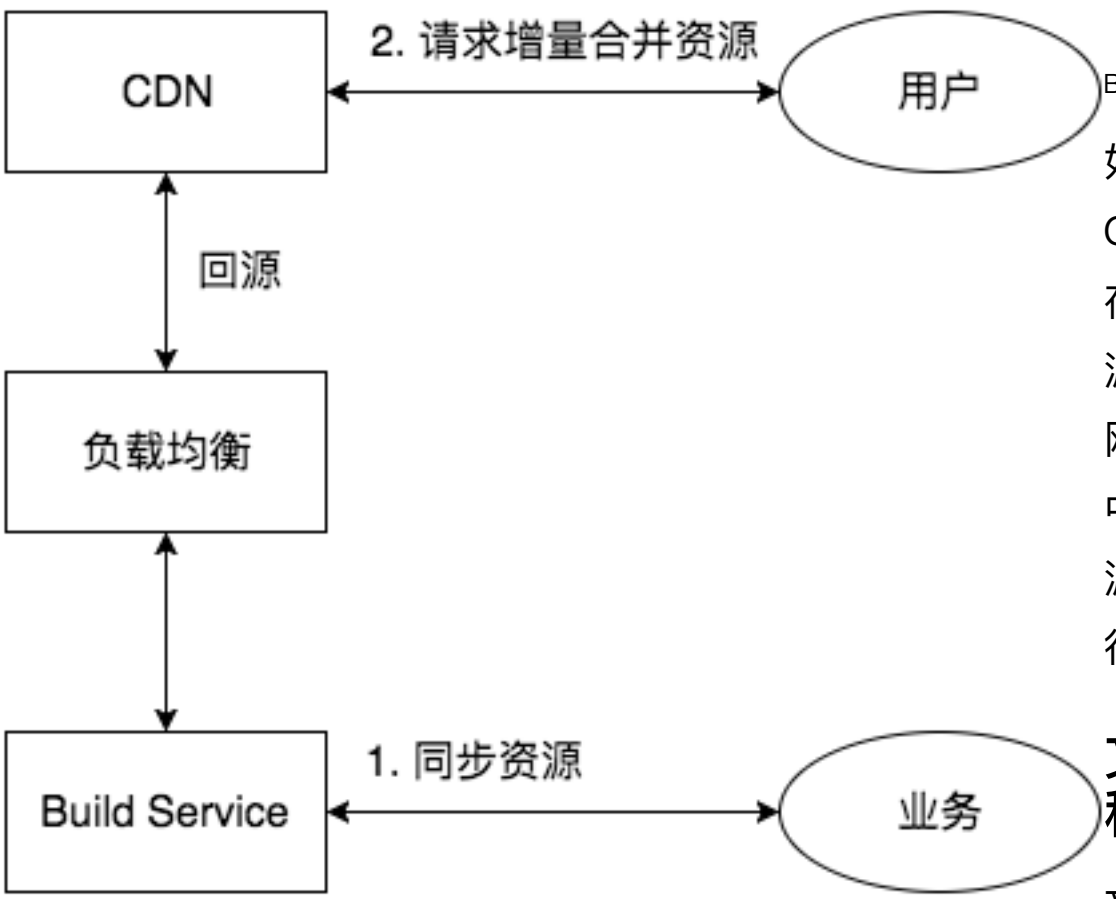
```
[
  'cmp/util.js',     // 资源文件名
  [0, 33],           // 需要保留的字符位置
  'mn',              // 需要拼接的字符
  [34, 10]                  // 需要保留的字符位置
]
```

以上数据结构表示原文件从第 0 个位置开始保留 33 个字符，连接 `mn` ，从第 34 个位置开始保留10个字符。

Patch Diff 结果就是利用增量更新的结果，结合原文件，将文件恢复至最新文件的过程。

Build Service 工作流程

Build Service 是美团平台的静态资源托管方案，提供静态资源部署、处理和分发能力，对接 CDN。



Build Service 基本链路

Build Service 基本链路

如图，用户请求达到 CDN 如果没有命中缓存，会一路回源至源站，源站检索并处理资源，经网关输出给 CDN。本文中提到的增量计算属于资源处理任务，由源站执行。

文本增量计算的工程选择

文本增量计算最初基于编辑距离原理实现，时间复杂度 O(N^2)，与文本长度正相关，实际应用时性能较差。Build Service 选择 **Myers 增量算法**🔗🔗 **(http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.6927)** ，有效降低单次增量计算的时间消耗。其时间复杂度由 O(N^2) 改变为 O(ND)，与文本长度、差异长度正相关。Web 业务迭代频率高、单次迭代差异小、D 接近常数，使用 Myers 增量算法时间复杂度可接近 O(N)。

