Производительность веб-сайтов и приложений можно значительно повысить за счет повторного использования ранее полученных ресурсов.

**Кэширование (или кэш)** - это некий промежуточный буфер, в котором хранятся данные. Благодаря кэшированию страница сайта не воссоздается заново для каждого пользователя. Кэширование позволяет осуществлять работу с большим количеством данных в максимально сжатые сроки и при ограниченных ресурсах (серверных и пользовательских).

Необходимо понимать, что работу с данными можно производить как на стороне клиента, так и на сервере. Притом, серверная обработка данных централизована и имеет ряд несомненных преимуществ (особенно для службы поддержки).

**Категории кэшей**:

* **Приватный** (private) кэш браузера: предназначен для отдельного пользователя. Вы, возможно, уже видели параметры кэширования в настройках своего браузера. Кеш браузера содержит все документы, загруженные пользователем по HTTP. Он используется для доступа к ранее загруженным страницам при навигации назад/вперед, позволяет сохранять страницы, или просматривать их код, не обращаясь лишний раз к серверу. Кроме того, кэш полезен при отключении от сети.
* **Общий** (shared) прокси-кэш: это кэш, который сохраняет ответы, чтобы их потом могли использовать разные пользователи. Например, в локальной сети вашего провайдера или компании, может быть установлен прокси, обслуживающий множество пользователей, чтобы можно было повторно использовать популярные ресурсы, сокращая тем самым сетевой трафик и время ожидания.

**Виды кэширования**:

**1. Браузерное кэширование или клиентское кэширование**

Представляет собой составление для браузера команды использовать имеющуюся кэшированную копию. Работа такого кэширования основана на том, что при повторном посещении, браузеру отдаётся заголовок 304 Not Modified, а сама страница или картинка загружаются из локального пользовательского кэша. Получается, что вы экономите на трафике между браузером посетителя и хостингом сайта. Соответственно, страница вашего сайта начинает загружаться быстрее.

**1.1 Кэширование файлов и картинок**

Браузерное кэширование как нельзя лучше подходит для сайтов, содержащих большое количество изображений: картинка не скачивается каждый раз при открытии сайта, а просто загружается через кэш браузера. Это первый уровень кэширования, который состоит в отдаче заголовка «expired» и заголовка «304 Not Modified». Наиболее эффективным считается кэширование на 2 недели. Однако в данном случае есть важный нюанс: если изображение на сайте меняется, то браузер узнает об этом не сразу, а только если выждать expiry или сбросить кэш в самом браузере. Это не очень эффективно, если файл постоянно изменяется и необходимо постоянно отдавать его актуальную версию.

**1.2 Кэширование https**

Специальные заголовки вида strict-security. Позволяет браузеру всегда обращаться по https к выбранному домену. Сохраняет это состояние довольно жёстко и, в случае отмены этого вида кэша, браузер ещё довольно долго будет пытаться загрузить страницу по https, при этом игнорируя текущие заголовки.

**1.3 Кэширование центра сертификации**

Так называемый, stamp центра сертификации.

Данный вид кэширования считается обязательным для применения, если вы не хотите, чтобы пользователи вашего сайта ждали, когда центр сертификации (а это некий сервер, который отвечает за достоверность вашего сертификата) обработает запрос от браузера пользователя и подтвердит, что ваш сайт действительно подтверждён им.

**1.4 Кэширование страниц**

Когда страница уже сгенерирована, нужно постоянно отслеживать ее актуальность. Для этого вы должны использовать серверный кэш с отслеживанием времени изменения отдельных частей страницы (если страница строится из множества динамически генерируемых блоков). При таком подходе в каждом ответе от сервера установлены специальные заголовки, обозначающие время изменения страницы, которые затем отправляются браузером пользователя при повторном обращении к странице сайта. Сервер при получении таких заголовков можем проанализировать текущее состояние страницы (возможно, даже отрисовать её), но вместо содержимого страницы отдать заголовок «304 Not Modified», что для пользовательского браузера будет означать, что можно показать страницу из своего (браузера пользователя) кэша.

