

[К списку статей](#)

JMeter — швейцарский нож тестировщика (Часть 1)

Александр Пермяков, QA-специалист

10.07.2019

Поговорим о JMeter'е? Скорее всего, если вы в теме, то давно уже прочли всё об этом инструменте для нагрузочного тестирования. Но мне есть чем вас удивить. За три года работы в QA я понял, что JMeter очень универсален и может использоваться в самых разных целях. Например, для:

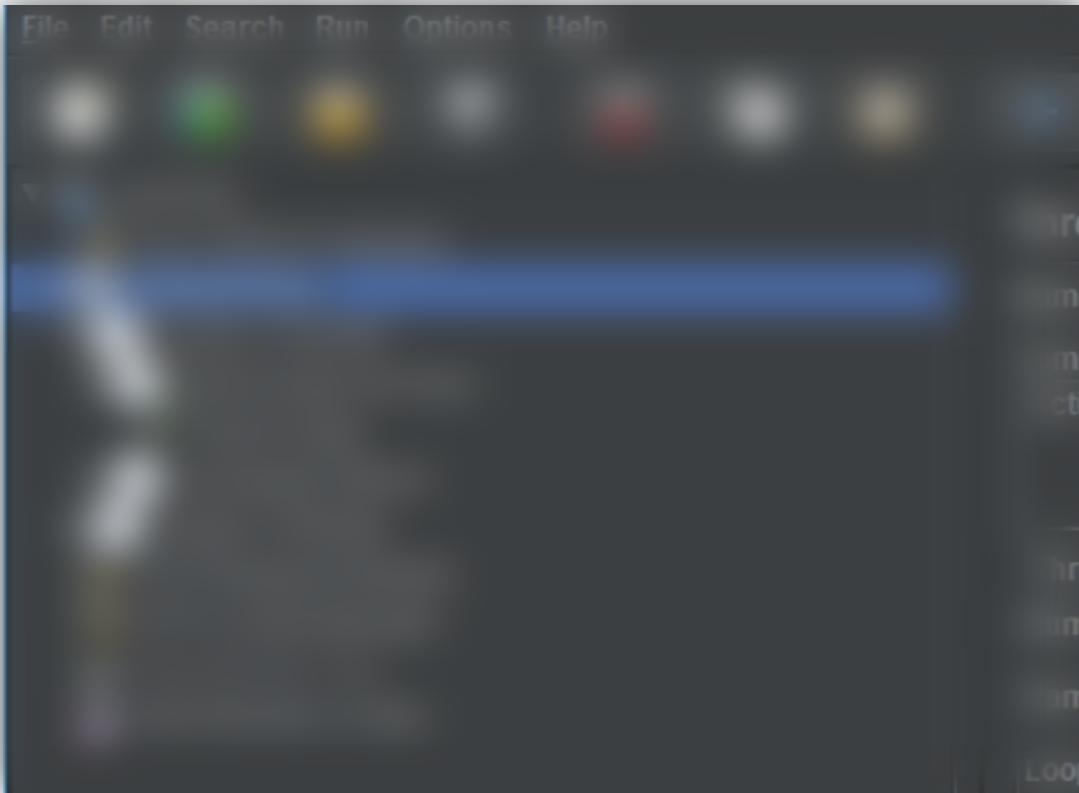
- поиска неуловимого бага с падением сайта у части юзеров;
- быстрого прогрева кэша на тысячах страниц продуктов;
- тестирования бэкенда мобильного приложения;
- создания 2000 записей с данными пользователей в базе приложения в сжатые сроки.

Получилось объемно, поэтому разобью статью на две части.

Disclaimer: По понятным причинам я не вставляю в статью скриншоты из реальных проектов (или вымарываю из них всю важную информацию). Иллюстрации представлены исключительно в исследовательских и ознакомительных целях.

fuse8**СТАТЬИ**

JMeter — Java-апплет с GUI. Для тестирования он запускается без графического интерфейса. А для написания тестовых скриптов у него панель, в которой можно сделать всё что угодно.



Так выглядит процесс создания скрипта (тут даже слово «написания» не подойдет)

Создается общий Test Plan, в который закидывается Thread Group с основными элементами теста: контроллеры, управляющие процессом, и запросы (HTTP, FTP и т.д.).

Помимо этого есть дополнительные элементы для задания параметров. Например, HTTP Request Defaults позволяет указать основной сервер, заголовки и включить/выключить загрузку дополнительных ассетов (картинок, стилей, шрифтов и прочего). Разобраться несложно. А ещё можно сразу из этого интерфейса запустить тест и посмотреть на результаты.

JMeter умеет записывать такие тестовые сценарии. Он запускается в виде прокси на локальной машине и, если вы настроите браузер (или приложение), вести трафик через эту проксию, JMeter запишет все запросы и ответы. А из этого набора можно состряпать тестовый сценарий, который будет повторять действия пользователя, и запустить его где и когда угодно:

Основная проблема — heap памяти самой Java. Он упирается в потолок и может упасть при 50 потоках одновременного тестирования даже на топовых машинах.

Если для проведения полного нагрузочного тестирования мощности машины не хватает — обратитесь к сторонним организациям. Они развернули у себя инфраструктуру, являющуюся набором серверов. На них поднят тот же самый JMeter. За вознаграждение эти ребята запустят ваш скрипт у себя на любой нагрузке. Мы обращались за такими услугами в Octoperf и Blazemeter.

