ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 240 за 2018 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Герман М.С., Ермаков А.В.

Блоковая структура кэширования web-страниц

Рекомендуемая форма библиографической ссылки: Герман М.С., Ермаков А.В. Блоковая структура кэширования web-страниц // Препринты ИПМ им. М.В.Келдыша. 2018. № 240. 17 с. doi:10.20948/prepr-2018-240
URL: http://library.keldysh.ru/preprint.asp?id=2018-240

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

М.С. Герман, А.В. Ермаков

Блоковая структура кэширования web-страниц

Москва — 2018

Герман М.С., Ермаков А.В.

Блоковая структура кэширования web-страниц

Предлагается реализация блоковой структуры кэширования страниц интернет-сайтов, рассматривается подход к организации кэширования динамически изменяемых частей этих страниц. Предлагаемый подход сопоставляется с известными решениями, заложенными в популярные системы управления сайтами CMS (Content management system). Описан пример ускорения построения веб-страницы за счёт распределённого хранения данных в оперативной памяти и на жёстком диске.

Ключевые слова: блоковая структура кэширования, CMS, веб-страница, динамическая часть.

Mikhail Sergeevich German, Alexey Viktorovich Ermakov

Block caching of web-pages structure

An implementation of the block structure of caching pages of Internet sites is proposed, an approach to organizing the caching of dynamically changeable parts of these pages is considered. The proposed approach is compared with well-known solutions embedded in popular content management systems (CMS). An example of accelerating the construction of a web page due to distributed data storage in RAM and on a hard disk is described.

Key words: block caching structure, CMS, web page, dynamic part.

Работа выполнена при поддержке Российского фонда фундаментальных исследований, Грант РФФИ № 16-01-00347-а, Грант РФФИ 18-07-00841-а, Грант РФФИ 18-07-01292-а.

Оглавление

Введение 3

Пример использования блокового кэширования 5

Апробация 11

Скорость загрузки 15

Заключение 16

Литература 17

Введение

Активное развитие интернета в последние десятилетия привело к быстрому развитию программ, позволяющих упростить разработку и создание сайтов. На рынке стали появляться не только новые языки веб-программирования, но и готовые наборы инструментов для быстрого создания сайта. Такие наборы инструментов называются Content management system (далее CMS). Назначение подобной системы – дать возможность разработчику (web-программисту) как можно проще и быстрее создать сайт, после чего останется только наполнить его контентом.

Рынок интернета развивается стремительно и, как следствие, потребности посетителей сайтов также растут. Интернет-сайты в настоящее время выполняют не только функции предоставления информации, но дают возможность организации обратной связи, электронного документооборота, оплаты услуг, бронирования номеров гостиниц, покупки билетов и т.д. Сайты стали богаче не только по функциональным возможностям, но и по дизайну – появились адаптивные версии вёрстки под планшетные устройства, мобильные телефоны, широкоформатные мониторы. Повышение сложности сайтов напрямую повлияло на развитие CMS – появилась необходимость разделения систем по направлениям, с различными наборами инструментов.

На практике использование CMS без дальнейших доработок удобно только для небольших сайтов, которые не требуют активного развития и адаптации под рынок Интернета, поскольку основное назначение CMS – быстрое создание сайта, без учёта возможного его дальнейшего изменения. Но CMS поддерживают техническую документацию, чтобы дальнейшее изменение готовой CMS не было столь сложной задачей.

Для различных направлений сайтов (сайты научных организаций, сайты коммерческих организаций, интернет-магазин и др.) требуется различный функционал. CMS обычно разделяют на обязательную неизменную часть для любых направлений сайтов (далее – ядро CMS) и необязательные части, которые могут отсутствовать или быть различными для направлений. Необязательные части могут как работать в рамках одной CMS (далее такие части будем называть модулями CMS), так и взаимодействовать с другими CMS и программами на веб-сервере (далее – инструменты). Оптимизация, введение новых открытых технологий и доработка найденных ошибок и уязвимостей в работе ядра CMS и модулей является правилом хорошего тона. На практике ядро CMS не нуждается в частом обновлении: любой сайт успешно работает и на изначально созданном ядре системы. Любые серьёзные изменения основного кода CMS могут повлиять на работоспособность всех модулей и инструментов, которые были установлены на сайтах до изменений.

Получается, при изменении ядра CMS (например, оптимизации, которая немного уменьшает нагрузку на веб-сервер или экономит место в памяти) необходимо обеспечить полную совместимость всех существующих к нему модулей, в том числе написанных сторонними разработчиками. Это трудозатратно, если, например, изменения в ядре были связаны с архитектурой. Следовательно, изменять работу ядра CMS просто невыгодно. Поэтому новые улучшения CMS, которые появлялись уже после начала эксплуатации какой-либо системы, не входили в состав ядра, а подключались как отдельные модули или инструменты.

Примером такого инструмента является Memcached [3], который обеспечивает кэширование в оперативную память (далее RAM). Основное назначение – по ключу положить данные в память и быстро их вернуть по соответствующему запросу. Этот инструмент был реализован Брэдом Фитцпатриком (Brad Fitzpatrick) в 2003 году в рамках работы над проектом сайта LiveJournal. Он использовал Memcached для разгрузки базы данных от запросов при формировании веб-страниц. В настоящее время его применяют во многих крупных проектах, например Wikipedia, YouTube, Facebook и другие.