初步效果

根据扫码付的统计结果，增量更新相比全量请求，传输数据可减少多至99%，合并请求平均可减少请求数95%。

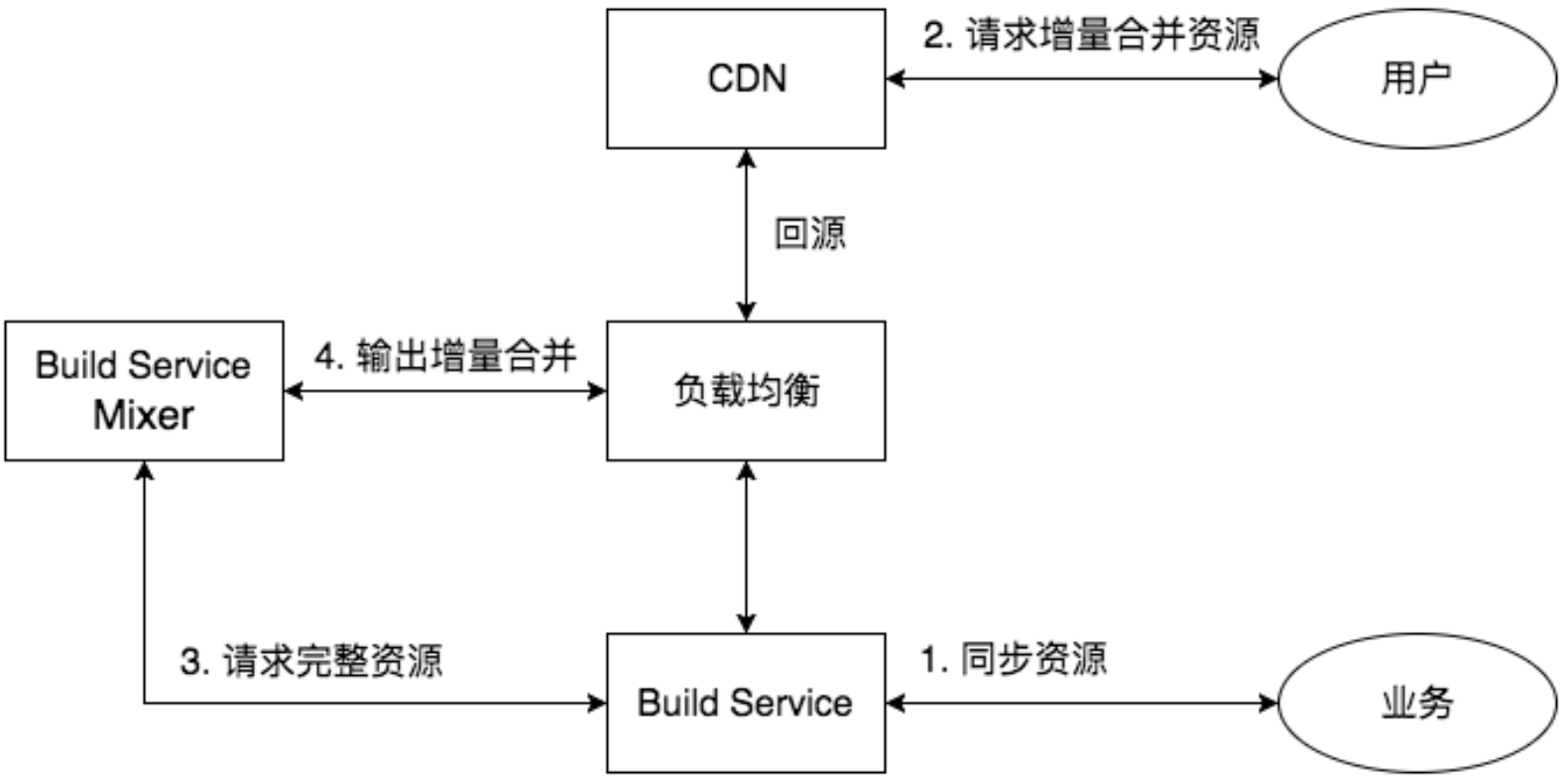
业务增长与计算瓶颈

随着业务的增长，PV 很快就在2017年4月份达到了百万级。扫码付业务采用细粒度模块化的设计，业务不断迭代，文件数越来越多，单次合并请求的文件数超过 30个。需要进行增量计算的版本组合也越来越多，跨越多个版本的增量计算开始出现，增量计算耗时增加，Build Service 遇到了计算能力的瓶颈。我们发现3s超时时间条件下，合并请求的失败率超过50%，于是着手开始优化。

