* [首页](https://km.woa.com/?kmref=km_header)
* [发现](https://km.woa.com/discovery?kmref=km_header)
* [悦读](https://km.woa.com/read?kmref=km_header)
* [乐问](https://km.woa.com/q?kmref=km_header)
* [直播](https://tencent.lexiangla.com/lives?company_from=tencent&kmref=km_header)
* [应用](javascript:void(0))
* [我的K吧](javascript:void(0))
* [[](https://km.woa.com/user/rickyrqzhao)](https://km.woa.com/user/rickyrqzhao)

窗体顶端



窗体底端

[](https://km.woa.com/group/edu)[在线教育部](https://km.woa.com/group/29048)

* [首页](https://km.woa.com/group/29048)
* [团队讨论](https://km.woa.com/group/29048/topics)
* [团队文章](https://km.woa.com/group/29048/articles)
* [团队文档](https://km.woa.com/group/29048/docs)
* [团队活动](https://km.woa.com/group/29048/events)
* [团队投票](https://km.woa.com/group/29048/surveys)
* [团队相册](https://km.woa.com/group/29048/photos/albums)
* [团队视频](https://km.woa.com/group/29048/videos)

目录

一、导语

二、Webpack5 持久化缓存初体验

1. 持久化缓存原理

2. 持久化缓存本地测试

3. 持久化缓存CI测试

4. 缓存初体验总结

三、持久化缓存的版本控制

1. 缓存失效场景

2. 基于 Master Branch 的缓存版本

3. 版本控制总结

四、CI环境缓存持久化方案探索

1. 对象存储 — 腾讯云COS

2. Docker Layer Caching

3. Docker Image 镜像文件

4. Docker Volumes 数据卷

五、CI环境缓存持久化方案选择

1. 方案选型

2. 构建机优化

六、优化数据展示

1. OCI 构建数据

2. 本地开发数据

3. 优化数据总结

七、进一步的优化和思考

Webpack5 持久化缓存实践—CI提速50%!

* [mickyzeng](https://km.woa.com/user/mickyzeng)

* 2021年09月02日 16:27

* 浏览(1165)

* [收藏(82)](javascript:void(0);)

* [评论(10)](javascript:void(0);)

* [分享](javascript:void(0);)

| 导语 Webpack 5 发布后，新特性 “持久化缓存” 原生支持了打包全过程的缓存，大幅提升了构建效率。本文将基于Webpack5的持久化缓存这个新特性，分享前端工程化中构建优化的一些实践和思考

**一、导语**

Webpack 5 发布后，企鹅辅导团队就已经尝试接入，早在之前做过一次[探索](https://km.woa.com/group/29048/articles/show/443356" \t "_blank)，但在工程中落地还需要对Webpack 5相关特性进行研究。其中和构建相关的更新中，有令人眼前一亮的特性：**持久化缓存**，从此无需各类plugin实现的cache，**Webpack原生支持了打包全过程的缓存，能大幅提升构建效率。**

团队也已经完成多个项目的接入，效果显著，本文将基于Webpack5的持久化缓存这个新特性，分享前端工程化中构建优化的一些实践和思考

**二、Webpack5 持久化缓存初体验**

团队在Webpack4 阶段就已经做了比较完整的缓存优化提高构建效率，参考[企鹅辅导极致优化 OCI 构建](https://km.woa.com/group/29048/articles/show/443586" \t "_blank)

“企鹅辅导团队构建工具 [IMFLOW](http://imflow.oa.com/) 做了一系列的优化，如利用缓存，将 babel、压缩 teser、CSS 编译等构建结果统一缓存起来，存储在缓存目录下，下次编译可利用上次的编译缓存， 进而提升构建速度。而对 CI 的构建优化，考虑到容器的无状态，将缓存存储在了云上，在构建前拉取云上的缓存结果，在构建后上传新生成的构建缓存。“

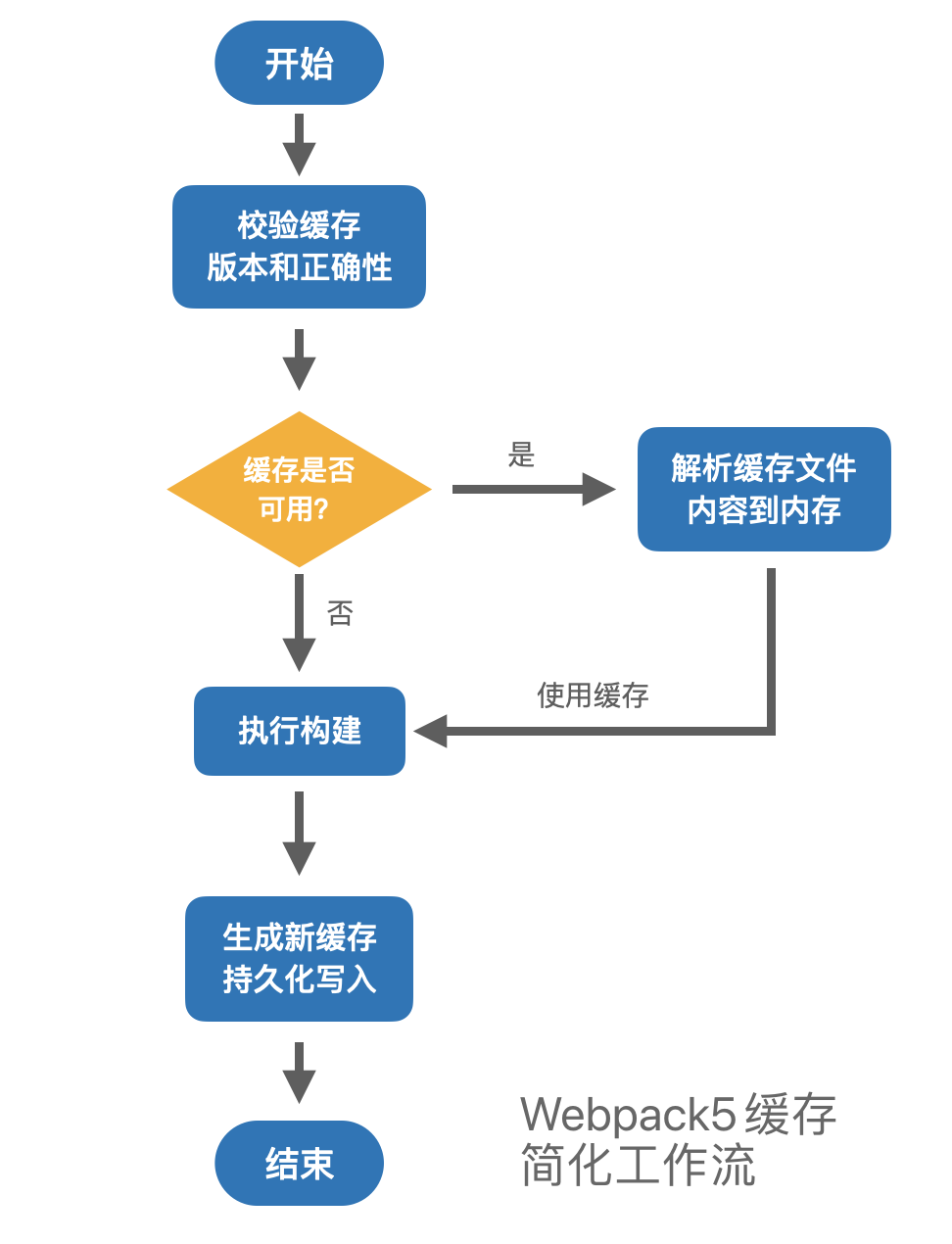
为了追求极致的构建效率，我们在IMFLOW基础上尝试接入Webpack5，看看是否能为项目构建速度带来新的突破～

