**Лекция «Оптимизация кода веб-приложений»**

Содержание:

[Оптимизация front-end 1](#_Toc135296515)

[1. Уменьшите ресурсы 1](#_Toc135296516)

[2. Уменьшите количество обращений к серверу 1](#_Toc135296517)

[3. Удалите ненужные пользовательские шрифты 1](#_Toc135296518)

[4. Постарайтесь сжать файлы 2](#_Toc135296519)

[5. Оптимизируйте изображения 2](#_Toc135296520)

[6. Применяйте технологию Lazy Loading 3](#_Toc135296523)

[7. Кеширование 4](#_Toc135296524)

[8. Включите предварительную выборку 4](#_Toc135296525)

[9. Используйте сеть доставки контента 5](#_Toc135296528)

[Оптимизация back-end 5](#_Toc135296529)

[Что такое действительно хорошая производительность PHP? 5](#_Toc135296530)

[Когда следует начинать оптимизировать PHP-код? 6](#_Toc135296531)

[Советы по оптимизации PHP-скриптов 7](#_Toc135296532)

[Узкие места производительности PHP 8](#_Toc135296533)

[Еще советы по улучшению производительности PHP 9](#_Toc135296534)

[Необходимо уметь контролировать производительность PHP 11](#_Toc135296535)

[Перспективы улучшения производительности PHP 11](#_Toc135296536)

# Оптимизация front-end

Большинство веб-сайтов не обеспечивают удовлетворительных впечатлений от продукта из-за плохо оптимизированного фронтенда. И чаще всего это связано с неоптимальной загрузкой данных и оптимизацией изображений. В этой статье я рассмотрю девять рекомендаций, которые будут полезны для оптимизации загрузки данных.

### **1. Уменьшите ресурсы**

Минификация ресурсов относится к процессу удаления ненужных, избыточных данных из HTML, CSS и JavaScript, которые не требуется загружать. Она включает в себя удаление комментариев и пробелов, неиспользуемого кода, символов новой строки и т. д.

Минификация HTML, CSS и JavaScript ускоряет время загрузки внешнего интерфейса, поскольку сокращает объем кода, который необходимо запрашивать от сервера.

Воспользуйтесь следующими инструментами для создания оптимизированной версии HTML кода, CSS и JavaScript.

HTML: [PageSpeed Insights](https://developers.google.com/speed/pagespeed/insights/" \t "_blank), [HTML Minifier](https://github.com/kangax/html-minifier)

CSS: [cssmin.js](https://www.phpied.com/cssmin-js/), [Chrome Dev Tools](https://developer.chrome.com/docs/devtools/coverage/), [YUI Compressor](https://yui.github.io/yuicompressor/)

JavaScript: [JSMin](https://www.crockford.com/jsmin.html" \t "_blank), Chrome Dev Tools

### **2. Уменьшите количество обращений к серверу**

В общем, чем больше запросов ваш интерфейс делает на сервер, тем больше времени нужно для его загрузки. Вы можете использовать несколько способов для уменьшения количества запросов к серверу, необходимых для загрузки страницы.

Используйте CSS-спрайты — это один из самых простых способов уменьшить количество обращений к серверу. Вместо того, чтобы загружать на сайт десять отдельных изображений, спрайт загружает один файл изображения, объединенный с использованием коллекции изображений. Вы можете использовать свойства background-image и background-position, чтобы отобразить нужный сегмент изображения. Делая это, вы сокращаете количество требуемых запросов к серверу.

Уменьшите количество сторонних плагинов, которые делают большое количество внешних запросов.

Исправьте неработающие ссылки на несуществующие файлы.

Кроме того, вы также можете посмотреть рендеринг на стороне сервера, чтобы ускорить начальную загрузку приложения. Если при загрузке первой страницы, присутствуют минимальные данные — это существенно повлияет на производительность.

### **3. Удалите ненужные пользовательские шрифты**

Пользовательские шрифты стали очень популярными, поскольку они помогают добавить на сайт индивидуальность. Однако они также приводят к снижению производительности.

Пользовательские шрифты могут быть довольно большого размера, а веб-шрифты, такие как шрифты Google, добавляют HTTP-запросы к внешним ресурсам. Это вредит скорости рендеринга страницы.