Build Service 优化策略

服务拆分与隔离

Build Service 最初直接通过公共集群提供文本增量计算服务。公共集群同时还承载着其他计算任务，如文件压缩、引用计算等。增量计算与其他任务相比，计算规模差异巨大，消耗了集群大多数算力，导致其他计算任务延迟大幅升高。为了避免公共集群不可用对公司其他业务产生影响，Build Service 紧急拆分上线 Build Service Mixer 服务（以下简称 Mixer 服务），将请求合并和增量计算独立出来单独搭设集群，实现业务隔离。



Build Service Mixer

Mixer 服务上线后，隔离了增量计算对其他业务的影响，争取了一些时间优化整个方案。

持久化计算缓存

合并请求的各个资源文件是互相独立的。Mixer 收到一次请求，会分别缓存每个资源文件的计算任务输出。不同的资源合并请求可以复用结果片段，减少不必要的计算。上线后，Mixer 服务的计算能力显著增强，日可用性一度达到100%，计算成功的增量片段再输出的时间消耗稳定在50毫秒以内。

超时自动重启机制

Myers 增量算法大多数情况下性能提升显著，但是当文本差异较大时，计算耗时会显著增加。最不理想的情况下时间复杂度会退化到O(N^2)。Mixer 服务使用 Node 开发，计算增量与输出资源在一个进程，为了避免计算任务阻塞请求响应，我们将计算改为了进程内异步。

有时业务会上线差异较大的增量片段，在一个很短的时间窗口内，许多相似的用户请求会同时分摊给所有 Mixer 进程，宿主机的所有 CPU 核心被占用处理同一个慢的增量计算，导致 Mixer 服务输出能力下降，请求积压。为了临时解决这个现象，我们采用了简

单粗暴的自动重启，如果计算超时判定为慢计算，服务自杀由 PM2 重新拉起。服务重启后慢计算立即失败，用户侧降级到单资源请求，Mixer 有概率可以分配到快计算。时间窗口通过后不再出现慢计算时，Mixer 服务算力恢复。

这个机制一定程度上缓解了我们的计算瓶颈，但是没有完全解决问题。

ThunderJS 优化策略

限制合并请求文件数

实际业务使用中，我们发现由于没有对合并请求的文件数做限制，一次合并请求会合并过多的请求，特别是在扫码付这个项目中，导致一次请求的计算量过大，造成比较严重的超时问题。

正逢 Mixer 瓶颈阶段，为了降低 Mixer 的输出压力，我们需要使一次合并请求能够使用更少内存更快地完成。综合考虑后，我们降低了单次请求合并的资源数量上限，从最初的不设限改为限制最多 10个资源，这样由原本一次请求30个文件的增量结果，改成并发3个请求，每次请求10个文件，同时 Mixer 配合参数调优，一定程度上缓解了超时问题。

业务降级机制

合并请求失败后的单文件加载缓存

正如前文所说，在实际情况中，Mixer 计算服务会不可避免的遇到超时的问题，为了避免超时后导致无法加载相应的静态资源，我们有针对性的设计了降级机制。

在最初的 ThunderJS 中，如果遇到超时，会重新使用 `createElement` 方式将合并请求中的资源单独加载（直接请求文件，而不是请求文件两个版本的增量结果）。但是在实际业务中，我们注意到，如果能将单独加载的文件也做缓存，那在超时比较严重的时段，能有效避免老用户重复进行请求，因此我们将 `createElement` 方式换成 XHR ，将请求响应的文件内容存入 LocalStorage，实现了在降级机制下增强缓存的效果。