**1. 持久化缓存原理**

查阅[官方文档](https://github.com/webpack/changelog-v5/blob/master/guides/persistent-caching.md#internal-workflow)，得知Webpack5持久化缓存设计的原理：

**将编译过程中的依赖关系和编译产物压缩保存到本地文件，下次构建的时候对缓存文件进行解压缩复用**

如下图是缓存简化工作流

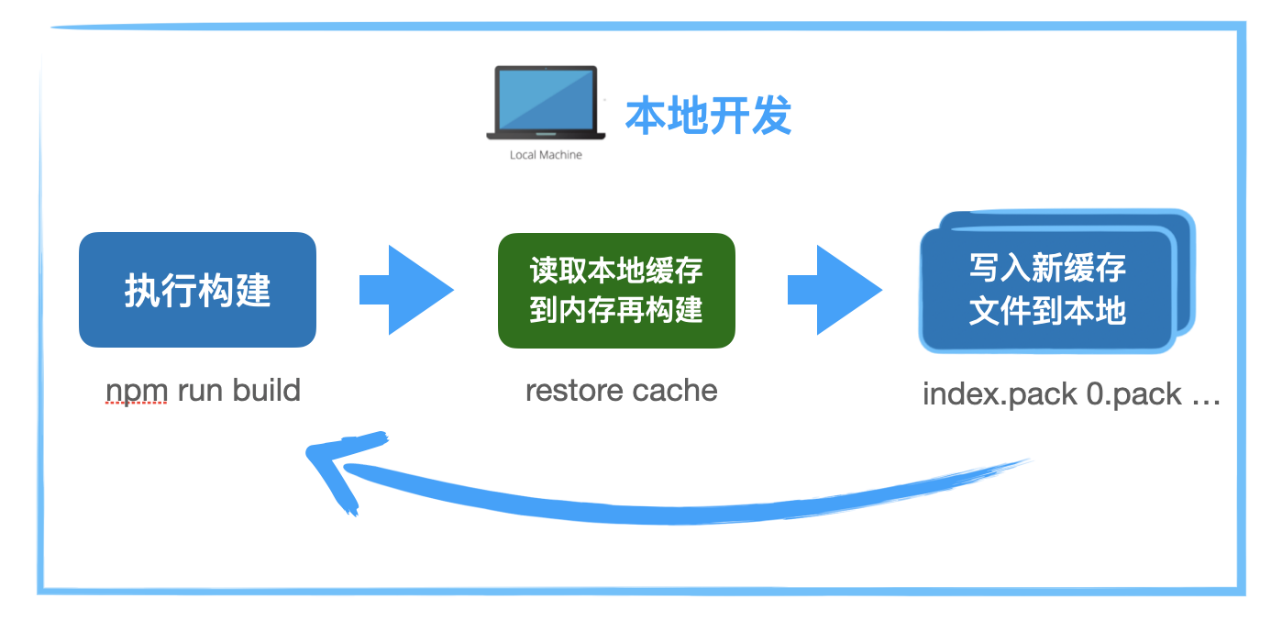


**了解了持久化缓存的工作原理，我们开始进行本地环境和CI环境的测试**

**2. 持久化缓存本地测试**

项目完成Webpack5升级后([升级指南](https://webpack.js.org/migrate/5/))，[配置cache选项](https://webpack.js.org/configuration/cache" \t "_blank)启用持久化缓存，验证缓存的效果

经过首次编译缓存生成后，二次编译时就可以感受到编译速度的大幅提升，如下图所示



1. 每次构建，会先读取本地缓存文件，解析缓存内容到内存再执行构建，如果**缓存命中的话构建速度将极大提升**
2. 构建完成后，Webpack会**刷新缓存文件内容**，再重新写入到缓存目录，这一步是为了保持缓存内容是最新的代码编译内容

下面在本地环境测试，看看Webpack5的优势体现在哪

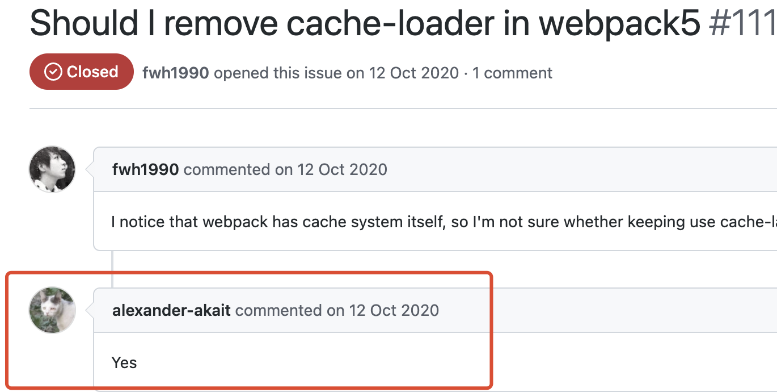
* Webpack4使用 cache-loader + babel-loader cache + hard-source-webpack-plugin + terser cache + css cache 的缓存方案
* Webpack5使用type为filesystem的持久化缓存方案

Webpack4和Webpack5对比数据如下 (npm run build)

| **构建时长** | **首次构建** | **二次构建** | **三次构建** |
| --- | --- | --- | --- |
| Webpack4 插件缓存 | 4.32m | 1.30m | 1.23m |
| Webpack5 原生缓存 | 3.75m | 47.95s | 29.47s |
| 对比 | 提升13% | 提升38.4% | 提升60% |

可以看到**Webpack4插件的缓存提速效果，远比不上Webpack5**，并且**Webpack4需要学习不同插件进行复杂的配置，Webpack5只要一行配置！**

