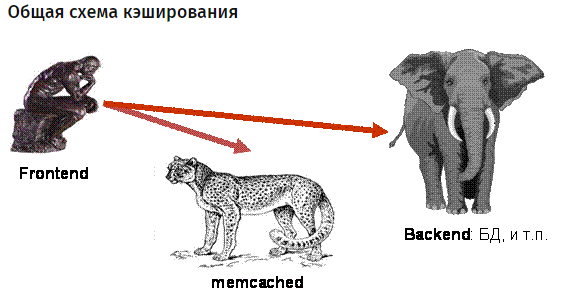
[Memcached](http://danga.com/memcached/) представляет собой огромную хэш-таблицу (КЛЮЧ: ЗНАЧЕНИЕ) **в оперативной памяти**, доступную по сетевому протоколу. Он обеспечивает сервис по хранению значений, ассоциированных с ключами. Доступ к хэшу мы получаем через простой сетевой протокол, клиентом может выступать программа, написанная на произвольном языке программирования (существуют клиенты для C/C++, PHP, Perl, Java и т.п.).  
  
Самые простые операции – получить значение указанного ключа (get), установить значение ключа (set) и удалить ключ (del). Для реализации цепочки атомарных операций (при условии конкурентного доступа к memcached со стороны параллельных процессов) используются дополнительные операции: инкремент/декремент значения ключа (incr/decr), дописать данные к значению ключа в начало или в конец (append/prepend), атомарная связка получения/установки значения (gets/cas) и другие.  
  
Memcached был реализован Брэдом Фитцпатриком (Brad Fitzpatrick) в рамках работы над проектом ЖЖ (LiveJournal). Он использовался для разгрузки базы данных от запросов при отдаче контента страниц. Сегодня memcached нашел своё применение в ядре многих крупных проектов, например, Wikipedia, YouTube, Facebook и другие.



В общем случае схема кэширования выглядит следующим образом: frontend’у (той части проекта, которая формирует ответ пользователю) требуется получить данные какой-то выборки. Frontend обращается к быстрому как гепард серверу memcached за кэшом выборки (get-запрос). Если соответствующий ключ будет обнаружен, работа на этом заканчивается. В противном случае следует обращение к тяжелому, неповоротливому, но мощному (как слон) backend’у, в роли которого чаще всего выступает база данных. Полученный результат сразу же записывается в memcached в качестве кэша (set-запрос). При этом обычно для ключа задается максимальное время жизни (срок годности), который соответствует моменту сброса кэша.  
  
Такая стандартная схема кэширования реализуется всегда. Вместо memcached в некоторых проектах могут использоваться локальные файлы, иные способы хранения (другая БД, кэш PHP-акселератора и т.п.) Однако, как будет показано далее, в высоконагруженном проекте данная схема может работать не самым эффективным образом. Тем не менее, в нашем дальнейшем рассказе мы будем опираться именно на эту схему.

Потеря ключей

Memcached не является надежным хранилищем – возможна ситуация, когда ключ будет удален из кэша раньше окончания его срока жизни. Архитектура проекта должна быть готова к такой ситуации и должна гибко реагировать на потерю ключей. Можно выделить три основных причины потери ключей:

1. Ключ был удален раньше окончания его срока годности в силу нехватки памяти под хранение значений других ключей. Memcached использует политику LRU, поэтому такая потеря означает, что данный ключ редко использовался и память кэша освобождается для хранения более популярных ключей.
2. Ключ был удален, так как истекло его время жизни. Такая ситуация строго говоря не является потерей, так как мы сами ограничили время жизни ключа, но для клиентского по отношению к memcached кода такая потеря неотличима от других случаев – при обращении к memcached мы получаем ответ «такого ключа нет».
3. Самой неприятной ситуацией является крах процесса memcached или сервера, на котором он расположен. В этой ситуации мы теряем все ключи, которые хранились в кэше. Несколько сгладить последствия позволяет кластерная организация: множество серверов memcached, по которым «размазаны» ключи проекта: так последствия краха одного кэша будут менее заметны.

Все описанные ситуации необходимо иметь в виду при разработке программного обеспечения, работающего с memcached. Можно разделить данные, которые мы храним в memcached, по степени критичности их потери.  
  
**«Можно потерять»**. К этой категории относятся кэши выборок из базы данных. Потеря таких ключей не так страшна, потому что мы можем легко восстановить их значения, обратившись заново к backend’у. Однако частые потери кэшей приводят к излишним обращениям к БД.  
  
**«Не хотелось бы потерять»**. Здесь можно упомянуть счетчики посетителей сайта, просмотров ресурсов и т.п. Хоть и восстановить эти значения иногда напрямую невозможно, но значения этих ключей имеют ограниченный по времени смысл: через несколько минут их значение уже неактуально, и будет рассчитано новое значение.  
  
**«Совсем не должны терять»**. Memcached удобен для хранения сессий пользователей – все сессии равнодоступны со всех серверов, входящих в кластер frontend’ов. Так вот содержимое сессий не хотелось бы терять никогда – иначе пользователей на сайте будет «разлогинивать». Как попытаться избежать? Можно дублировать ключи сессий на нескольких серверах memcached из кластера, так вероятность потери снижается.