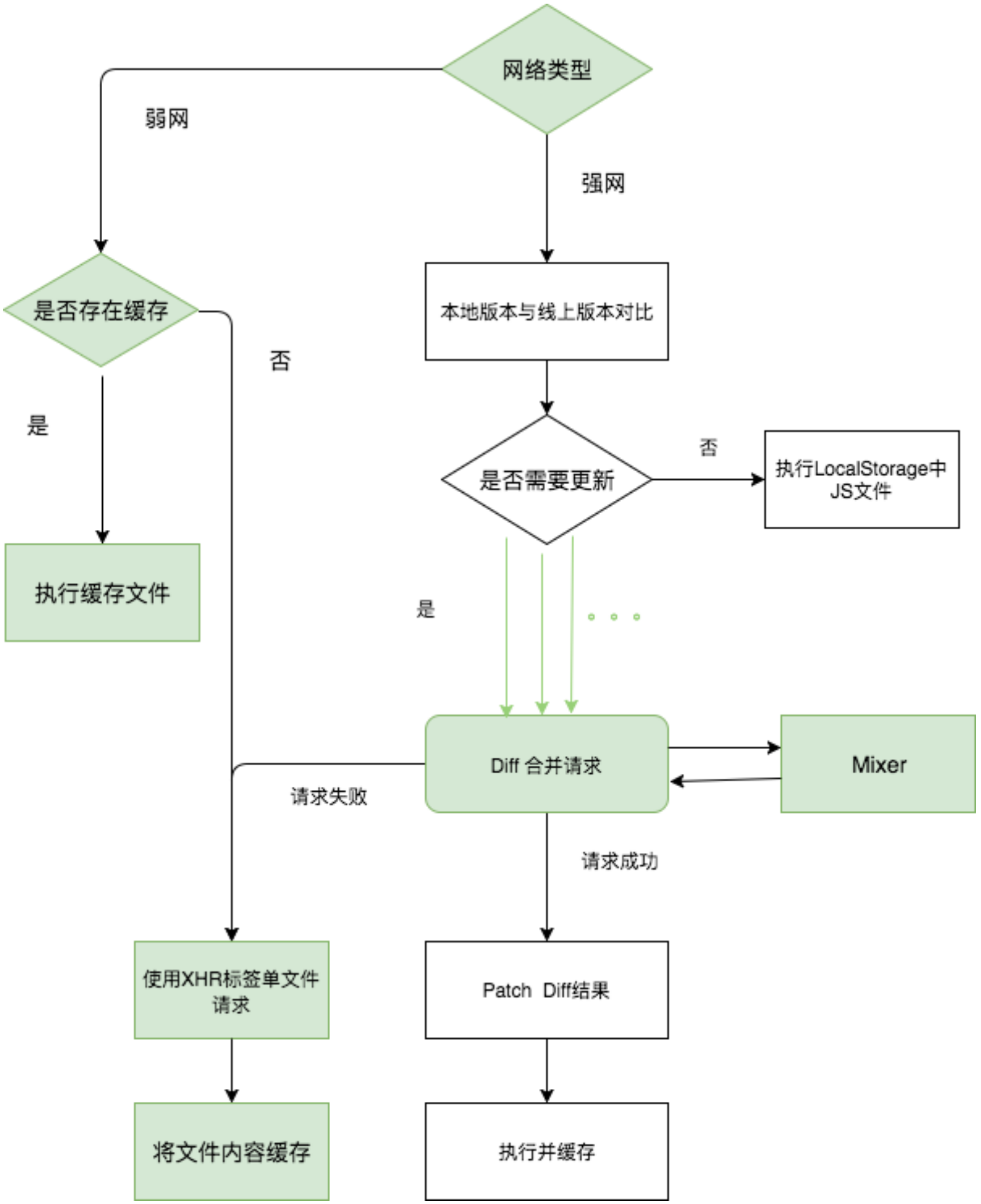
弱网优先使用缓存文件

不管请求有多快，终究还是需要发起网络请求，最好的方式就是不需要网络请求即可使用，我们将网络状况分为 **WiFi**、**4G**、**3G**、**2G**、**unknown** ，其中 2G和unknown 被我们认为是 **弱网**，大概占比在10.35%，对于这部分用户，我们选择优先执行缓存中文件，没有相关文件则进行单文件请求。