同时，升级Webpack5后这类插件都可移除，简化项目配置。如下图是官方给出的对于 cache-loader 升级后是否需要移除的回复

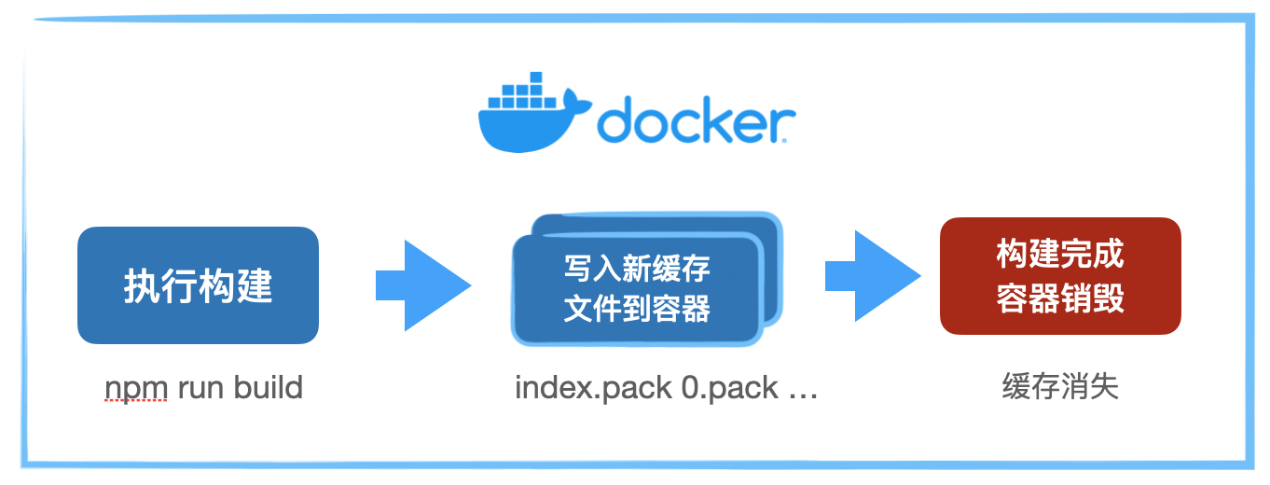
****

小结：Webpack5比Webpack4缓存方案带来**更大的速度提升**，并且能**简化项目配置**，减少理解成本。

**3. 持久化缓存CI测试**

本地测试通过后，将配置好的代码提交到仓库，触发构建后发现构建速度和升级前几乎没有变化

思考这个问题，发现在OCI环境中流程如下

****

1. 每次构建都是一次新的环境，没有上次编译生成的缓存，所以不存在读取缓存的步骤
2. 构建完成后Webpack会生成新的缓存，但缓存没有被保存在机器上，构建任务完成后缓存就会消失

小结：CI过程中没有使用上次缓存构建，并且新生成的缓存文件没有被持久化，缓存无法在CI环境产生效果

**4. 缓存初体验总结**

1. 持久化缓存的重点是需要**对缓存产物进行持续地使用刷新**，才能提高缓存命中率加速构建
2. 本地环境测试缓存效果符合预期，**本地开发效率得到明显提升**
3. CI环境测试缓存无效，无法提升构建速度，因为缓存产物**没有被正确引用和持久化**
4. **需要探索如何对缓存文件进行持久化，让CI也能正常使用缓存，提升构建效率**

**下文我们将探索在CI环境下，我们可以从什么角度对产物进行持久化操作**

**三、持久化缓存的版本控制**

在探索持久化方案之前，需要考虑缓存的版本控制策略，这将决定**如何保证缓存有效性，何时进行缓存持久化**

**1. 缓存失效场景**

Webpack 可以通过 构建依赖（[Build dependencies](https://webpack.docschina.org/configuration/cache/#cachebuilddependencies)）和 缓存版本（[Version](https://webpack.docschina.org/configuration/cache/#cacheversion)）控制缓存的失效时机

除此之外以下操作均会[使缓存失效](https://github.com/webpack/changelog-v5/blob/master/guides/persistent-caching.md)

* 当 npm 升级 loader 或 plugin 时
* 当更改配置及配置中读取的文件时
* 当 npm 升级配置中使用的 dependencies 时
* 当不同命令行参数传递给 build 脚本时
* 当有自定义构建脚本并进行更改时

所以开发者需要**认真评估可能影响缓存缓存的相关文件**，比如项目的webpack.config.js、自定义配置文件等，防止在相关依赖变更时，使用的还是旧缓存，造成不可预料的构建结果

同时**Webpack5 支持部分缓存生效，在构建依赖和版本未变更的情况下，业务代码的改动只会使部分缓存失效**

**合理使用snapshot配置（Contenthash）能使缓存在CI环境的持久化更有效，详见**[**Snapshot types**](https://github.com/webpack/changelog-v5/blob/master/guides/persistent-caching.md#snapshot-types)

**小结：**

1. 开发者需要**理解缓存失效的原因和场景，正确配置依赖**，避免出现不合理的缓存构建结果
2. **Webpack5 支持部分缓存生效，**这表示只需要保证依赖文件不变更即不同程度可利用缓存

**2. 基于 Master Branch 的缓存版本**

企鹅辅导的项目当前的开发流程为**feature 分支开发，发布完毕合并 master** ，如果master有代码变更，其他分支提交时一定要先合并master 分支

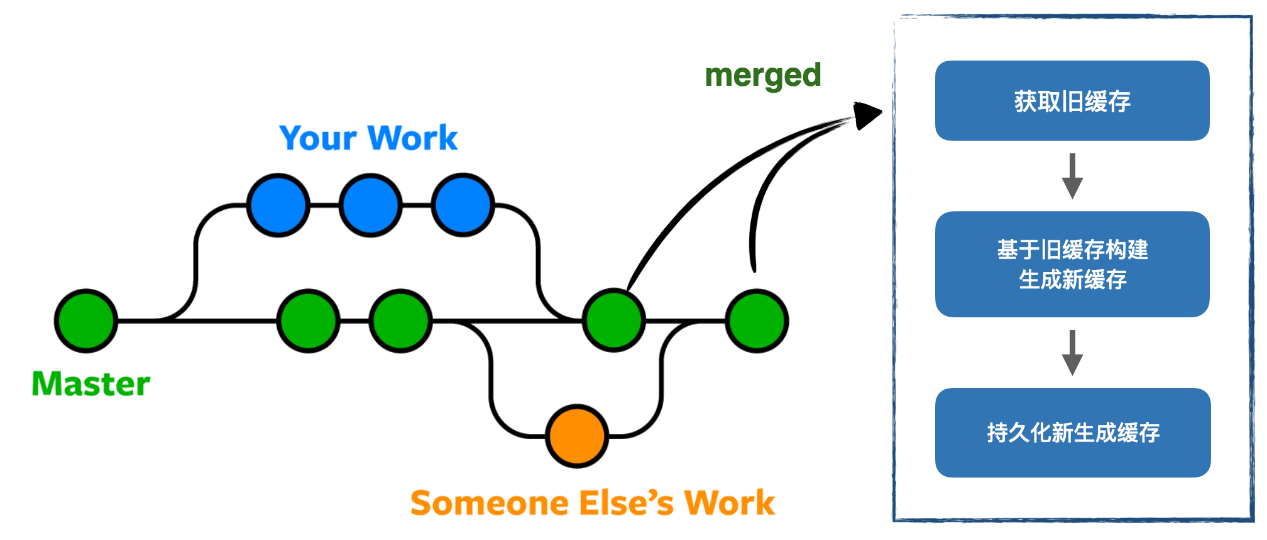
**基于这个流程，如果每次构建都进行缓存更新的话，需要思考以下问题**

1. 以分支为单位构建的缓存，共用会造成缓存混乱，大幅度**降低缓存命中率（依赖文件可能变更）**
2. 以分支为单位构建的缓存，单独使用会增加空间成本，**失去了构建机中的共用缓存优势，每个分支提交都需要经历冷启动**
3. **在CI环境，频繁地保存Webpack缓存都会增加时间和空间成本**