Ниже приведены некоторые действия, которые вы можете предпринять при использовании шрифтов на своем веб-сайте:

Преобразование шрифтов в наиболее эффективный формат: загрузка современного формата, такого как WOFF2, может уменьшить размер файла примерно на 30% по сравнению с другими форматами.

Подмножество шрифтов для удаления неиспользуемых символов: комплексные файлы шрифтов содержат символы для многих языков, которые могут никогда не использоваться. Разделив шрифты на подмножества, мы можем удалить ненужные символы из шрифта и оставить только то, что нам нужно для написания контента на сайте.

Предварительно загружайте шрифты, которые явно требуются для страницы.

Использование пары нестандартных шрифтов может не причинить вреда, но их чрезмерное использование может существенно повлиять на время загрузки вашего интерфейса. Поэтому убедитесь, что вы не переоцениваете влияние и потребность в пользовательских шрифтах, которые используются на вашем сайте.

### **4. Постарайтесь сжать файлы**

Объем пропускной способности ограничивает объем данных, доставляемых в заданное время. Чем больше размер файла, тем больше времени требуется для загрузки. Современные веб-сайты часто имеют большие пакеты HTML, CSS и JavaScript.

Сжимая файлы подходящим способом, вы легко можете существенно изменить время загрузки вашего сайта. Ниже приведены два хороших варианта сжатия файлов:

Gzip — самый популярный метод сжатия и распаковки данных, который в настоящее время поддерживается всеми современными браузерами. Gzip сжимает пакеты HTML, CSS и JavaScript на стороне сервера перед отправкой их в браузер, а на стороне клиента он распаковывает файлы и доставляет контент.

Brotli — предлагает лучшую степень сжатия по сравнению с другими доступными в настоящее время методами сжатия. Согласно исследованию CertSimple, Brotli сжимает файлы JavaScript на 14% меньше, чем Gzip, а степень сжатия HTML и CSS на 21% и 17% лучше, чем Gzip.

### **5. Оптимизируйте изображения**

Когда дело доходит до веб-сайтов, изображения являются жизненно важной частью. 93,7% веб-сайтов в Интернете используют хотя бы один формат файлов изображений, поскольку они помогают создать хорошее впечатление.

Однако отрицательная сторона использования изображений заключается в том, что они плохо влияют на время загрузки страницы, если изображения не оптимизированы. Однако есть несколько способов оптимизации изображений:

### **Используйте WebP или AVIF**

Использование новых форматов изображений, таких как WebP и AVIF, дает лучшую производительность, чем старые форматы, такие как JPEG и PNG.

WebP на 26% меньше по сравнению с PNG и на 25–35% меньше по сравнению с JPEG. AVIF на 50% меньше по сравнению с JPEG и на 20% меньше по сравнению с WebP.

Однако обратная сторона — поддержка браузера. WebP недавно получил поддержку в браузерах, поэтому более старые версии могут не поддерживать его. А формат AVIF поддерживается только в Chrome и Opera.

### **Используйте адаптивные изображения**

Еще один способ сократить время доставки изображений и повысить производительность веб-сайта — использовать адаптивные изображения. Более 50% трафика поступает со смартфонов и планшетов. Масштабирование изображения до популярных размеров устройства и их обслуживание с помощью srcset еще больше оптимизируют время загрузки сайта.

Помимо выбора подходящего формата и размеров, есть несколько других способов, чтобы сократить время загрузки внешнего интерфейса за счет оптимизации изображений. Ниже приведены некоторые дополнительные рекомендации, которые вы можете использовать:

Сжатие изображений.

Использование прогрессивных JPEG.

Показывайте изображения меньшего размера пользователям с медленным подключением.

Использование HTTP / 2 вместо HTTP / 1.1.

Использование наборов изображений.

### **6. Применяйте технологию Lazy Loading**

Lazy Loading помогает еще больше сократить время загрузки. При отложенной загрузке веб-страница сначала загружает только необходимый контент, а остальной контент загружается, когда он нужен пользователю.

Например, в результатах поиска изображений Google сначала загружается только небольшой набор изображений. Google загружает изображения-заполнители для скрытого содержимого вместо фактических, что сокращает время загрузки. Фактический контент будет отображаться, если пользователь прокручивает страницу вниз.