Аналогичным инструментом является Redis [4], появившийся в 2009 году. Он имеет более расширенный функционал: поддерживает сложные типы данных (например, списки), поддерживает высокоуровневые операции (например, сортировка), имеет встроенную поддержку репликации данных на другие сервера с целью защитить данные и повысить скорость записи/чтения данных при нагрузке. К сожалению, этот инструмент не поддерживается ОС Windows. Существенным минусом Memcached и Redis является необходимость отслеживать актуальность данных и хранить информацию о ключах, по которым данные можно получить. При существенном масштабировании сайта это сильно усложняет сам процесс веб-разработки страниц.

Но хранение данных возможно не только в оперативной памяти, но и на жёстком диске. В отличие от оперативной памяти, жёсткий диск является наиболее доступным аппаратным решением в силу своей дешевизны. Поэтому для сайтов, требующих хранения большого количества данных, этот подход является более приоритетным.

В текущих решениях автоматическое кэширование страниц на жёстком диске имеет линейную зависимость от объёма хранимых данных. Чем больше страниц на сайте, тем больше данных сохраняется на жёстком диске. Это характерно для полностраничного кэширования страниц: веб-сервер вычисляет страницу, а потом её html-код сохраняется на жёсткий диск. При следующей загрузке страницы вычислений не происходит: отображается уже вычисленный ранее html-код.

Но для большинства сайтов части html-кода страниц повторяются и на других страницах. Для таких страниц необходимо эти части вычислять отдельно и сохранять как независимые от самой страницы части кэша. Это позволит нескольким различным страницам использовать уже вычисленный ранее кэш этих частей при компоновке странице в момент её запроса. Для сайтов, которые содержат повторяющиеся части html-кода на разных страницах, объём файлов кэша занимает меньше места на жёстком диске, чем при полностраничном кэшировании.

Второе важное преимущество этого подхода основано на том, что при разделении страницы на части каждая часть кэшируется отдельно по собственному алгоритму. Например, на одной и той же странице могут быть части, которые зависят и не зависят от авторизации клиента, просматривающего страницу.

Таким образом, нами разработан подход, позволяющий выделять статические и динамические части страниц сайтов. Статические части хранятся в кэше или на жестком диске. Динамические части также сохраняются, но автоматически перевычисляются, как только произошло изменение в этой части страницы.

Пример использования блокового кэширования

Рассмотрим детали предлагаемого подхода на примере сайта Института прикладной математики им. М.В. Келдыша РАН (http://www.keldysh.ru/) с учетом планов по его развитию и введению механизма авторизации для сотрудников.

Предположим, что шапка сайта неизменна, а меню отображает разные пункты в зависимости от текущего пользователя и может принимать различные значения (рис. 1).

Рис. 1. Пример разделения на части сайта http://www.keldysh.ru/.
Выделена часть №1, которая не зависит от текущей авторизации,
и часть №2, которая зависит.

В таком случае часть сайта, которая формирует отображение шапки сайта, должна храниться после вычисления как один файл кэша, поскольку имеет только одно возможное отображение. А часть № 2 сайта может принимать N+1 значений, где N – число зарегистрированных пользователей сайта и гость. То есть у части №2 может быть не более N+1 файлов кэша.

Если предположить, что меню сайта http://www.keldysh.ru/ имеет такую же структуру и на других страницах сайта, то меню уже не нужно будет вычислять, можно использовать уже имеющиеся файлы кэша.

Здесь следует сразу отметить, что в настоящий момент количество ежедневных посещений сайта не настолько значительно (около 2000), чтобы катастрофически перегрузить имеющиеся ресурсы. Т.е. вышеприведенные рассуждения для нас интересны именно как пример использования подхода для

научного сайта (ниже будет показан пример коммерческого сайта, в котором динамика страниц высока и объем сетевого трафика уже значителен).

При вышеописанном подходе не важно назначение сайта: научный, коммерческий или промышленный – это влияет на содержимое, но не на алгоритм блочного кэширования. Например, html-блоком может быть таблица, содержащая изображения, текст, ссылки, кнопки и др. элементы html. На рис. 2 показана реализация указанного подхода на сайте https://whoplay.ru. Аналогичная таблица была бы и на сайтах другой тематики – кэширование будет работать по тому же алгоритму во всех случаях и с любыми данными (рис. 3).

№	id Роли	steam language	steam avatar	Логин	BattleNET img			
0	Админ SteamRole	RU						
1	Админ SteamRole							
2				german1748				
3	SteamRole	RU		icegull				
4				icegull2887				

Рис. 2. Пример блока с таблицей пользователей.

№	URL	Разрешено пользователям	Разрешено ролям	Запрещено ролям	Доступ авторизованным пользователям	Доступ не авторизованным посетителям	Доступ по-умолчанию	
0	/						+	
1	/lk				+	-	-	
3	/control/	German	Админ		-	-	-	
5	/login		SteamRole	Админ	-	+	+	

Рис. 3. Пример блока с таблицей доступа.