**我们可以从以下两点切入：**

1. 考虑**以公共分支作为基准分支构建缓存**，其他分支基于公共分支缓存进行构建
2. 控制好公共分支的缓存构建时间，**保证缓存的最大新鲜度**

基于这个结构方案设计如下：



1. 在**merge 事件触发时**，**刷新缓存内容并进行持久化，**这样能保证缓存的最大新鲜度
2. 各分支开发的时候，使用同一份master生成的缓存，这样就实现了 **最少的缓存生成次数，最大化的缓存利用 ！**

**3. 版本控制总结**

1. 开发者需要**理解缓存失效规则，合理控制缓存版本**防止出现不预期结果
2. 推荐使用基于master分支的缓存，能**最低频次更新，最大化缓存利用**
3. 开发分支在缓存**依赖未变更时，能较好利用缓存**，变更后将无缓存构建

理解了缓存失效规则，确认了缓存版本方案后，我们进一步研究如何在CI实现缓存的持久化

**四、CI环境缓存持久化方案探索**

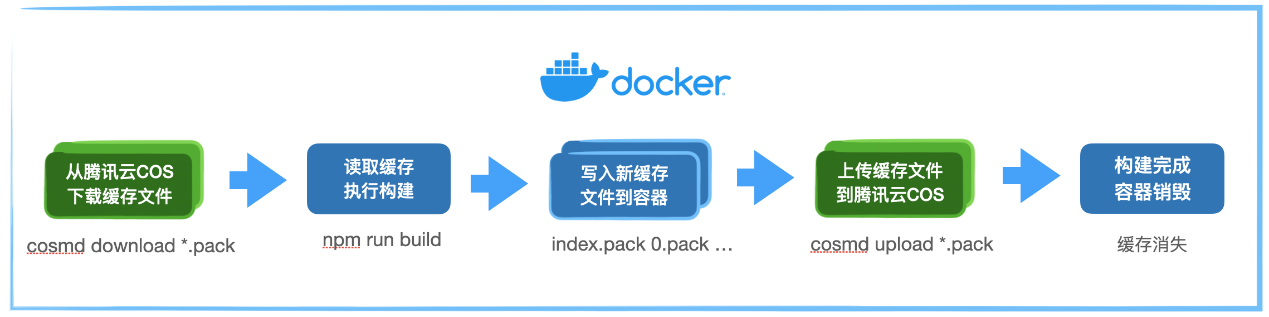
那如何持久化缓存呢？现在前端工程化大都使用了容器化虚拟化的技术，每一次构建都是全新的环境，如果需要持久化数据，可以思考以下几个方向

1. 使用 对象存储等远端存储方案，构建前先拉取缓存，构建完成再上传缓存
2. 使用[Docker Layer Caching](https://medium.com/swlh/docker-caching-introduction-to-docker-layers-84f20c48060a) ，将Webpack缓存保存在Docker缓存层，再从缓存层取出，达到数据持久化
3. 使用 [Docker Image](https://docs.docker.com/engine/reference/commandline/build/) ，将Webpack缓存保存为 Docker 镜像，使用时从镜像中取出，同时镜像缓存也能保证效率
4. 使用 [Docker Volumes](https://docs.docker.com/storage/volumes/) 数据卷，能达到和本地接近的效果，但前提是需要CI支持

下面我们探索这几种方式，看实际使用会遇到什么问题

**1. 对象存储 — 腾讯云COS**

考虑使用腾讯云COS来存储构建缓存，并在构建过程做好缓存管理，CI调整后流程如下



1. 从腾讯云COS拉取缓存
2. 基于缓存执行构建
3. 构建完成缓存被更新，上传最新的缓存到腾讯云COS

可以看到这次**成功实现了缓存的优化，提高了构建效率**，但是这个方案有一个不可忽视的的问题：**高成本**

****

如图Cache的文件大小为223M，上传时间为28s，下载速度为7MB/s， 按照10MB/s计算，几百兆的文件也需要耗时几十秒，网络带宽限制的问题很明显

除了网络导致上传下载速度慢 影响构建时间，还有一点要考虑的是 流量费用也会增多，方案缺点明显

**小结：**

1. 每次构建都要进行COS上传下载操作，**耗时较长，影响整体构建时间**
2. 当项目越来越大，文件体积增大，构建将越慢，**项目规模和构建效率成反比**
3. COS操作对网络依赖严重，**构建速度无法得到一致保证**

**单纯远端存储缓存的方案并不适合大文件频繁操作，方案不可取，需要继续探索**

**2. Docker Layer Caching**

方案设计之前我们需要了解 Docker Layer Caching，可以看这篇 [Introduction to Docker Layers](https://medium.com/swlh/docker-caching-introduction-to-docker-layers-84f20c48060a) 或 [掌握Docker Layer Caching](https://www.cnblogs.com/JulianHuang/p/12953129.html)

一句话描述就是：**Dockerfile和相关文件未更改，重建(rebuild)时可以重用本地镜像缓存中的某些现有层，减少重建时间**

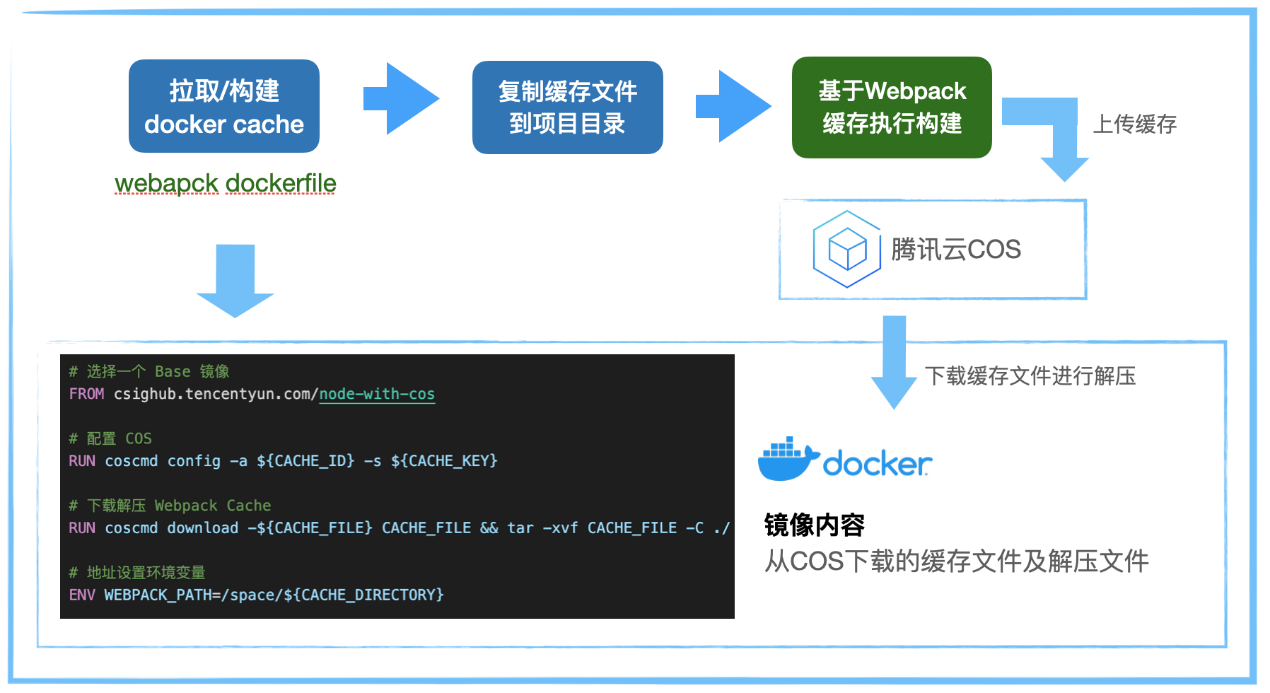
目前很多主流的CI工具都提供了Docker Layer Caching 能力，如[Circle CI](https://circleci.com/docs/2.0/docker-layer-caching/" \t "_blank)、[Semaphore CI](https://docs.semaphoreci.com/ci-cd-environment/docker-layer-caching/" \t "_blank)等