优先执行缓存的出发点在于弱网下加载文件成本较高，我们需要优先保证支付流程的完善，即使这样无法给用户带来最新的用户体验。

完善降级机制后的流程图如下所示：

完善降级服务



完善降级服务

实践证明，降级机制起到了非常大的作用，在前期超时问题比较严重的情况下（超时率50%+），降级机制甚至承担了主要角色，在后期，降级机制的存在也是根本解决计算瓶颈的方案之所以能实施的前提。

计算瓶颈的根本解决方案

扫码付业务持续增长，增量计算服务的瓶颈依然存在。根据公司的基础监控服务的数据，Mixer 服务周期出现的请求积压越来越频繁，`CLOSE_WAIT` 数会快速增长。`CLOSE_WAIT` 是一种连接状态，在服务端响应未完成的请求前，连接被请求方

关闭时可能出现。这个指标的快速增长意味着大量的请求不能在超时区间内处理，直接表征 Mixer 服务算力不足。

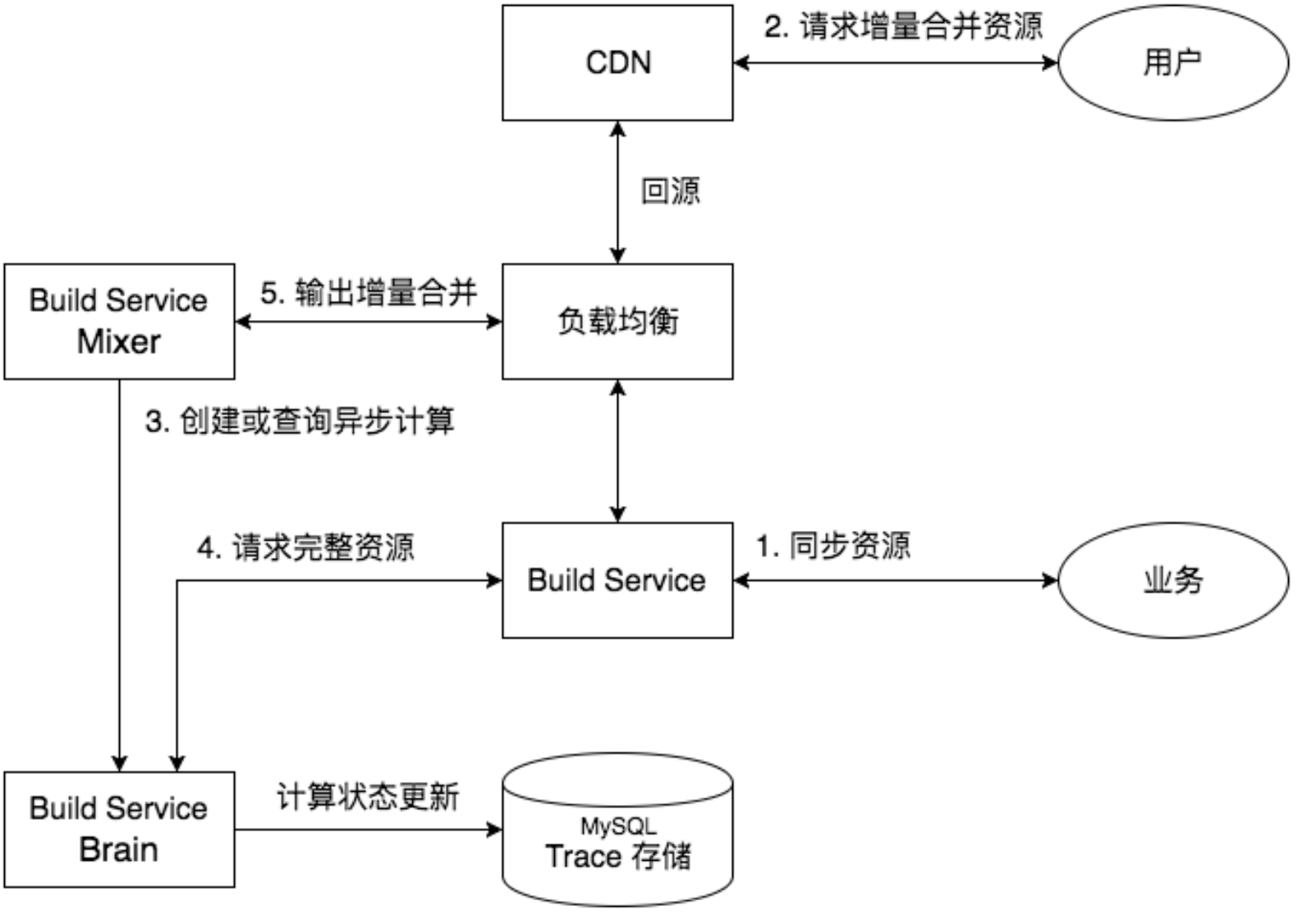
前文中我们讨论过增量计算时间复杂度高，即便使用 Myers 增量算法，也不会快到没有耗时。除非文本增量计算有重大理论突破，否则静态资源的文本增量计算的固有耗时是不可能降低的。