Помимо отложенной загрузки заполнителей, есть несколько других параметров, таких как «Собственная отложенная загрузка» и «Эффект размытого изображения», вы можете выбирать тот, который вам подходит.

### **7. Кеширование**

Каждый раз при уникальном посещении все, от HTML до CSS и JavaScript, необходимо загружать отдельно. Это приводит к более медленной загрузке данных.

Лучший способ предотвратить эту проблему — использовать параметры кэширования. При правильной настройке браузеры сохранят файлы в своем локальном кеше и не будут загружать одни и те же ресурсы при последующих посещениях страниц на сервере. Помимо кеша браузера, доступно несколько других опций кеширования для сокращения времени загрузки.

Кэш-сервер — запрос пользователя отправляется на ближайший кеш-сервер из набора кэш-серверов.

Кэш памяти — путем сохранения определенных частей данных в памяти (например, переменных JavaScript) без их перезагрузки для изменения маршрута.

Кэш диска — похож на кеш памяти, за исключением того, что он использует обычное хранилище файлов, обычно обрабатываемое браузером.

### **8. Включите предварительную выборку**

Предварительная выборка ресурсов — это еще один метод повышения производительности, который вы можете использовать для оптимизации загрузки данных веб-страницы.

Как разработчики, вы знаете свое приложение лучше, чем браузер. Таким образом, предварительная выборка использует эту информацию, чтобы подсказывать браузеру, какие ресурсы могут потребоваться в будущем.

В основном существует три типа предварительной выборки. Хотя предварительная выборка ссылок является наиболее популярным и широко используемым методом, предварительная выборка DNS и предварительная отрисовка также являются полезными вариантами.

### **Предварительная загрузка ссылок**

Предварительная выборка ссылок позволяет браузеру собирать ресурсы, которые пользователь может запросить в ближайшем будущем. Таким образом, разработчики могут предположить, что пользователи могут посетить конкретную веб-страницу.

Однако предварительная выборка ссылок работает только с кешируемыми ресурсами, такими как изображения и JavaScript.

### **Предварительная выборка DNS**

Каждый раз, когда пользователь запрашивает ресурс, размещенный на определенном IP-адресе, выполняется поиск DNS, чтобы найти доменное имя, которому принадлежит IP-адрес.

Предварительная выборка DNS позволяет браузеру выполнять поиск DNS для ссылок на веб-странице в фоновом режиме, в то время как пользователь просматривает текущую страницу, чтобы минимизировать задержку, когда пользователь кликает ссылку с включенной предварительной выборкой DNS.

### **9. Используйте сеть доставки контента**

Когда пользователь находится далеко от сервера географически, задержка увеличивается. Более того, загрузка запроса также может повлиять на время обслуживания контента.

Скорость загрузки внешнего контента, включая HTML-страницы, таблицы стилей, файлы JavaScript и изображения, можно оптимизировать с помощью CDN. Сеть доставки контента (CDN) — это группа серверов, распределенных в нескольких географических точках, которые хранят кэшированную версию контента для быстрой доставки конечному пользователю.

CDN перенаправляет запросы пользователей на ближайший сервер. Существует также определенный тип CDN, называемый CDN изображений, который отлично подходит для оптимизации ваших изображений. С их помощью вы можете сэкономить 40–80% размера файла изображения. Учитывая, что изображения обычно занимают более половины объема страницы, интеграция CDN для изображений может значительно повысить скорость загрузки.