Актуальность этого подхода основана на том, что большинство сайтов в интернете имеет хотя бы один повторяющийся блок на разных страницах. Чтобы понять, как стоит делить страницы на блоки, необходимо задать условия их формирования, которые задаются веб-программистом. Поэтому функция разделения страницы на части, которые кэшируются отдельно, возложена на

разработчика сайта, который знает, что, например, меню неизменно, а блок авторизации зависит от текущего пользователя.

Чтобы в момент запроса страницы понимать, какой файл кэша части отображаемой страницы содержит актуальную информацию, а какой нет, необходимо отслеживать изменения данных, которые влияют на формирование этой части. Распространённым решением среди CMS является удаление файлов кэша через равные временные промежутки. Хотя такая реализация самая простая, она не является наиболее эффективной. В этом случае происходят излишние повторные вычисления частей страниц, которые строятся на основе не менявшихся данных. Происходит не только излишняя трата вычислительных мощностей, но и излишние операции записи на жёсткий диск.

В разработанном нами подходе предлагается удалять файл кеша только в случае изменения данных. При следующем запросе к странице, содержащей эту html-часть и при отсутствии файла кэша, часть будет вычислена заново, уже на основе новых данных. Таким образом, при каждом запросе будет использоваться только актуальная информация.

Чтобы реализовать отслеживание изменения данных, необходимо каждую операцию чтения, записи, удаления или обновления данных логгировать и запоминать, какая часть использует какие данные. Если при формировании html-части происходит чтение данных, система сохраняет данные в подобном формате:

- ссылка на файл кэша html-части,
- строка подключения к базе данных,
- таблица в базе данных.

Если в процессе работы сайта происходит удаление, обновление или создание записи в таблицу базы данных, то после выполнения операции удаляются все файлы кэша, ссылки которых были сохранены в системе ранее. Аналогично с кэшем на жёстком диске необходимо обновить данные и в оперативной памяти, если она также используется.

Поскольку на html-часть может влиять множество параметров (такие как идентификатор пользователя), необходимо, чтобы они влияли на url-ссылку файла кэша. В нашем подходе были выделены следующие параметры, которые могут влиять на отображение html-части:

- url-запрос: на разных страницах сайта html-часть может принимать разные значения,
- тип запроса (GET или POST),
- cookies-файлы (например, если сайт хранит данные о поведении пользователя в cookies и на веб-странице надо реализовать раздел «Вы смотрели ранее» на основании этих данных),

- текстовый идентификатор, позволяющий различным динамическим частям использовать общий кэш. Если у двух разных html-частей будет одинаковый идентификатор, то место хранения в памяти жёсткого диска будет одинаковым при одинаковых параметрах загрузки. Это позволяет вычислять html-блоки один раз, после чего использовать уже вычисленный кэш,
- уникальный идентификатор посетителя, например логин. Позволяет кэшировать индивидуальное для каждого посетителя состояние html-частей.