当然优秀的OCI也提供了～[文档查看](http://doc.orange-ci.woa.com/internal-steps/docker/cache.html" \l "docker-%E7%BC%93%E5%AD%98" \t "_blank)，下面会基于OCI的Docker Cache能力展开

了解清楚Docker Cache的作用和使用方法后，基于以下几点考虑

* 缓存的内容较大，重复下载成本较高
* 缓存的内容改动不频繁，改动后可以通过版本文件刷新缓存
* 缓存需要提供给后续步骤使用

**考虑使用Docker Cache承载Webpack Cache，**基于Docker Cache的特点，方案设计如下



1. 构建Docker Cache，这里Dockerfile 下载存储在COS的Webpack缓存并解压处理
2. 构建前 复制Docker Cache保存的缓存，基于缓存执行构建操作
3. 构建完成后，将最新缓存上传到 COS

实践证明OCI Docker Cache 结合 COS的方案能**成功实现缓存的复用**

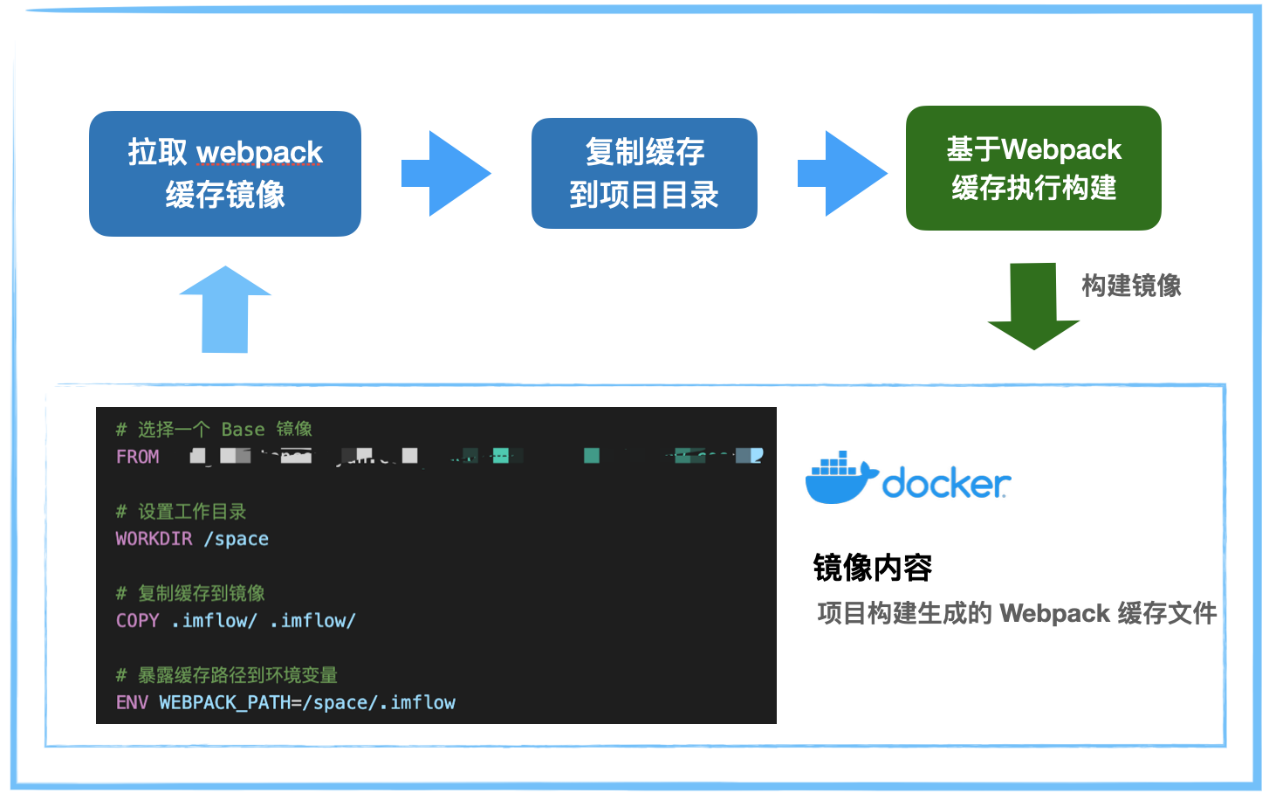
****

**小结：**

1. Docker Layer Caching 结合腾讯云 COS **实现了缓存文件的持久化，提升了构建效率**
2. 生成Docker Cache的过程，还依赖了远端存储的参与，**步骤上较复杂，方案还有优化空间**

**3. Docker Image 镜像文件**

Docker镜像承载缓存文件从技术上来说肯定是没得问题的，技术方案如下



1. 初始的时候需要构建镜像，Dockerfile 很简单，直接复制缓存目录，再Build镜像就可以了
2. 在构建之前**使用OCI的 基于镜像的stage**，将镜像内的缓存文件复制出来，就能愉快地使用缓存了

**实践效果如下**

镜像拉取在有缓存情况下在3s内完成，使用镜像内的缓存成功提升了构建速度



**小结：**

1. Docker Image 方案简单粗暴，**实现了缓存文件的持久化，提升构建效率**，**流程对比Docker Cache也清晰简单很多**
2. Docker Image 的使用在OCI内部已经有缓存实现，拉取速度快；镜像构建的速度也较快（复制操作），**整体成本较低**
3. Docker Image **需要对镜像仓库内镜像版本进行管理**，这里可以通过规范或工具进行操作