# Оптимизация back-end

**Что такое действительно хорошая производительность PHP?**

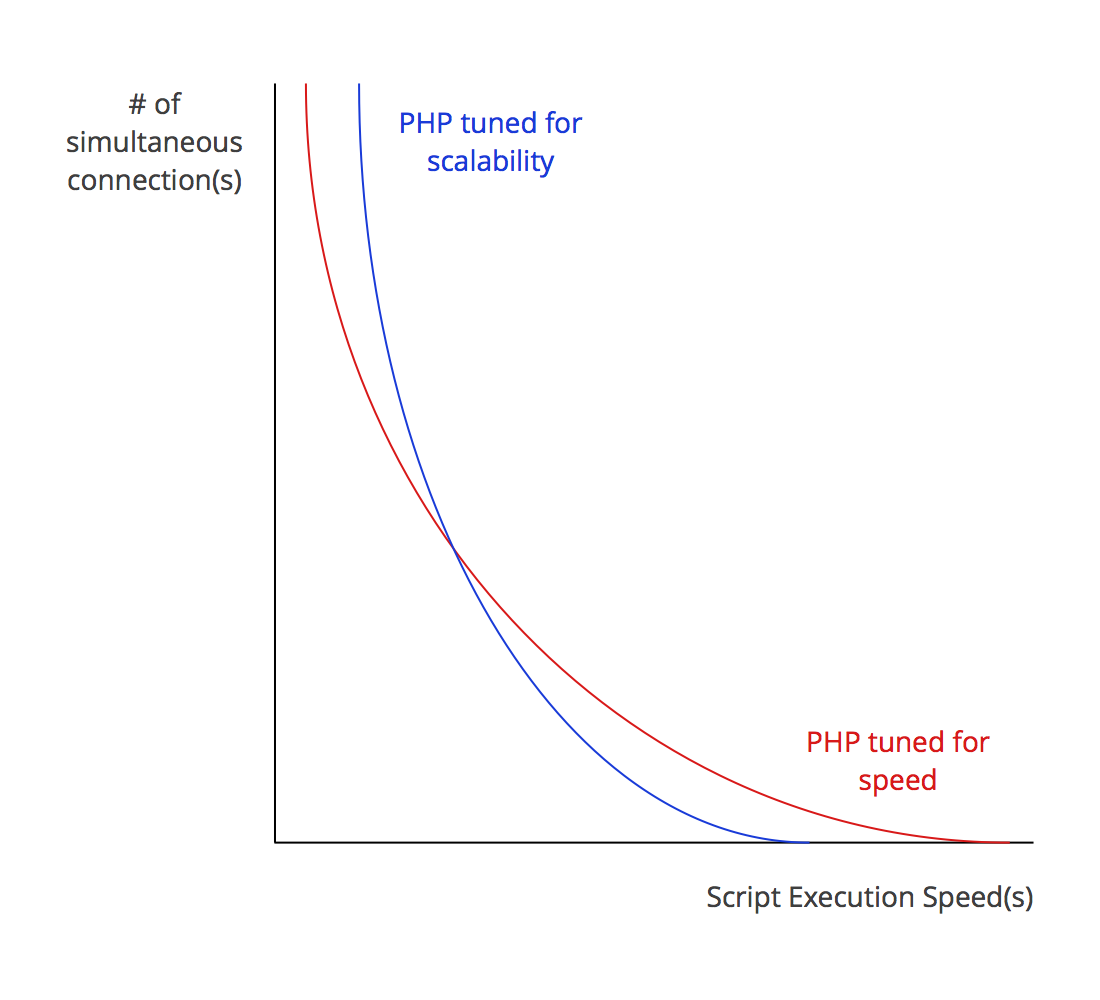
Следует иметь ввиду, что производительность и скорость - не являются синонимами. Оптимальная производительность балансирует между скоростью, безошибочностью и масштабируемостью. Например, при написании веб-приложения придется выбирать между двумя приоритетами:

1) *приоритетом скорости*, написав скрипт, который заранее загружает все в память;

2) *приоритетом масштабируемости* со скриптом, который загружает данные блоками.

На следующем рисунке 1 показан теоретический компромисс между скоростью и масштабируемостью.

Красная линия представляет сценарий, оптимизированный для скорости, а синяя линия — для масштабируемости. На горизонтальной оси отложена скорость выполнения скриптов), на вертикальной — количество одновременных вызовов сайта).



*Рис. 1. Теоретический компромисс между скоростью и масштабируемостью*

Из нижней части рисунка видно, что малом числе одновременных вызовов сайта скорость выполнения скриптов на красной линии выше, чем на синей. Из верхней части рисунка следует, что когда число пользователей растет, скорость выполнения скриптов на красной линии становится ниже, чем на синей. Скорость замедляется и на синей линии, когда трафик растет, но гораздо медленнее, чем на красной.

Получаем, что когда трафик превышает определенный порог, сценарий для скорости становится медленнее сценария для масштабируемости . Этот порог хорошо виден на рисунке как пересечение красной и синей линий.