Mixer 增加超时重启机制后，提高快计算被分配到的概率，但并未达到 100%。更糟糕的是，计算完成后写入本地持久化缓存的过程是异步的，服务遇到慢计算后重启，上一个写入可能并未完成。这样下次请求到达后，缓存不可用，快计算也需要重新计算。由于 Mixer 服务设计为各节点完全等价，无论扩容多少个节点，当业务请求窗口到达时，慢计算都会出现在所有节点。

计算结果不可在节点间复用、慢计算导致服务反复重启、计算结果不能确保持久化缓存，浪费了整个服务的算力。

Build Serivce 异步计算服务

我们重新设计了静态资源服务的架构，将计算服务（Build Service Brain，以下简称 Brain 服务）和分发服务分离开来。



Build Service Brain

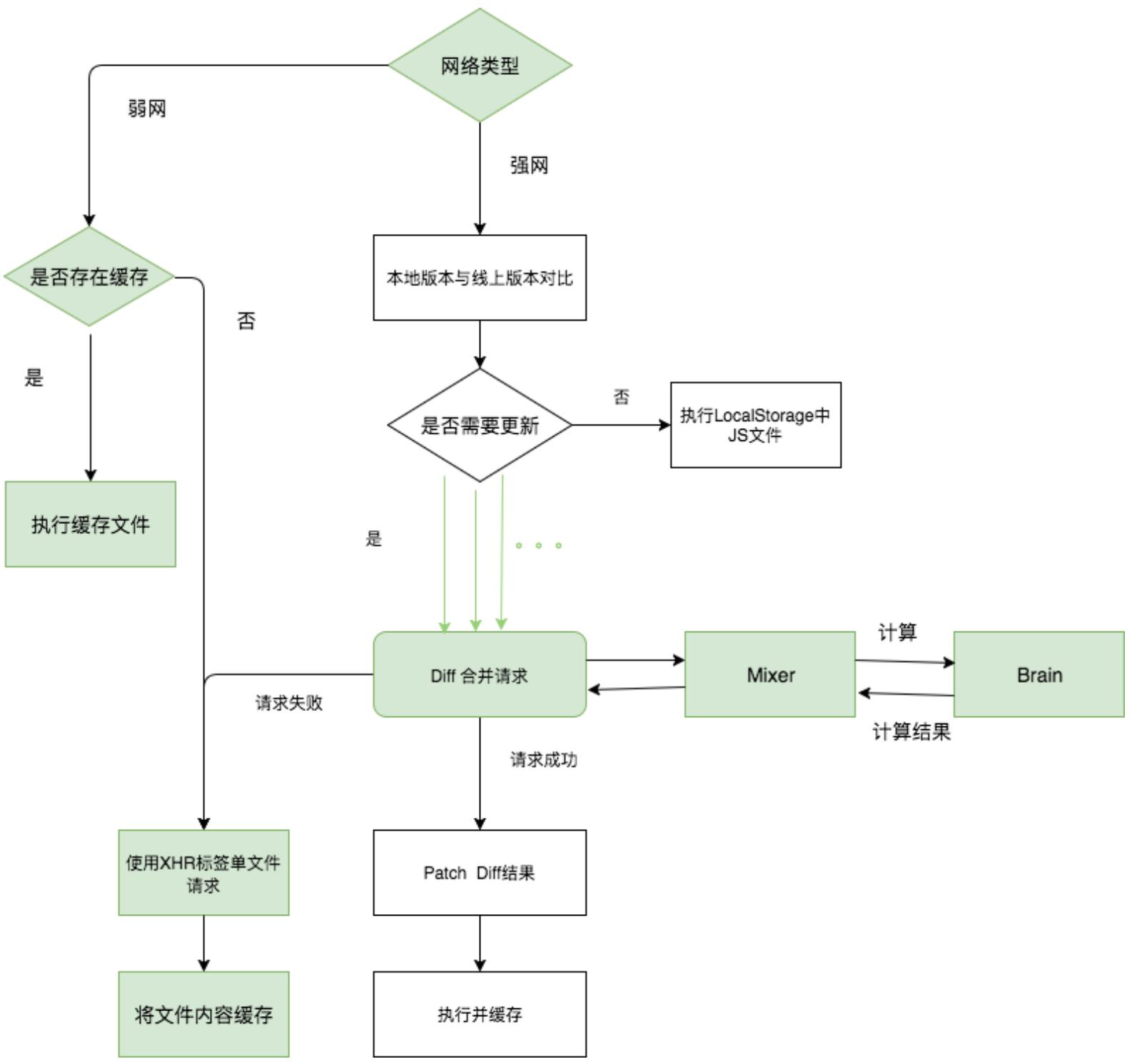
Build Service Brain

Brain 服务使用 MySQL 存储计算的唯一标识（我们称为 trace 信息）。每个 trace 可以唯一指代一个计算，每个计算仅允许一个节点执行。当计算任务到达 Brain 服务的随机一个节点后，Brain 服务首先检查是否已经被分配，如果已经分配立即返回状态信息；如果计算任务完成直接路由到对应节点输出结果。这个设计使 Brain 服务成为可水平扩展算力的分布式计算和存储服务。计算任务本身改为另起进程，完全避免计算任务和网络服务进程抢占资源的问题。只要部署节点数达到一定数量，集群就可以避免整体被某个慢计算挂起。

Brain 服务上线后，Mixer 服务不再负责计算，可用性提升至稳定 99.99%。整体后端响应时间（TP90）从 5800 ms 提升至 90 ms。Brain 服务在一个月时间内完成了 10W+ 计算。根据业务的统计数据，Diff 合并请求成功率提升至少 50%。

线上发版前的预热方案

经过以上 ThunderJS 和 BuildService 的优化 ，我们的超时率降到 3%，收益非常显著。此时的流程是：



解决超时

解决超时