Это футбол, детка: как мы ловили баг с частичным падением сервера

fuse8 СТАТЬИ

крупные появились недавно. Например, мы сделали сайт на Sitecore для одного из футбольных клубов английской Премьер-лиги с парадигмой SPW (single page website: все страницы открываются без перезагрузки самой страницы браузера). Под капотом — админка для управления страницей живого матча. Это текстовая онлайн-трансляция игры: основные события (голы, удаления, штрафные удары) подгружаются автоматически из системы Opta, а фото, комментарии и репосты болельщиков из Twitter и Instagram публикуют живые люди — контент-менеджеры футбольного клуба.

Не без гордости отмечу: после релиза британские СМИ выпустили статьи под заголовками вроде «Наконец-то гегемония старых сайтов футбольных клубов закончена». На тот момент у большинства клубов уже были сайты. Но выглядели они так, словно разрабатывались в начале нулевых и с тех пор не менялись — с соответствующим дизайном и UX.

Перед запуском мы провели полное нагрузочное тестирование и убедились, что всё работает как надо. Структура серверов выглядела так:

1. запрос от клиента идёт на сервер распределения нагрузки →
2. сервер распределения нагрузки проверяет состояние восьми CD-серверов →
3. сервер распределения нагрузки направляет запрос на наименее нагруженный CD-сервер →
4. CD-сервер отвечает клиенту.

Через год после запуска во время большого наплыва юзеров на сайт клиент позвонил и сообщил, что сайт не работает:

— У нас сайт не работает! Вообще не работает! Просто белый экран и надпись от браузера, что сайт не работает! — говорит клиент.

Мы в панике открываем сайт, а с ним все OK:

— Всё же работает, — отвечаем.

Клиент открывает сайт с телефона и компьютера и...у него тоже все OK!

— Действительно, извините. Видимо, у юзеров проблемы.

Такие сообщения на пустом месте не появляются, поэтому мы провели исследование: запустили утилиту для отслеживания состояния серверов Server Density и посмотрели, что происходит с ними во время наплыва пользователей на сайте:

Нагрузка есть, но все CD-сервера выглядят примерно одинаково и ровно

Скоро история повторилась — часть пользователей сообщили о полностью неработающем сайте. Это была не какая-то ошибка или ненайденная страница — сервер просто не отвечал на запрос. Отловить ситуацию получилось с помощью **JMeter'a**.

Цель была простой — нагрузить все восемь серверов и посмотреть, что произойдёт. Стресстест проводили, когда в Челябинске занималась заря, а в Лондоне была глубокая ночь. Мы использовали несколько машин в безинтерфейсном режиме (он позволяет запускать гораздо больше потоков одновременно). Скрипт бесконечно открывал домашнюю страницу, и в этом и была наша основная ошибка.

Мы загружали одни и те же ассеты, которые уже сто раз закэшировались. Поэтому и нагрузка выходила незначительной. Тогда пришла идея: на сайте — вагон и маленькая тележка страниц с новостями за определённые даты. Если зарандомить пару переменных и подставить их в URL, мы всё время будем получать случайную страницу (например — `...url/news/2016/08/23/?page=4`).

Внезапно мы осознали, что Server Density имеет некое встроенное «сглаживание» в графике нагрузки на сервер за прошлые периоды. Если за пять минут нагрузка на сервер достигла 100%, потом упала до 0%, а через 20-30 секунд выросла до 90%, он покажет среднее значение за этот период — около 80-90%. Поэтому мы и не видели реального падения серверов.

Детально разобрав логи во время стресс-тестов, мы выяснили, что один из CD-серверов при стопроцентной загрузке уходил в ребут и никому об этом не сообщал (вот такой он интроверт).

Сервер распределения нагрузки смотрел только на загрузку CPU всех серверов. Он видел, что один из них загружен на 20%, а остальные — под 90%, и отправлял юзера на первый. А он находился в ребуте и выдавал белый экран. Кроме того, сервер распределения выставлял юзеру куку и намертво привязывал его к неработающему серверу. Поэтому даже после обновления сайт был недоступен. Остальные 7 из 8 юзеров при этом радовались жизни и говорили, что все работает.

Вывод: мы выяснили, что JMeter можно использовать для стресс-тестов и одновременного анализа состояния серверов во время тестирования другими инструментами. Нам удалось «поймать» проблему с падением сайта у части пользователей и решить её, поправив логику распределения нагрузки и добавив контроль состояния.

В следующей части расскажу, как мы с помощью JMeter'а наладили процесс кэширования продуктовых страниц, проверили работу мобильного приложения без самого приложения и создали 2000 юзеров в системе без доступа к базе данных.

Статья на английском языке — в [блоге Delete Agency](#).

[VKontakte](#)[Facebook](#)[Twitter](#)

03.12.2020

Владимир Лысов

Как избежать race condition при использовании JWT-авторизации с помощью Axios

В статье рассмотрим авторизацию с помощью токенов, и как управляться с ними на фронте. Узнаем о способах обновления токенов, проблемах при обновлении и способах их решения.

Как прикрутить авторизацию в ASP.NET Core 3.1 через ASP.NET Core Identity и JWT — пошаговая инструкция.

«Цессионарий»: система для работы с просроченным задолженством по кредитам

Веб-разработка # Автоматизация бизнеса # Финансовые платформы

Редизайн страницы товара в интернет-магазине forsarm.ru

Веб-разработка # Веб-дизайн

© 2001-2021 ООО «Фьюз Эйт Онлайн»

📞 +7 (351) 903-99-81 +7 (351) 225-18-74 📩 info@fuse8.ru



Положение о конфиденциальности персональных данных