Здесь можно провести аналогию с бегунами на разные дистанции: со *спринтером* (бегуном на короткие дистанции) и *стайером* (бегуном на длинные дистанции). Спринтеры намного быстрее бегут на короткие дистанции, но на более длинных они утомляются. Стайеры держат более медленный, но более стабильный темп, который позволяет им **сохранять энергию на больших расстояниях**. Аналогично, разные скрипты работают лучше в различных ситуациях. Правильный выбор скрипта для конкретного приложения требует наблюдения за активностью пользователей на сайте. При росте трафика придется переходить с одного скрипта на другой.

**Когда следует начинать оптимизировать PHP-код?**

Опытные программисты время от времени сохраняют протестированный код, заканчивая тем самым цикл проекта. Но это разумно делать только при хорошей производительности PHP-приложения!

Как добиться хорошей производительности PHP-приложения? Необходимо **проводить тесты во время процесса разработки**. Иначе придется переписывать большие блоки кода, чтобы заставить приложение нормально функционировать.

Оптимизировать PHP-код следует начинать перед созданием PHP-приложения! Необходимо сразу оценить состояние вашего железа и программного обеспечения, чтобы определить параметры их производительности. Эта информация при кодировании приложения позволит оценить риски и выигрыш конкретных компромиссов. Причем необходимо использовать **адекватные тестовые данные**, иначе код приложения окажется бесперспективным.

**Советы по оптимизации PHP-скриптов**

Просто написание хорошего кода — важный первый шаг создания быстрых и стабильных PHP-приложений. Применяя с самого начала методы, описанные ниже, вы сэкономите время при поиске ошибок.

**1. Используйте готовые функции PHP**

Везде, где это возможно, используйте готовые функции PHP. Избегайте написания ваших собственных функций. Для этого потратьте немного времени на изучение функций PHP. Тогда код приложения получится более быстрым и эффективным.

**2. Используйте JSON вместо XML**

Функции PHP **json\_encode()** и **json\_decode()** просто невероятно быстры. Поэтому использование JSON предпочтительнее использования XML.

Если вам все же приходится разбираться с XML, лучше использовать шаблонные регулярные выражения, чем манипуляции с DOM.

**3. Используйте методы кэширования**

Кэш-память особенно полезна для сокращения объема загружаемых данных.

Кэширование байт-кода с помощью APC или OPcache сильно экономит время выполнения скомпилированного сценария.

**4. Уберите лишние вычисления**

Если одно и то же значение выражения используется многократно, вычислите его заранее и присвойте переменной. Тогда не придется его вычислять каждый раз.

**5. Используйте isset()**

Проводите сравнения с помощью пар **count()**, **strlen()** и **sizeof()**, **isset()**. Это быстрый и простой способ поиска значений, больше нуля.

**6. Отключите ненужные классы**

Если не планируется использовать классы или методы многократно, то они вообще не нужны. Если необходимо все же использовать классы, лучше использовать **методы производного класса**, поскольку они быстрее методов базовых классов.

**7. Отключите отладочные сообщения**

Сообщения об ошибках необходимы только во время кодирования. Но после запуска готовой задачи они становятся еще одним процессом, замедляющим выполнение кода. Отключите такие сообщения.

**8. Закрывайте соединения с базой данных**

Сбрасывание переменных и закрытие соединений с базой данных сэкономит драгоценную память.

**9. Ограничьте обращения к базе данных**

Старайтесь использовать совокупности запросов к базе данных. Это сокращает количество обращений к базе данных, приложение будет работать быстрее.

**10. Используйте строковые функции Str**

**str\_replace** быстрее, чем **preg\_replace**, а **strtr** в четыре раза быстрее, чем **str\_replace**.

**11. Используйте одинарные кавычки**

Когда только возможно, используйте одинарные кавычки, а не двойные. Двойные кавычки проверяются компилятором на переменные, что понижает производительность.

**12. Используйте три знака равенства**

Поскольку «**= = =»** проверяет величины только одного типа, это делает оператор сравнения «**= = =»** более быстрым, чем оператор «**= =»**.

**Узкие места производительности PHP**

Бывает, конечно, что переделка сценариев выгодна. Однако есть проблемы, которые понижают производительность PHP, не связанные с кодом приложения. Поэтому разработчикам нужно разбираться в подсистемах их сервера, чтобы **определить и устранить узкие места**. Ниже перечислены области, которые нужно проверять при появлении проблем с производительностью.