当一个计算任务的固有耗时无法减少时，可以通过提前计算来避免用时压力。以前的 Build Service 架构不能支持我们任意预热，但是新架构的设计是允许预热的。所以我们进一步实施了预热方案。

实施预热首先要考虑的点就是哪两个版本之间的预热，在 ThunderJS 的设计中，文件版本号取自 Git 的 CommitId，每次提交后，即使文件内容没有变化版本号也会递进，导致需要进行不必要的合并请求。这一点在之前我们优化超时问题时，被认为是 ThunderJS 的一个待优化点，而在预热阶段，CommitId 比文件内容的 Hash 值更有价值，通过追踪 Git 提交历史，我们可以很容易的找到所有文件的线上版本；如果使用文件内容的 Hash 值作为版本，不能描述版本先后关系，无法明确找到文件增量计算的前后版本，预热也就无从谈起了。

通过我们埋点计算，线上发版之前预热 5 个版本（分别计算最近 5 个版本到最新版本的增量）能将超时率降到1.5%，预热 10 个版本能将超时率降至 1.1%。

理论上，预热更多版本可以进一步降低超时率，预热所有版本可以使超时消失，但是预热所需时间也会大幅增加。在实际情况中，我们需要在预热效果和预热成本之间折衷选择。

总结

项目发展至今，ThunderJS 增量更新方案在扫码付项目中取得了非常好的收益。

缓存命中率	缓存命中文件数	增量请求占比	增量请求次数	缓存利用率	单文件全量大小
31%	2925W/天	36.7%	2083W/天	89.12%	1.77Kb

ThunderJS收益

ThunderJS收益

扫码付项目的所有请求中，有90%来自于移动网络，10%来自于 WiFi，通过缓存平均每天节约流量 49.37GB，通过增量更新平均每天节约流量 33.41GB。对访问量大，网络环境要求严苛的 C 端产品来说，节约的流量和网络请求时间消耗都是我们为用户带来的价值。

招聘信息

美团金融智能支付大前端研发团队正在招聘资深前端工程师，前端技术专家，集团重点业务（真心重点，不吹），机会多，挑战大，来来来，我们好好聊聊，即使暂时不考虑机会，也可以和我们聊聊妹子，聊聊比特币，万一哪天你看机会了呢？Email: sunhui04#meituan.com。