Конечно, можно отправлять соответствующие заголовки без использования серверного отслеживания кэша, но в таком случае большинство пользователей получат обновление контента страницы довольно поздно. При таком подходе браузер иногда опрашивает сервер для получения обновлений, но периодичность и правила для каждого браузера настраиваются его разработчиком, поэтому надеяться на то, что ваши пользователи получат обновления вовремя, не приходится.

Как правило, кэш подразделяется по типу пользователей:

* для авторизованных;
* для неавторизованных.

Данное разделение обусловлено уникальностью контента, для каждого авторизованного пользователя и общностью контента для гостевых пользователей. В большинстве сайтов неавторизованный пользователь не может изменять содержимое сайта, а значит и влиять на его содержимое.

Браузерный кэш позволяет экономить трафик и время, затрачиваемое на загрузку страниц. Но для достижения эффекта экономии, пользователь должен хотя бы один раз посетить нашу страницу, а это означает, что нагрузка на серверные ресурсы уменьшится, но не значительно.

**2. Серверное кэширование**

Под серверным кэшированием понимаются все виды кэширования, при котором данные хранятся на серверной стороне. Эти данные не доступны клиентским браузерам. Кэш создается и хранится по принципу «один ко многим» (многие, в данном случае, - это клиентские устройства).

**2.1 Кэширование страницы целиком**

Наиболее эффективный кэш. Чем он интересен? Самое большое его достоинство в том, что отдача страницы происходит практически в момент обращения, как следствие - это возможность обработки миллионов запросов даже на самом слабом сервере со скоростью работы памяти и с незначительным задействованием процессора.

Пожалуй, любой когда-либо мечтал о сайте, работающем со скоростью «ping» или быстрее.

Но и у этого типа кэша есть свои минусы: например, невозможность кэшировать страницы для авторизованного пользователя, либо пользователя, содержимое страницы которого зависит от текущих переменных пользователя.

Используйте этот кэш, если серверу известны все статичные состояния внешних данных, такие как: uri, get (без дополнительных параметров), пользователь не авторизован - то есть, фактически, это идеальное состояние страницы для гостевых пользователей. Учитывайте тот факт, что при таком кэшировании архитектура сайта или приложения всегда должна однотипно обрабатывать входящие запросы и отдавать однотипные ответы. Такое состояние есть в любом приложении или сайте, его нужно лишь отследить и применить к нему кэш.

Кэширование страниц целиком, чаще всего, применяют в каких-то экстренных случаях, при этом кэш страниц сохраняется на заранее указанное время (от 2 минут), в течение которого ответы от сервера однотипны (не позволяйте браузеру кэшировать это).

**2.2 Кэширование результатов компиляции php-файлов**

Различают как чистую компиляцию кода, так и его оптимизацию во время компилирования (подмена скриптов).

**2.3 Кэширование отдельных блоков страницы**

Это, пожалуй, самый интересный, но и сложный вид кэширования. Тем не менее, он тоже может быть эффективным, и на его примере легче всего объяснить принципы кэширования в целом.

Необходимо отслеживать: состояние таблиц, состояние сессии пользователя, выключать ли кэширование при POST или GET запросах (http query), зависимость от текущего адреса, постоянство кэширования (при изменении предыдущих условий) или его динамическую подстройку.

Кэширование отдельных блоков страниц лучше других типов кэширования подойдет, если вам нужно, например, уменьшить количество запросов к базе данных от реальных (авторизованных) пользователей. Кстати, при правильно заданных зависимостях, он будет работать даже эффективнее, чем все последующие виды кэширования.

Почему этот вид кэширования настолько важен? Всё дело в том, что расширение пула серверов баз данных намного более сложная задача, чем расширение пула серверов php-части сайта. Более того, php конфликты состояния кэширования решаются гораздо легче, чем конфликты при работе с множеством баз данных.