**1. Сеть**

Один из очевидных источников узких мест — это сети. Может просто не хватить ресурсов сети для обработки передаваемого объема данных.

**2. Центральный процессор**

Передача простых страниц HTML через сеть не истощает центральный процессор сервера, тогда как PHP-приложения перегружают его. Можно, по крайней мере, использовать многопроцессорный сервер, чтобы обработать PHP-код более эффективно.

**3. Совместно используемая память**

Отсутствие совместно используемой памяти снижает межпроцессорный обмен данными, что приводит к падению производительности. Поэтому, имея многопроцессорный сервер, не забывайте использовать совместную память.

**4. Файловая система**

Файловая система со временем становится фрагментированной. Поэтому используйте файловый кэш RAM, который ускорит доступ к диску, если этот кэш достаточного размера.

**5. Управление процессами**

Удостоверьтесь, что сервер не перегружен ненужными процессами. Удалите любые неиспользованные сетевые протоколы, антивирусные сканеры, почтовые серверы и драйверы оборудования.

Выполнение PHP в многопоточном режима также улучшает время отклика на запросы (но не рекомендуется, в общем случае, для высоконагруженных систем, потому что создает дополнительные издержки на переключение контекстов разных ядер).

**6. Другие серверы**

Если приложение зависит от внешних серверов, их узкие места будут снижать производительность. В такой ситуации, увы, мало что можно изменить. Тем не менее, всегда можно придумать какие-то изменения на своей стороне, чтобы смягчить такое падение производительности.

**Еще советы по улучшению производительности PHP**

**1. Используйте расширение ядра ZendOPCache**

Так как PHP интерпретируется в выполняемый код, программисту приходится повторно компилировать код даже при небольшом его изменении на работающем сайте. К сожалению, такая повторная компиляция практически одинакового кода снижает производительность. Отсюда понятно, почему кэширование промежуточного кода — OPCache — очень полезно.

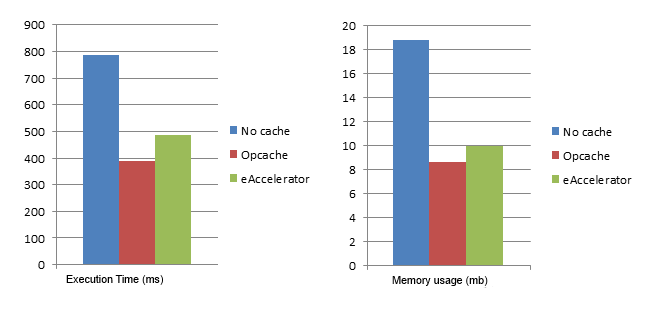
Zend OPCache — это расширение, которое сохраняет скомпилированный код в памяти. Это позволяет PHP в следующий раз при выполнении кода проверять разметки времени и размеры файла, чтобы определить, были ли части исходного файла изменены. Если таких изменений не было, то будет запущен сохраненный код.

Более подробную информацию можно получить в статье на Хабрахабре [«PHP Performance Series: Caching Techniques»](https://timeweb.com/go?url=https%3A%2F%2Fhabrahabr.ru%2Fpost%2F31446%2F&hash=3ad957d59aca051d9cce4072c70fa8cb3f018e89)

На рисунке 2 показано различие во времени выполнения и использовании памяти между PHP-приложением, выполняемым: 1) без кэша; 2) с OPcache; 3) с eAccelerator (другой инструмент PHP-кэширования).

Правый график показывает время выполнения в миллисекундах (Execution Time (ms)), левый — использование памяти в мегабайтах (Memory usage (mb)). Столбики синего цвета соответствуют отсутствию кэширования, красного — кэшированию OPcache, зеленого — кэшированию eAccelerator.

Из рисунка 2 следует, что кэширование OPcache снижает как время выполнения, так и использование памяти примерно в два раза по сравнению с отсутствием кэширования. Кэширование eAccelerator немного уступает кэшированию OPcache.



*Рис. 2. Столбиковые диаграммы различия во времени выполнения и использовании памяти между PHP-приложением, выполняемым без кэширования, с OPcache и с eAccelerator*

**2. Выявите задержки базы данных**

Как уже сказано выше, проблемы производительности не всегда связаны с кодом. Большинство узких мест возникает при обращении приложения к ресурсам. Обслуживание доступа к данным PHP-приложения может составлять до **90 процентов времени выполнения**. Поэтому в первую очередь необходимо проанализировать все случаи доступа к базе данных.

