**HTTP/2 & SERVLET 4.0**

1. **Тема**

HTTP один из самых широко используемых протоколов в мире. И сегодня я хочу рассказать:

* о недостатках НТТР 1.1 и зачем было вводить новый протокол
* о новой версии НТТР – 2.0
* какие НТТР 2.0 дает преимущества в сравнении с предыдущей версией
* и как новый протокол повлиял на выпуск свежей версии SERVLET 4.0.

1. **История**

HTTP 0.9 (Hyper Text Transfer Protocol) – опубликован в далеком 1991 году. Начиналось как простой однострочный протокол для получения гипертекста (“GET /resource” – Телнет к google.com на 80 порт). Быстро развилось в основной протокол доставки гипертекста.

Ныне, самой распространенной версией протокола НТТР является версия 1.1, опубликованная в 1999 году (18 лет тому).

Первый масштабный проект реформирования HTTP был представлен в 2009 году инженерами Google – это протокол SPDY. Основная цель – ускорение работы веб-сайтов.

В 2015 году опубликована RFC нового протокола НТТР/2 – № 7540.

1. **SPDY**

SPDY представлен шестью годами ранее HTTP/2, во многом основывается на SPDY.

В основу первого черновика HTTP 2.0 положен SPDY/3

Тесты протокола SPDY показали:

* в ряде случаев страницы загружались на 55% быстрее, чем при использовании HTTP
* в среднем время загрузки сократилось на 36%

1. **Поддержка HTTP 2.0 браузерами**

HTTP/2 широко поддерживается во всех новых версиях популярных браузеров.

1. **Поддержка HTTP 2.0 сайтами**

w3techs.com: 23,4% сайтов (14 января 2018)

1 января 2017 – меньше 12%

1. **Поддержка HTTP 2.0 серверами**

Самые популярные версии веб-серверов поддерживают новый НТТР стандарт.

1. **Средний размер веб-страниц и среднее кол-во запросов**

Все больше повседневных человеческих взаимодействий мигрирует в веб-среду. Пользователи требуют почти мгновенного отклика.

Требования к веб-приложениям существенно изменились. Раньше веб среда была не такой.

Страница среднестатистического современного сайта – 1,9 МБ данных:

- images

- JS

- CSS и др.

Без модификаций HTTP 1.1 уже не может дать ожидаемого перфоманса. HTTP должен был эволюционировать.

1. **Проблемы 1.1 – Почему он не может справиться с дальнейшим увеличением**

- ответы обрабатываются в порядке отправки запросов

- блокировка очереди запросов (Head-of-Line blocking)

- один запрос – одно TCP-соединение

- заголовки запросов часто повторяющиеся и чрезмерные.

1. **Объяснение проблем 1.1: последовательные ответы, HOL**

Суть проблемы показана на слайде.

Отправка запросов происходит последовательно после приема ответа – один за другим (HTTP 1.0).

Решение: конвейерная передача – pipelining (НТТР 1.1) Несколько запросов подряд отправляются один за другим еще до получения ответа по первому.

Но время от времени все равно случались серьезные задержки, связанные с проблемой Head-of-Line blocking.

Как известно в протоколе 1.1 ответы обрабатываются последовательно в той последовательности, в который они были отправлены. Поэтому, если по какой-либо причине зависнет один запрос в начале очереди запросов, то все запросы за ним будут оставаться в режиме ожидания.

1. **Очередь**
2. **Жизненный цикл запрос-ответ**

* (Worst case) DNS lookup to resolve the hostname to IP address
* (Worst case) New TCP connection, requiring a full roundtrip to the server
* (Worst case) TLS handshake with up to two extra server roundtrips!
* HTTP request, requiring a full roundtrip to the server
* Server processing time

1. **Как разработчики решали проблемы HTTP 1.1**

- Создание спрайтов

Это когда из множества маленьких изображений собирается одно большое.

- Встраивание

Вместо отправки отдельных изображений оно встраивается в страницу в виде data URL (base64)

- Объединение

Множество js и css файлов объединяются в один большой ком. Недостаток – реально может потребоваться только небольшая часть из них.

- Шардинг

Рассредоточение ресурсов по максимально возможному числу различных хостов

1. **Как HTTP/2 решает проблемы**

* мультиплексирование запросов и ответов
* управление потоками через выставление приоритетов и зависимостей между потоками
* новая фича – сервер пуш
* сжатие заголовков

1. **Ключевые концепции**

• Соединение – Один TCP socket

• Stream – двунаправленный логический канал внутри ТСР сокета

• Message – Логическое сообщение – запрос, ответ, управляющее сообщение

• Frame – Наименьший элемент соединения в HTTP/2. Один запрос может быть разбит на несколько частей.

1. **НТТР/2 Юниты**

* Соединение – Один TCP socket
* Stream – Логический канал внутри потока
* Message – Логическое сообщение – запрос, ответ, управляющее сообщение
* Frame – Наименьший элемент соединения в HTTP/2. Один запрос может быть разбит на несколько частей.

1. **Бинарный протокол**

Базовый юнит коммуникации – фрейм. Существует 12 типов фреймов

Заголовок: длина, тип, флаг, идентификатор фрейма.

DATA: так называемая полезная нагрузка фрейма.

Длина – размер секции DATA

Тип – специализация фрейма

Флаг - состояние фрейма (Boolean)

Идентификатор – соотносит фрейм с логическим потоком. Идентификатор потока привязывает каждый фрейм к потоку.

DATA (данные) и HEADERS (заголовки).