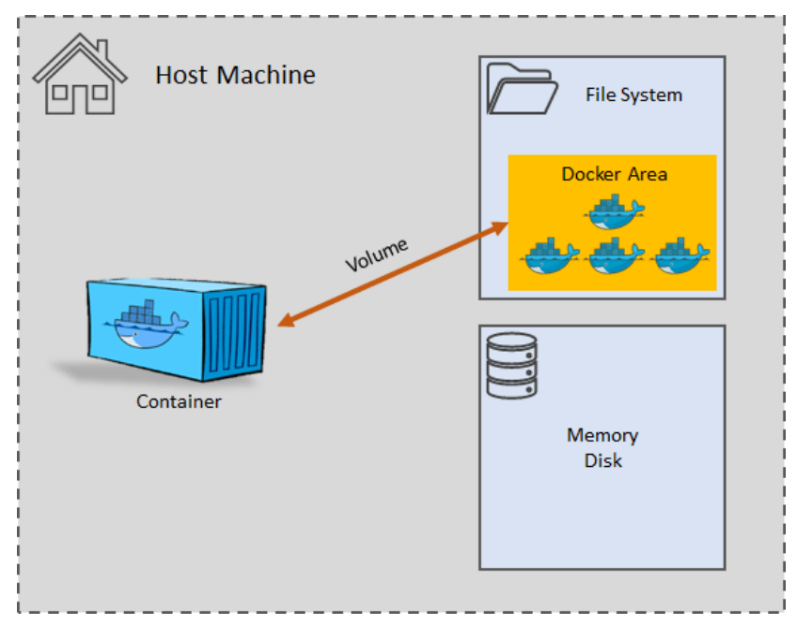
**4. Docker Volumes 数据卷**

方案设计之前我们需要了解 [Docker Volumes 数据卷](https://yeasy.gitbook.io/docker_practice/data_management/volume)

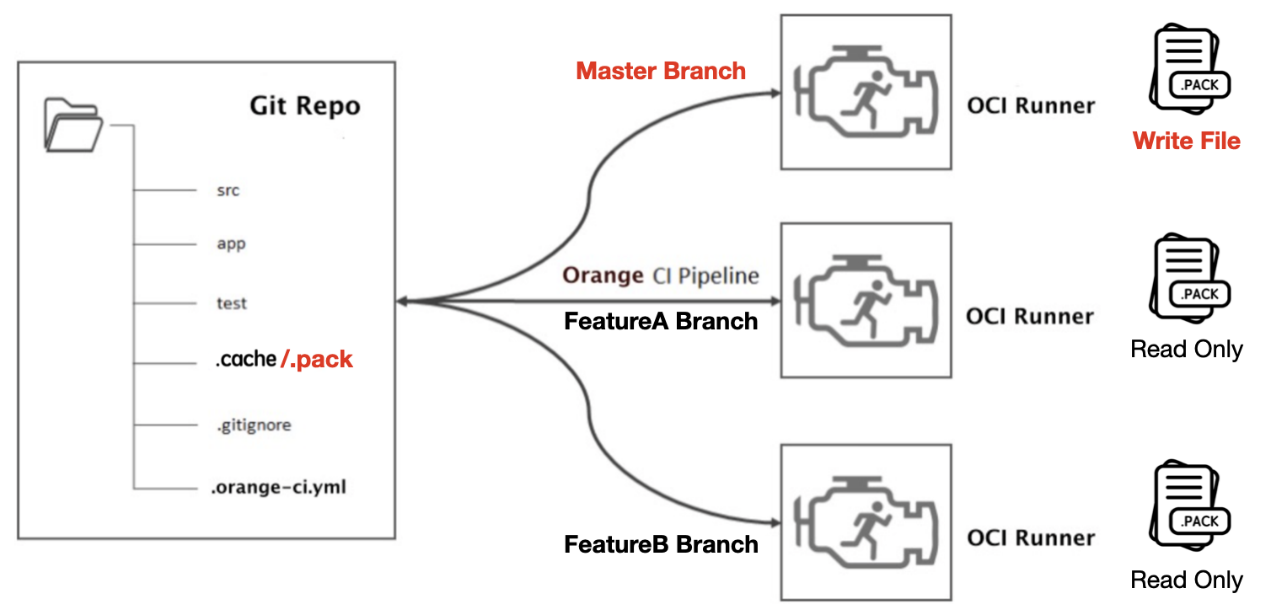
数据卷 是一个可供一个或多个容器使用的特殊目录，可以提供很多有用的特性：

* **数据卷 可以在容器之间共享和重用**
* 对 数据卷 的修改会立马生效
* 对 数据卷 的更新，不会影响镜像
* **数据卷 默认会一直存在，即使容器被删除**

如图是数据卷在 Docker里面的位置



OCI使用了docker作为runner，可以开启OCI的 [Volumes 能力](http://doc.orange-ci.woa.com/grammar.html#_3-volumes-%E7%A4%BA%E4%BE%8B)(OCI团队给力！)，结合Webpack Cache方案如下



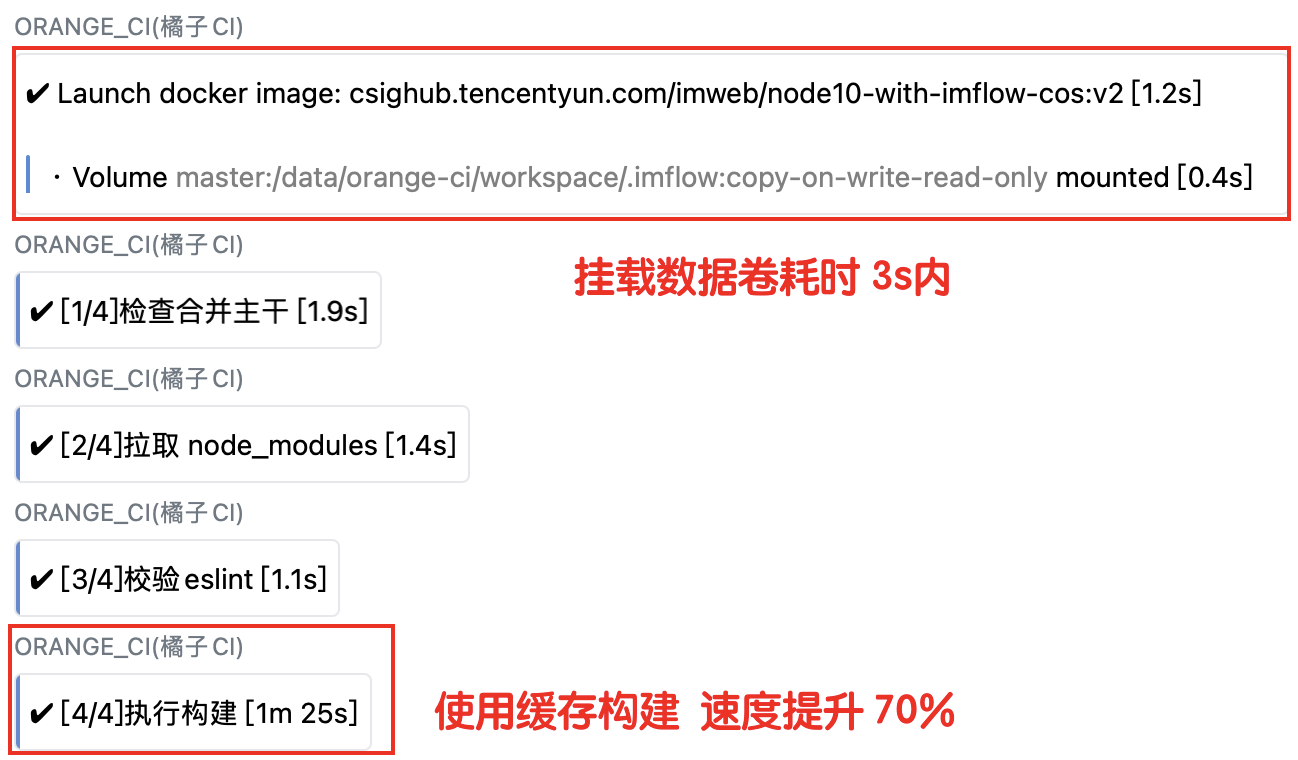
1. 指定项目的缓存路径，示例是.cache目录，在.orange-ci.yml 文件声明volumes 数据卷，路径是项目在runner中的绝对路径
2. 触发master分支构建后将缓存文件(.pack)写入缓存路径，**OCI会进行 copy-on-write 操作保留在Runner上**，这里master分支的策略是可读写文件
3. 当前构建的OCI Runner将会永久保存这份.pack缓存，**再次构建时使用到这台Runner即可使用缓存进行构建加速**
4. 当所有的Runner(指定affinity，参考下文)都执行过一次构建后，**所有的分支的构建任务都将能利用缓存加速构建**