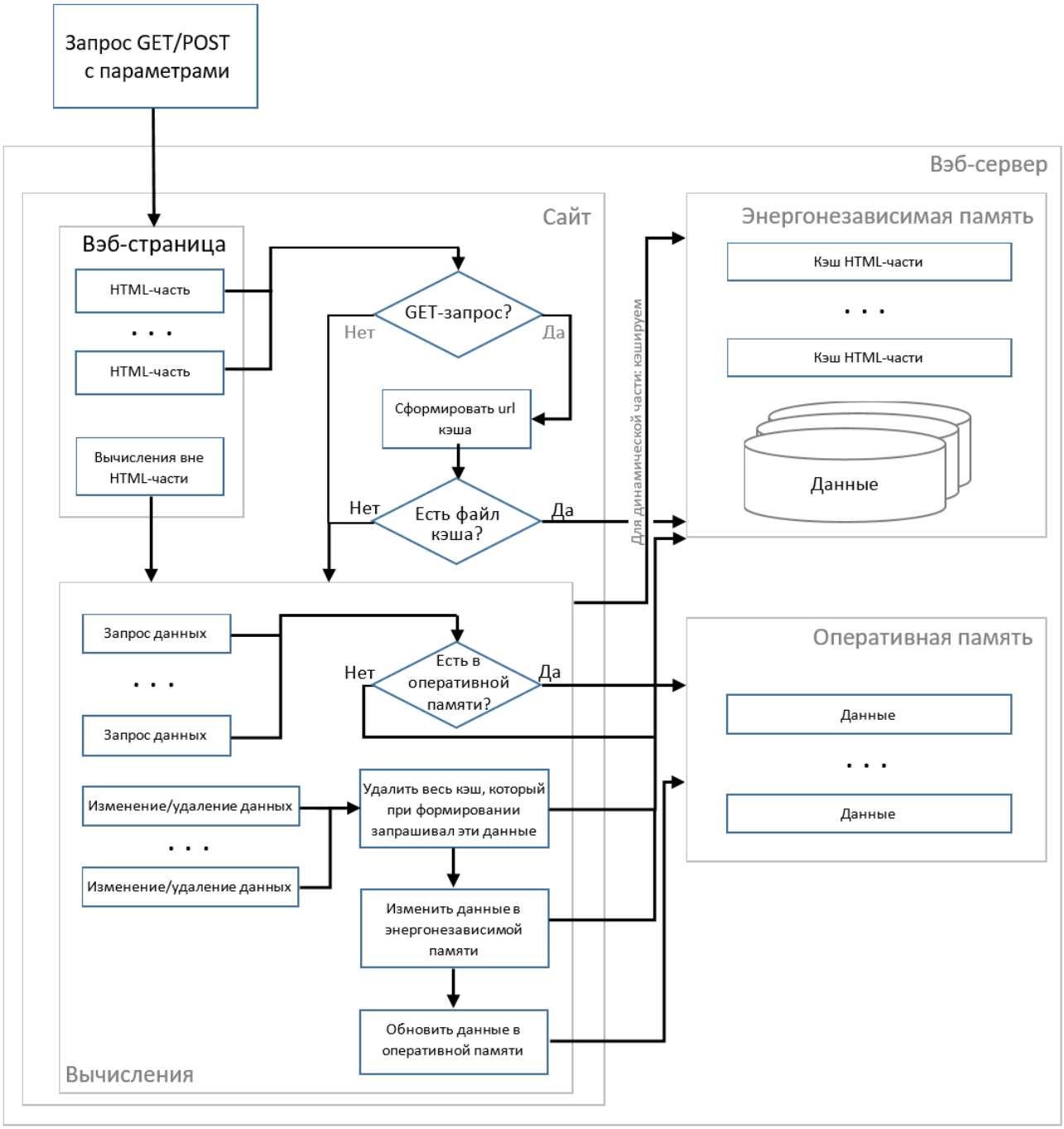

Рис. 4. Схема кэширования HTML-частей.

Рассмотрим подробнее, от чего может меняться состояние html-частей (далее – параметры загрузки):

- url-запросы,
- тип запроса (GET или POST),
- cookies-файлы (например, если сайт хранит данные о поведении пользователя в cookies и на веб-странице надо реализовать раздел «Вы смотрели ранее» на основании этих данных),
- текстовый идентификатор, позволяющий различным динамическим частям использовать общий кэш. Если у двух разных динамических частей будет одинаковый идентификатор, то место хранения в памяти жёсткого диска будет одинаковым при одинаковых параметрах загрузки. Это позволяет вычислять динамические блоки один раз, после чего использовать уже вычисленный кэш,
- уникальный идентификатор посетителя, например логин. Позволяет кэшировать индивидуальное для каждого посетителя состояние динамических частей.

В рамках данной разработки всю логику html-блока решено выделить в отдельный класс (далее – модуль). Каждый модуль содержит информацию о том, какие факторы влияют на его формирование. Результатом вычисления модуля является список из следующих данных:

- HTML код,
- javascript-код,
- javascript-код, который запускается по событию onload при загрузке страницы,
- CSS стиль,
- POST ответ в виде текстовой строки,
- дополнительные данные, которые вычисляются внутри модуля и необходимы вне его. Например, если модуль вычисляет наименование категории продукта по url-ссылке, то стоит вернуть наименование для формирования «хлебных крошек» (навигация web-страницы) вне модуля.

Модуль может не просто отображать данные, но и, благодаря обработчику POST-запросов, может отправлять запросы к веб-серверу и получать ответы без перезагрузки страницы (технология AJAX [5]). Если данные, передаваемые с помощью AJAX, не зависят от параметров загрузки и меняют отображение модуля, то кэш модуля будет неизменным, ведь изначальный код страницы не изменился. Например, если модуль отображает динамичную файловую структуру: слева список папок, а при клике на папку появляются вложенные в папку файлы (рис. 5).