**2.4 Кэширование php на основе неразделяемых ресурсов**

Лучше всего подходит при стандартизации запросов, получении данных из общих ресурсов, наличии внутренних переменных, к которым php-ресурсы обращаются несколько раз при генерации страницы.

**2.5 Кэширование php на основе общих ресурсов**

Такое кэширование применяйте для хранения сериализированных данных. Например: конфигурационного файла, состояния таблиц, списков файловой системы.

**2.6 Кэширование mysql на основе query cache**

Это довольно известная и наиболее освещенная тема. Тем не менее, хотелось бы рассмотреть специфику работы с timestamp и то, как можно избежать постоянного сброса query cache.

Наверняка, вы регулярно сталкивались с ситуацией, когда необходимо отдать новые материалы, дата публикации которых уже разрешена текущим timestamp? Проще говоря, WHERE show\_ts<=UNIX\_TIMESTAMP()

Если использовать постоянно меняющийся timestamp в таких запросах, то sql кэш будет не только бесполезен, но даже вреден, так как будет копиться количество кэшированных запросов, данные которых устарели в момент создания кэша. Мы предлагаем следующий выход из ситуации:

Как правило, любой материал публикуется в определенные моменты времени. К примеру, 00:00. Всё что нужно сделать - создать запрос, который будет оценивать таблицу по максимальной дате, при этом, меньшей текущей.

Что-то вроде: SELECT SQL\_NO\_CACHE MAX(show\_ts) … WHERE show\_ts<=UNIX\_TIMESTAMP();

Да, этот запрос кэшироваться не будет, но будут кэшироваться все запросы к этой таблице, если их количество больше одного. Эта простая операция существенно улучшит жизнь sql-кэширования.

Кэшировать эти запросы имеет смысл, если чтений из таблицы немного больше чем записи.

**2.7 Кэширование mysql результатов работы, агрегирующие таблицы**

Существует правило: обновлений данных должно быть значительно меньше, чем чтения для их отдачи. То есть не имеет смысл агрегировать то, что изменится в тот же момент, при этом важна актуальность агрегированных данных. Что выбирать для агрегирования? Обычно это какая-то статистическая информация о числе записей, дате последнего обновления, авторе последнего обновления и тому подобное.

Существуют также кеши шлюзов, [CDN](https://www.cloudflare.com/learning/cdn/what-is-caching/), реверсные прокси кеши и балансировщики нагрузки, разворачиваемые на серверах для повышения надежности, производительности и масштабируемости веб-сайтов и веб-приложений.

**3. Кэширование и прокси-серверы**

В компьютерных сетях прокси-серверы могут быть представлены специальным аппаратным обеспечением или соответствующими приложениями. Они играют роль посредников между клиентами и серверами, хранящими данные, которые этим клиентам требуются. Кэширование - это одна из задач, которую они решают. Рассмотрим различные виды прокси-серверов.

**3.1 Шлюзы**

Шлюз (gateway) - это прокси-сервер, который перенаправляет входящие запросы или исходящие ответы, не модифицируя их. Такие прокси-серверы ещё называют туннелирующими прокси (tunneling proxy), веб-прокси (web proxy), прокси (proxy), или прокси уровня приложения (application level proxy). Эти прокси-серверы обычно совместно используются, например, всеми клиентами, находящимися за одним и тем же файрволом, что делает их хорошо подходящими для кэширования запросов.

**3.2 Прямые прокси-серверы**

Прямой прокси-сервер (forward proxy, часто такие серверы называют просто proxy server) обычно устанавливается на стороне клиента. Веб-браузер, который настроен на использование прямого прокси-сервера, будет отправлять исходящие запросы этому серверу. Затем эти запросы будут перенаправлены на целевой сервер, расположенный в интернете. Одно из преимуществ прямых прокси заключаются в том, что они защищают данные клиента (однако, если говорить об обеспечении анонимности в интернете, безопаснее будет пользоваться VPN).