**master分支**需要进行写入操作，读写策略为 **copy-on-write**

**开发分支**需要将策略改为**copy-on-write-read-only** 实现**只读缓存**的操作



**实践效果**

****

**小结：**

1. 通过配置数据卷，**能将缓存永久地保存在Runner上**，实现了类似本地开发的效果，**配置使用成本最低**
2. Runner不共享数据，项目**需要多次触发master构建填充**完所有Runner后，所有的分支才能构建将都能使用缓存

**五、CI环境缓存持久化方案选择**

**1. 方案选型**

第四部分的方案似乎**都能实现缓存持久化**，那么我们应该怎么选择呢

下面从几个纬度去比较方案

* **前置条件：**指实现方案需要做的准备或依赖
* **改造/接入成本：**指项目CI流程改造需要的操作成本
* **时间成本：**指方案消耗的时间
* **空间成本：**指方案需要占用的存储空间（如COS对象/镜像）
* **缓存刷新速度：**指最新缓存生成后到构建可用的时间

| **方案** | **前置条件** | **改造/接入成本** | **时间成本** | **空间成本** | **缓存刷新速度** | **综合评价** |
| --- | --- | --- | --- | --- | --- | --- |
| 对象存储 COS | 无 | 高（COS操作） | 极高（网络） | 中 | 快 | 不推荐 |
| Docker Cache | CI支持DockerCache | 高（Cache+COS操作） | 低 | 中 | 快 | 可选 |
| Docker Image | 无 | 中（镜像操作） | 低 | 中 | 快 | **推荐** |
| Docker Volumes | CI支DockerVolumes | 低 | 极低 | 低 | 中（需填充Runner） | **推荐** |

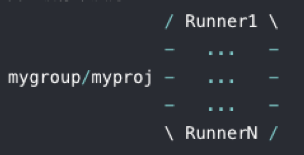
**选型结论：**

1. 对象存储COS方案不推荐使用，因为**改造成本高，时间成本极高**
2. Docker Cache可选择性使用，因为改造过程较复杂，其他表现一般
3. **推荐使用 Docker Volumes，但注意需要CI支持 Volumes 能力**
4. **推荐使用Docker Image方案**，能较好实现持久化，尤其是**CI不支持Volumes的情况下**
5. Runner较多的情况可优先考虑 Docker Image，快速刷新缓存

**2. 构建机优化**

确定了持久化方案，继续考虑**对构建环境进行优化**，当前项目OCI构建机轮转策略是 [随机轮转](http://doc.orange-ci.woa.com/grammar.html#affinity)，

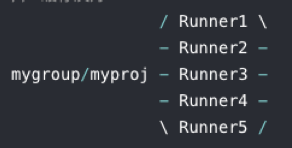
docker cache/ docker volumes 都无法固定机器使用，构建时从runner池**随机挑选，缓存命中率较低**



 命中缓存带来的收益是巨大的，参考下面对比图

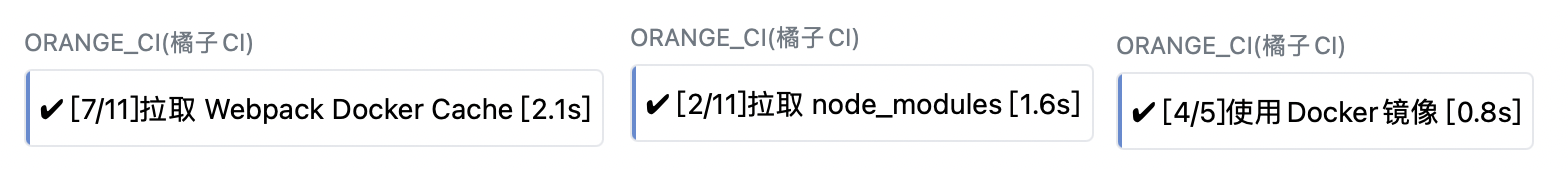


**为了提高缓存命中率**，需要将调整OCI构建轮转策略为**[Hash轮转](http://doc.orange-ci.woa.com/grammar.html" \l "affinity" \t "_blank)**



**固定数量顺序轮转调度构建机 docker cache 在每台机器留下缓存后 缓存版本不变的情况下，缓存命中率 100%！**

如图命中缓存的情况下操作时间消耗控制在 3s 内



**小结：合理配置构建机轮转策略，有利于缓存的使用，提升构建效率**

**至此我们就完成了方案的选型和优化～ 下一步就是在项目中的应用了**

**六、优化数据展示**

**1. OCI 构建数据**

OCI项目构建的数据查询接口点[这里](http://api.orange-ci.oa.com/graphql" \t "_blank)  
OCI完整链路如下：

**提交代码 →  触发OCI → 检查合并主干 → 安装依赖 → 校验eslint → 执行构建 → 同步文件至nohost –> 构建完成**

****

1. 企鹅辅导H5项目

**数据环境：**测试环境、push 触发构建一周数据

| **构建时长** | **平均** | **最低** | **最高** |
| --- | --- | --- | --- |
| Webpack4 | 5m31s | 4m25s | 8m30s |
| Webpack5 | 2m43s | 1m52s | 4m39s |
| 对比 | 提升50% | 提升57% | 提升45% |