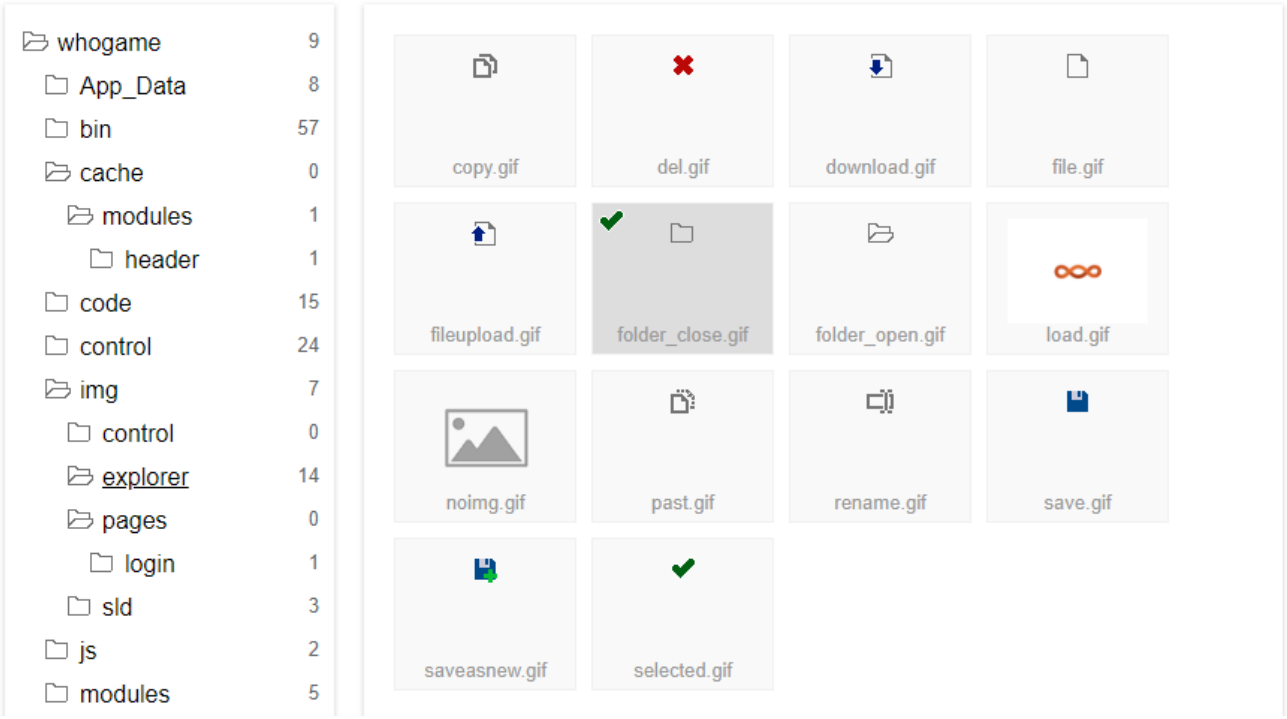

Рис. 5. Модуль, отображающий файловую систему.

В итоге, если состояния модулей для определённых параметров запроса уже были вычислены ранее, скорость загрузки страницы будет максимально быстрой, поскольку не потребуется никаких вычислений.

Апробация

На основе предложенных подходов был разработан коммерческий сайт http://chik-chik.ru/, на примере которого, как более динамичного, чем научный, далее рассмотрим подробно работу предложенного подхода (на основе раздела доставки). Этот раздел состоит из нескольких страниц:

1. http://chik-chik.ru/delivery
   Страница описания способов доставки заказа.
2. http://chik-chik.ru/delivery/pickup
   Страница доставки заказа с помощью самовывоза.
3. http://chik-chik.ru/delivery/pickup/moskva
   Пример доставки самовывоза в один из доступных городов (в данном случае г. Москва).
4. http://chik-chik.ru/delivery/pickup/moskva/7
   Пример страницы точки самовывоза в г. Москва.

Каждая из указанных страниц содержит:

1. html-часть, представляющую верхнее меню: отображается одинаково, вне зависимости от параметров загрузки, поэтому имеет только одно

состояние, а кэш хранится по адресу «/cache/modules/menu_horizont.txt» (рис. 6);

2. html-часть, представляющую нижнее меню, – также отображается вне зависимости от параметров загрузки, поэтому, как и предыдущая часть, имеет только одно состояние, а кэш хранится по адресу «/cache/modules/underground.txt» (рис. 6);
3. html-часть, отвечающую за содержание на странице информации о самой доставке.