Удостоверьтесь, что лог медленных запросов SQL включен, чтобы иметь возможность выявить их. Затем изучите эти медленные запросы, чтобы оценить их эффективность. Если обнаружится, что выполняется слишком много запросов или одни и те же запросы необоснованно повторяются, внесите соответствующие изменения. Такие изменения должны повысить производительность приложения, сокращая время доступа к базе данных.

Как узнать, какие из запросов выполняются дольше всего? Более подробно см. статью на Хабрахабре «Как выявить медленные SQL запросы?», URL: https://habrahabr.ru/post/31072/

**3. Очистите файловую систему**

Проанализируйте файловую систему на неэффективность, то есть удостоверьтесь, что файловая система не используется для хранения сессий. Самое главное - внимательно следите за функциями статистики файлов: **file\_exists()**, **filesize()** и **filetime()**. Попадание этих функций внутрь цикла приводит к проблемам с производительностью.

**4. Тщательно контролируйте показ API**

Большинство веб-приложений, которые зависят от внешних ресурсов, используют удаленный API. Хотя удаленный API находится вне вашего контроля, однако можно смягчить проблемы API-производительности. Например, можно **кэшировать API-вывод или делать фоновые вызовы API**. Установите разумные интервалы для API-запросов и, если это возможно, показывайте на дисплее API-вывод без ответа API.

**5. Профилируйте PHP**

Использования OPcache и управления внешними ресурсами достаточно, чтобы большинство приложений выполнялись благополучно. Но если ваши потребности растут, пора профилировать PHP. Конечно, полное профилирование PHP-кода отнимает много времени, но оно дает всестороннюю информацию о производительности PHP-приложения. Имеются общедоступные программы для профилирования PHP-кода, такие, как Xdebug.

Xdebug рассматривается в статье на Хабрахабре [«Introducing xdebug»](https://timeweb.com/go?url=https%3A%2F%2Fhabrahabr.ru%2Fpost%2F31452%2F&hash=dbe9704fda006726a502176f74de07f39dc9d5ec)

**Необходимо уметь контролировать производительность PHP**

Веб-приложение может хорошо работать минуту, но внезапные проблемы с трафиком могут прервать его нормальную работу. Правда, к этому можно подготовиться. Понятно, что внесение изменений всегда требует времени, усилий и денег, и всегда трудно сказать, стоят ли того инвестиции. Лучший способ обосновать принятие решений — **постоянно собирать данные**.

Программное обеспечение по контролю производительности PHP немедленно оценивает эффекты любых вносимых изменений. Конечно, нужно знать, что именно оценивать. В этом плане скорость и использование памяти — лучшие индикаторы производительности, так как они влияют на время загрузки страницы, что весьма критично для веб-приложений.

Несмотря на то, что сбор данных важен, необходимо выключить систему контроля, когда она не нужна, поскольку поток логов замедляет выполнение приложения. С другой стороны, эти логи дают ценную информацию об улучшении производительности. Таким образом, необходимо периодически контролировать пиковые периоды трафика.

**Перспективы улучшения производительности PHP**

Эволюция PHP продолжается. Новейшее изменение в разработке PHP 8 — это добавление компиляции «на лету», или JIT-компиляции, которая позволит создавать еще более быстрые веб-приложения. Так как темп технологического прогресса растет, растут ожидания пользователей. Поэтому разработчики должны всегда внимательно следить за последними изменениями.

[Технология JIT-компиляции описана в Википедии.](https://timeweb.com/go?url=https%3A%2F%2Fru.wikipedia.org%2Fwiki%2FJIT-%25D0%25BA%25D0%25BE%25D0%25BC%25D0%25BF%25D0%25B8%25D0%25BB%25D1%258F%25D1%2586%25D0%25B8%25D1%258F&hash=180dd466a8182540ca0dae25c6ef15ed56a8618d)

Строя веб-приложения, помните, что сегодняшние приложения могут не работать следующем году. Вероятно, придется вносить изменения, чтобы обеспечить стабильную производительность PHP. Представление полной картины процесса развития — лучшая стратегия массового строительства PHP-приложений и веб-сайтов.