数据总结：优化后平均构建效率 提升50%+

2.企鹅辅导PC项目

**数据环境：**测试环境、push 触发构建一周数据

| **构建时长** | **平均** | **最低** | **最高** |
| --- | --- | --- | --- |
| Webpack4 | 4m46s | 4m15s | 7m19s |
| Webpack5 | 2m40s | 2m16s | 4m35s |
| 对比 | 提升44% | 提升46% | 提升38% |

数据总结：优化后平均构建效率 提升40%+

**2. 本地开发数据**

**数据环境：**企鹅辅导H5项目、本地机器主动触发测试

1. 本地启动 npm run dev 测试

| **构建时长** | **首次DEV** | **二次DEV** | **三次DEV** |
| --- | --- | --- | --- |
| Webpack4 | 2.6m | 25s | 24s |
| Webpack5 | 2.34m | 7.27s | 6.21s |
| 对比 | 用时接近 | 提升71% | 提升74% |

数据总结：优化后二次构建 70%+提升，构建在10s内，大幅提升了开发体验

**3. 优化数据总结**

* 使用Webpack5缓存后本地开发**首次构建速度接近**，**二次构建有大幅提升（70%）**
* Webpack5缓存结合CI能**明显提升线上构建速度（40%～50%）**

**七、进一步的优化和思考**

经过上文的探索和实践，**Webpack5结合OCI 已经为团队项目构建显著提升了效率**

但现在的缓存方案还存在优化空间，因为缓存是基于master的，在涉及到分支开发改动较大或者依赖文件变更将会使缓存命中率降低或失效

要做到更好的持久化缓存还需要继续探索，如以下几点：

* 按分支使用缓存文件来保证分支缓存版本效率和可用性
* 抽象通用工具，实现其他项目的缓存持久化快速接入
* 如何确保构建产物安全，做好缓存对构建产物零影响

前端工程化是永远在进行中的话题，本文从个人实践角度浅显地分析了Webpack5缓存持久化方案，难免有疏漏和错误，欢迎各位大佬指正交流！也希望文章能为大家带来一点小小的帮助和启发～

最后感谢在接入Webpack5过程中 **haigecao、enjoychen、jsonhuang** 的建议和指导，OCI团队 **youkunhuang、loviselu**大佬的支持和帮助！

最后更新于 2021-09-02 20:42

已申报

如果觉得我的文章对您有用，请随意赞赏

赏

[1人已赞赏](javascript:void();)

[](https://km.woa.com/user/haigecao)

仅供内部学习与交流，未经公司授权切勿外传

分类：[Web](https://km.woa.com/group/edu/articles?cat_id=155065" \t "_blank) 标签：[构建](https://km.woa.com/group/edu/articles?tag_id=9006" \o "构建" \t "_blank)(1) [构建优化](https://km.woa.com/group/edu/articles?tag_id=164364)(1) [webpack](https://km.woa.com/group/edu/articles?tag_id=105121)(2) [OCI](https://km.woa.com/group/edu/articles?tag_id=85951)(1) [缓存](https://km.woa.com/group/edu/articles?tag_id=919)(1)



本文专属二维码，扫一扫还能分享朋友圈

想要微信公众号推广本文章？[点击获取链接](javascript:void(0);)

我顶 (35)

收藏 (82)

* [转载](javascript:void(0);)

* [收录](javascript:void(0);)
* [评论](javascript:void(0);)(10)
* [反馈](javascript:void(0);)

分享到

大家评论

[](https://km.woa.com/user/sourceliu)

[sourceliu](https://km.woa.com/user/sourceliu)

2021-09-02 16:33:54

king of webpack

 顶 [回复](javascript:void(0);)

[](https://km.woa.com/user/xpniceli)

[xpniceli](https://km.woa.com/user/xpniceli)

2021-09-02 16:36:20

太强了 逐个字节进行学习

 顶 [回复](javascript:void(0);)

[](https://km.woa.com/user/loviselu)

[loviselu](https://km.woa.com/user/loviselu)

2021-09-02 17:48:03

太强了

 顶 [回复](javascript:void(0);)

[](https://km.woa.com/user/bobbli)

[bobbli](https://km.woa.com/user/bobbli)

2021-09-02 18:51:49

> 3. 持久化缓存CI测试  
  
Orange-CI 这个功能已经实现，感受不到的原因是 runner 是随机分配，缓存还没做网络盘跨机共享，可以先给 runner 设置 affinity 为 1 即可，快来尝鲜  
  
<http://docs.orange-ci.oa.com/grammar.html#affinity>

 顶 [回复 (2)](javascript:void(0);)

[](https://km.woa.com/user/mickyzeng)

[mickyzeng](https://km.woa.com/user/mickyzeng) (楼主)

2021-09-02 19:39:35

affinity 为1的确能持久化缓存，但考虑不同分支代码覆盖缓存，缓存的可用性其实是不可控的， 而且为1的情况可能不利于容灾～

[回复](javascript:void(0);)

[](https://km.woa.com/user/bobbli)

[bobbli](https://km.woa.com/user/bobbli)

2021-09-02 21:45:48

[@mickyzeng](https://km.woa.com/user/mickyzeng) 嗯，等后面找到比较好的跨 runner 共享缓存后，就能不设置 affinity 为 1 了，现在走网络盘的时候会导致 IO 性能骤降，所以先用 affinity: 1 来用

[回复](javascript:void(0);)

[](https://km.woa.com/user/aflextyang)

[aflextyang](https://km.woa.com/user/aflextyang)

2021-09-03 10:25:53

优秀

 顶 [回复](javascript:void(0);)

[](https://km.woa.com/user/louisshwliu)

[louisshwliu](https://km.woa.com/user/louisshwliu)

2021-09-03 14:43:46

6啊

 顶 [回复](javascript:void(0);)

[](https://km.woa.com/user/enjoychen)

[enjoychen](https://km.woa.com/user/enjoychen)

2021-09-03 15:38:44

强强强

 顶 [回复](javascript:void(0);)

[](https://km.woa.com/user/elliothu)

[elliothu](https://km.woa.com/user/elliothu)

2021-09-03 15:40:01

优秀！

 顶 [回复](javascript:void(0);)

[](https://km.woa.com/user/rickyrqzhao)



[切换到更多功能](javascript:void(0);)

关于作者

[](https://km.woa.com/user/mickyzeng)

[mickyzeng(曾志强)](https://km.woa.com/user/mickyzeng)

CSIG\在线教育部\Web开发四组员工

作者文章

* [Webpack5 持久化缓存实践—CI提速50%!](https://km.woa.com/posts/show/519978?kmref=author_post)
* [企鹅辅导H5性能极致优化](https://km.woa.com/posts/show/513052?kmref=author_post)

Copyright©1998-2021 Tencent Inc. All Rights Reserved

腾讯公司研发管理部 版权所有

[广告申请](https://km.woa.com/chartlets/km/) [反馈问题](javascript:void(0);)

[500/535/518 ms]