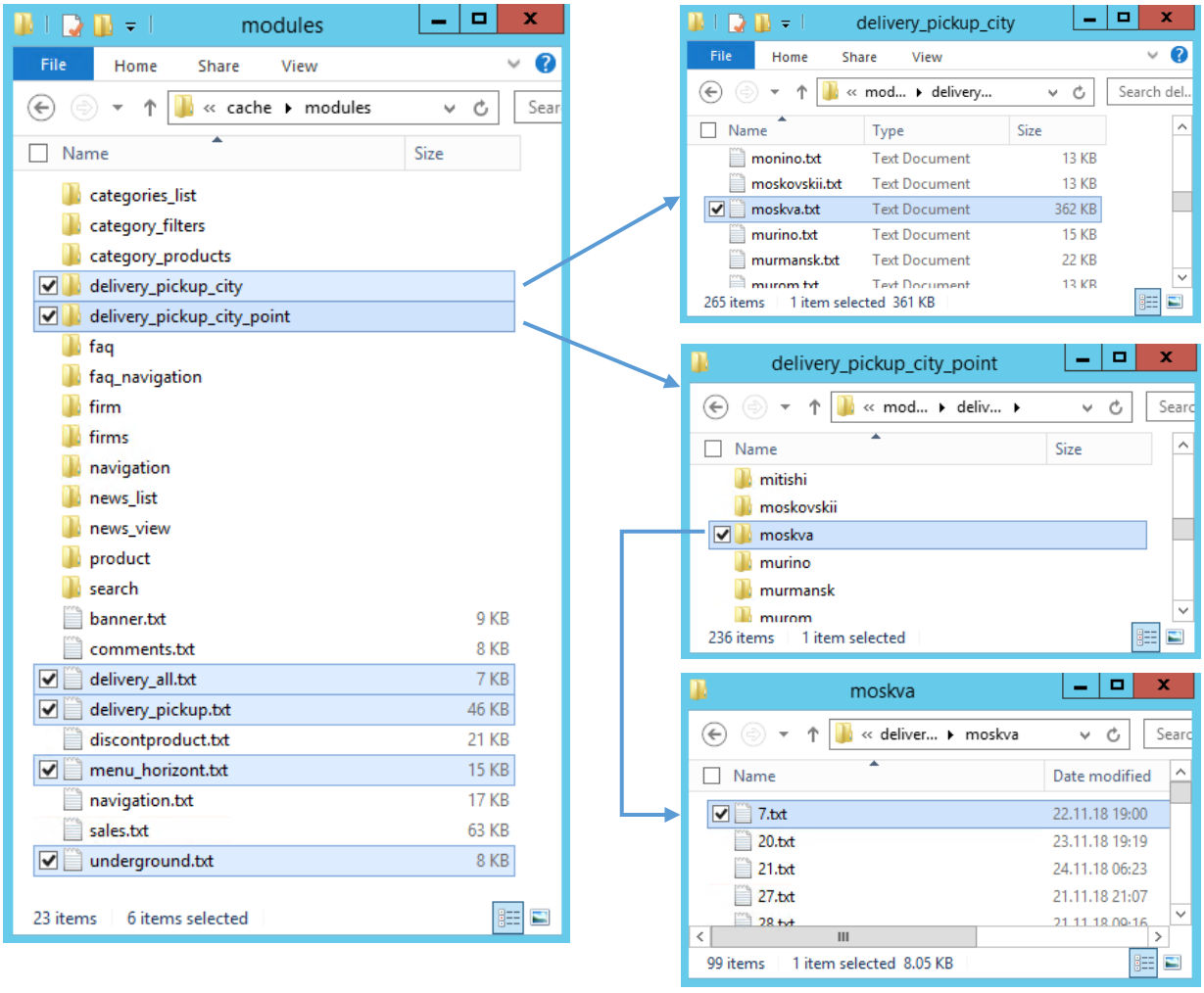

Рис. 6. Хранения файлов кэша для страниц доставки на сайте http://chik-chik.ru/.

Html-части, которые формируют на рассматриваемых страницах уникальную информацию о доставке товаров, формируются с помощью модулей.

Для «/delivery» (рис. 7) – это модуль «delivery_all», который всегда принимает одно и то же состояние и хранит кэш по адресу «/cache/modules/delivery_all.txt». Модуль при формировании запрашивает

данные о способах доставки и формирует список способов доставки, стоимость (если она фиксированная), расписание доставки и другие параметры при их наличии. В самом файле кэша хранится вся необходимая информация для воспроизведения данных на странице без необходимости повторного формирования данных: css-стили, javascript-код (в том числе тот, который срабатывает при событии onload), html-код страницы, дополнительные данные, которые могли быть вычислены внутри модуля (в нашем случае таких данных не было).

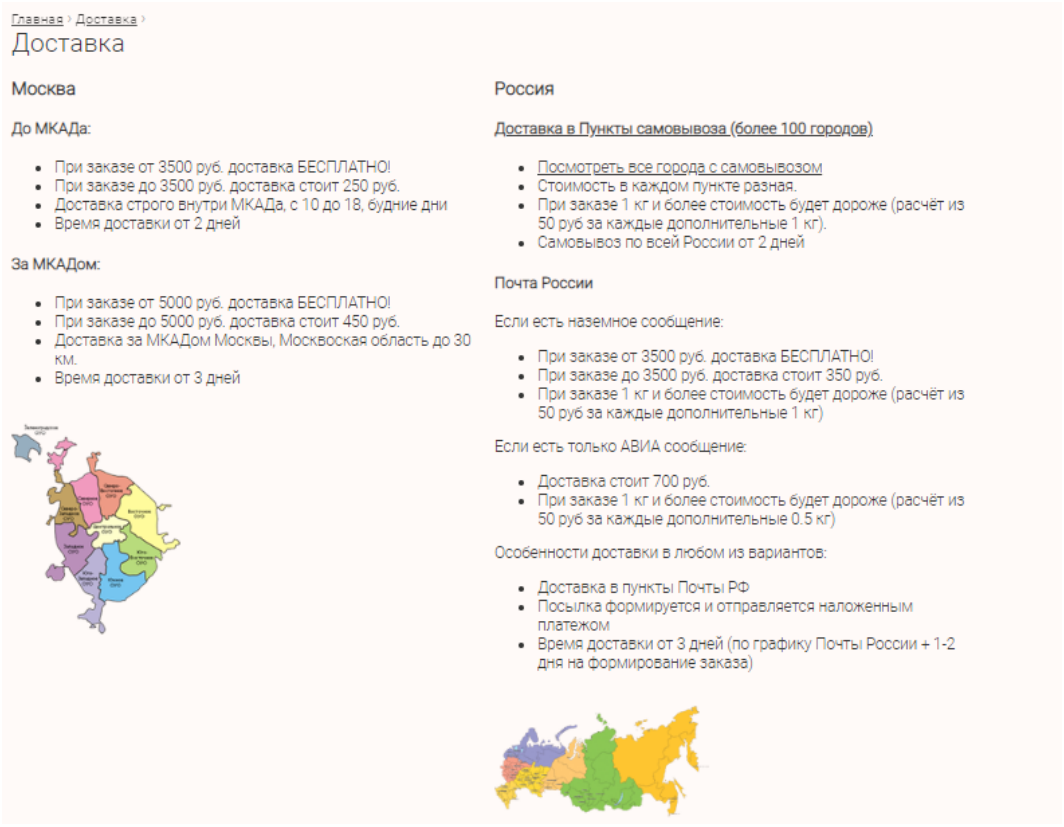

Рис. 7. Визуальное отображение http://chik-chik.ru/delivery/pickup.