**3.3 Веб-ускорители**

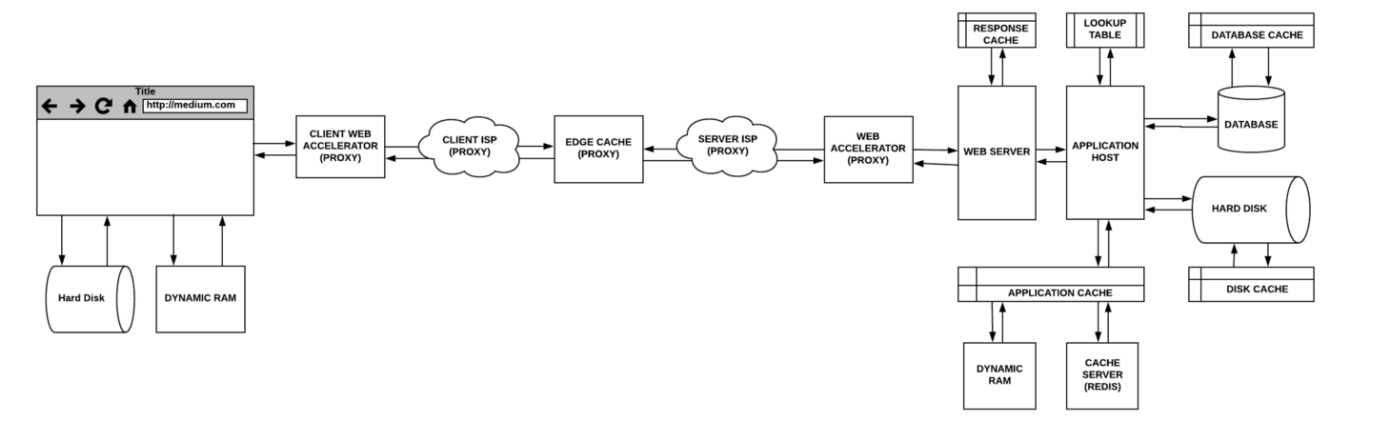
Веб-ускоритель (web accelerator) - это прокси-сервер, который уменьшает время доступа к сайту. Он делает это, заранее запрашивая у сервера документы, которые, вероятнее всего, понадобятся клиентам в ближайшем будущем. Подобные серверы, кроме того, могут сжимать документы, ускорять выполнение операций шифрования, уменьшать качество и размер изображений, и так далее.

**3.4 Обратные прокси-серверы**

Обратный прокси-сервер (reverse proxy) - это обычно сервер, расположенный там же, где и веб-сервер, с которым он взаимодействует. Обратные прокси-серверы предназначены для предотвращения прямого доступа к серверам, расположенным в частных сетях. Обратные прокси используются для балансировки нагрузки между несколькими внутренними серверами, предоставляют возможности SSL-аутентификации или кэширования запросов. Такие прокси выполняют кэширование на стороне сервера, они помогают основным серверам в обработке большого количества запросов.

**3.5 Пограничное кэширование**

Обратные прокси-серверы расположены близко к серверам. Существует и технология, при использовании которой кэширующие серверы располагаются как можно ближе к потребителям данных. Это - так называемое пограничное кэширование (edge caching), представленное сетями доставки контента (CDN, Content Delivery Network). Например, если вы посещаете популярный веб-сайт и загружаете какие-нибудь статические данные, они попадают в кэш. Каждый следующий пользователь, запросивший те же данные, получит их, до истечения срока их кэширования, с кэширующего сервера. Эти серверы, определяя актуальность информации, ориентируются на серверы, хранящие исходные данные.



Примечание. В тестировании практически во всех случаях правило - очищать кэш после каждого прохода теста (если только это не целенаправленной тестирование самого кэша или требуется наличие кэша по каким-либо причинам). Дело в том, что кэш очевидно искажает показатели performance testing, а также может быть причиной ошибочного дефект-репорта из-за устаревания и/или несогласованности актуальных и сохраненных данных. В некоторых ситуациях без очистки кеша не обойтись даже просто из-за огромного количества кэшируемых данных.