Страница «/delivery/pickup» содержит список всех городов, упорядоченных по алфавиту. Необходимо учитывать, что этот список сформирован на основе данных, которые предоставляет сторонняя транспортная кампания. Раз в сутки в 04:00 часов ночи по МСК времени сайт обновляет данные по всем точкам самовывоза. Список городов может как расширяться, так и сокращаться. Если список городов был изменён, то кэш модуля, отвечающий за формирование этой страницы, удаляется и при следующей загрузке посетителю будет показан уже актуальный список городов для доставки заказа.

Рассмотрим самый многочисленный город по точкам самовывоза – г. Москва с адресом «/delivery/pickup/moskva». Город содержит 187 точек самовывоза, причём данные на сайте могут меняться ежесуточно. Это не только открытие или закрытие точки: даже если меняется расписание работы пункта самовывоза в предпраздничные дни или пункт приёма добавил способ оплаты (например, появилась возможность бесконтактной оплаты банковской картой), данные нужно обновлять. Каждый пункт может содержать информацию: полный адрес, контактные телефоны, график работы, сроки доставки заказа в этот пункт, работает ли пункт с банковскими картами или нет, подробно описание проезда, координаты, стоимость доставки и другие данные, которые сайт получает от службы доставки. Наш подход позволяет для любого посетителя отображать уже готовую страницу, если уже данную страницу кто-то запрашивал ранее и существует её актуальный кэш (для города Москвы это «/cache/modules/delivery_pickup_city/moskva.txt»).

Модуль, отвечающий за формирование контента страницы о доставке в г. Москва, помимо css, javascript и html-кода, также возвращает дополнительную информацию для формирования «хлебных крошек». Поскольку «хлебные крошки» присутствуют не только на страницах с информацией о доставке, то необходимо эту часть кода вынести в отдельный блок. В нашем случае часть «хлебных крошек» (наименование города) вычисляется в одном из модулей страницы. Поэтому, чтобы не вычислять ту же информацию повторно, этот модуль возвращает наименование города: «Главная › Доставка › Самовывоз › Москва › ».

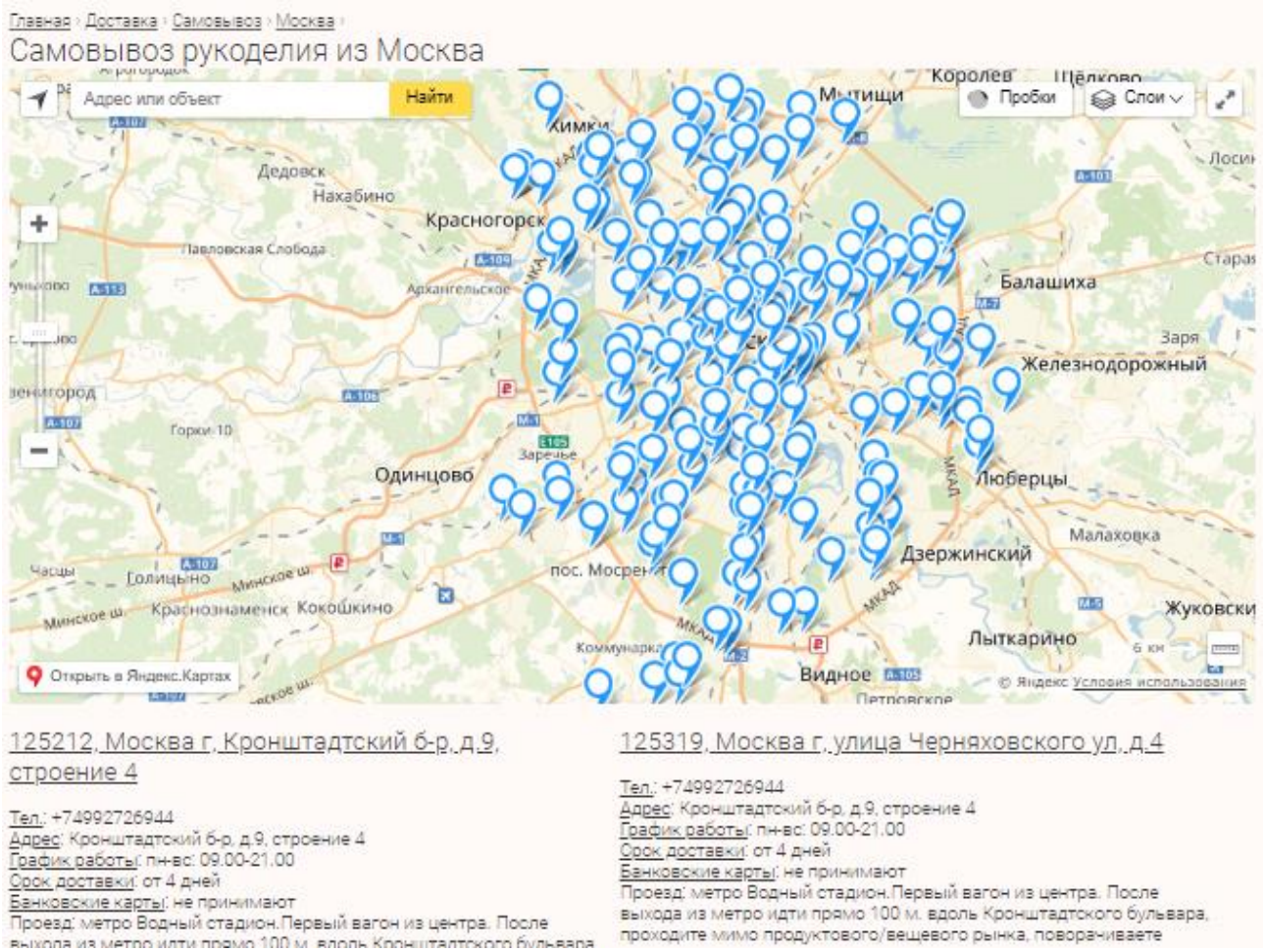

Рис. 8. Часть страницы точек самовывоза в г. Москва

Страница «/delivery/pickup/moskva/7» содержит информацию об одном из пунктов самовывоза в г. Москва. Если пункт был закрыт, то модуль, вычисляющий контент страницы об этом пункте, вернёт текст «Пункт не найден», и, как следствие, в файл кэша будет записана информация об отсутствии пункта самовывоза для этого url-запроса. При следующем запросе уже не будет производиться проверка существования этого пункта, поскольку будет существовать файл кэша с этой информацией.

Скорость загрузки

Сайт http://chik-chik.ru/ показал в среднем хорошие результаты в нагрузочном тестировании по сравнению с другими крупными сайтами (см. табл. №1). Для тестирования мы выбирали сервера, близкие к физическому расположению веб-сервера сайта.

	min, мс	max, мс	**avg, мс**	size, мб	Html/js/img %
chik-chik.ru	170	6 930	**231**	0.7 мб	11 / 12 / 74 %
amazon.com	28	999	**316**	6.7 мб	2 / 23 / 70 %
ozon.ru	1 040	24 410	**1 440**	2.7 мб	1 / 35 / 53 %
yandex.ru	599	1 760	**717**	0.9 мб	10 / 32 / 15 %

Таблица №1. Скорость загрузки страниц по данным Loadimpact [6].

В таблице указана минимальная (столбец min) и максимальная (столбец max) скорость загрузки страницы, но их нельзя считать объективным показателем для сравнения. В данном случае стоит считать среднее значение (столбец avg) на всём периоде тестирования, которые уже предоставляет Loadimpact. Стоит также обратить внимание на специфику самих сайтов: тестируемый прототип является интернет-магазином и на главной странице подгружает определённые товары по определённому алгоритму. Но, в отличие от аналогичных страниц у Amazon.com (крупнейший интернет-магазин в мире) и Ozon.ru (крупнейший интернет-магазин в России), размер передаваемых данных у chik-chik.ru в разы меньше. Поэтому пришлось добавить Yandex.ru (популярнейший поисковик в России) с похожим размером страницы для чистоты эксперимента. Прототип, реализованный с помощью предложенного подхода, стал лидером по средней скорости загрузки.

Заключение

В данной работе был предложен новый подход к работе сайтов с разработанным блочным кэшированием. Был разработан первый прототип на основе предложенного подхода, в настоящее время готовится вторая версия. На основе разработанного прототипа были получены положительные результаты нагрузочного тестирования сайта по сравнению с такими известными сайтами, как Amazon.com, Ozon.ru и Yandex.ru.

Программа для ЭВМ “CMS с автоматическим кэшированием данных” зарегистрирована Федеральной службой по интеллектуальной собственности в реестре программ для ЭВМ, в результате чего выдано свидетельство о государственной регистрации программы для ЭВМ № 2018660021 от 20 июля 2018 г.

Литература

1. CMS для электронной коммерции. — URL: https://www.magento.com/
2. CMS общего назначения. — URL: https://www.1c-bitrix.ru/products/cms/
3. Сервис кэширования данных в оперативной памяти Memcached. URL: https://memcached.org/
4. Хранилище данных типа «ключ — значение» Redis. — URL: https://redis.io/
5. AJAX - Подход к построению интерактивных пользовательских интерфейсов веб-приложений . URL: https://www.w3.org/TR/XMLHttpRequest/
6. Инструмент для нагрузочного тестирования сайтов. URL: https://loadimpact.com/