Flags

DATA два флага:

- END\_STREAM (указывает на конец потока)

- PADDED (указывает на наличие отступов)

HEADERS такие же флаги, что и у DATA, плюс два:

- END\_HEADERS (указывает на конец фрейма HEADER)

- PRIORITY (указывает приоритет Стрима)

PUSH\_PROMISE два флага:

- END\_HEADERS

- PADDED

Другие типы фреймов не имеют флагов.

1. **Типы фреймов**
2. DATA – передачи HTTP-сообщений
3. HEADERS – передачи полей заголовка
4. PRIORITY – обновления приоритета потока
5. RST\_STREAM – сигнализации завершения потока
6. SETTINGS – связи параметров конфигурации подключения
7. **Типы фреймов**
8. PUSH\_PROMISE – предупреждения браузера об отправке дополнительных ресурсов с помощью PushBuilder
9. PING – измерения времени «туда-обратно» (запрос-ответ)
10. GOAWAY – информирует о прекращении создания потоков для текущего соединения
11. WINDOW\_UPDATE – реализации контроля потока данных
12. CONTINUATION – продолжения последовательности фрагментов блока заголовка
13. ALTSVC – сообщает о наличии альтернативного сервиса, дополнительный маршрут к такому же контенту, хост и номер порта.
14. **Основные возможности**

Мультиплексирование – Приоритезация – Контроль потока данных – Сжатие заголовка – Server Push

1. **Основные возможности – Мультиплексирование**
2. **Запрос-Ответ Мультиплексирование**

HTTP/1.1 один запрос - отдельное TCP-соединение.

Мультиплексирование – множество запросов в рамках одного TCP-соединения.

1. **Кол-во соединений**

Проверка: <http://www.browserscope.org>

Chrome 63.0.3239.132 (Official Build) (64-bit) – 6

Chrome 49.0.2623.112 – 6

Firefox 57.0.4 (64-біт) – 6

Microsoft Edge 25.10586.0.0 - 13

1. **Запрос-Ответ Мультиплексирование**

Главное преимущество HTTP/2 – пакеты множества потоков смешаны в рамках одного соединения.

В http2 мы можем увидеть десятки и сотни одновременных потоков. Цена создания потока очень низка.

Для одного соединения мы можем иметь множество потоков и для одного потока иметь множество фреймов.

Поток может быть закрыт любой из сторон.

Важен порядок, в котором отправляются фреймы. Получатель обрабатывает фреймы в порядке их получения

1. **Запрос-Ответ Внешний вид HTTP 1.1**

Пример привычного запроса-ответа 1.1.

How do the client and server negotiate the unique stream IDs? They don't. Client-initiated streams have even-numbered stream IDs and server-initiated streams have odd-numbered stream IDs. This offset eliminates collisions in stream IDs between the client and server.

1. **Запрос-Ответ Внешний вид HTTP/2**

Пример НТТР/2 запроса-ответа. Это полный аналог запроса-ответа в 1.1, показанного на предыдущем слайде. Этот пример состоит из трех фреймов – 1 header-запрос, 1 header ответ и data фрейм.

На изображении заголовок каждого фрейма состоит из типа фрейма, идентификатора потока и флагов.

1. **Сравнение запросов 1.1 и 2**

Давайте сравним запросы в протоколах 1.1 и 2.

В НТТР 1.1 запрос передаются как простой запрос без полезной нагрузки.

В обоих версиях есть заголовки. В версии 2 заголовки передаются как тело специализированного фрейма Headers.

1. **Сравнение ответов 1.1 и 2**

Теперь сравним ответы в обоих протоколах. Ответ в НТТР/2 состоит из двух фреймов: Headers, Data.

В Headers отправляется вся полезная информация относительно заголовков.

В Data передается непосредственно полезная нагрузка – необходимый ресурс.

1. **Чередование фреймов**

В НТТР/2 фреймы из разных потоков чередуются между собой, в зависимости от приоритетов, веса, зависимости между потоками.

Клиент принимает одновременно фреймы из разных потоков, которые связаны с разными ресурсами и соединяет их, ориентируясь на идентификатор потока.

Чтобы уменьшить блокировку HOL, стандарт HTTP 2.0 требует, чтобы каждый фрейм DATA не превышал 214-1 (16,383) байтов, что означает, что большие сообщения должны быть разбиты на более мелкие куски. Последнее сообщение в последовательности устанавливает флаг END\_STREAM для отметки конца передачи данных.

1. **Отклонение/Отмена потока**

Reset (сброс) — передумал

Один из недостатков HTTP 1.1, когда HTTP-сообщение отправлено с заголовком Content-Length определённой длины, вы не можете так просто его остановить. Конечно, зачастую вы можете (но не всегда – я пропущу здесь длинное объяснение почему это так) разорвать TCP-соединение, но ценой повторного согласования нового TCP-соединения.

Теперь вы можете просто отменить отправку и начать новое сообщение. Это может быть достигнуто отправкой http2-фрейма RST\_STREAM, который таким образом предотвратит растрату полосы пропускания и необходимости разрыва соединения.

1. **Управляющие фреймы**

С помощью управляющих фреймов можно:

1. **Сравнение Waterfall View**

Хорошо видно разницу между обоими протоколами. В 1.1 используется 6 соединений одновременно. И до загрузки 6 первых ресурсов другие ресурсы находятся в ожидании.

Другое дело в НТТР/2. Запросы на ресурсы отправляются в виде фреймов Headers и загрузка начинается почти одновременно с помощью фреймов Data. Скорость загрузки ресурсов зависит от приоритетов, зависимостей.

Зеленая зона характеризует время от отправки запроса до получения первого байта, так называемый показатель TTFB (Time to first byte).

1. **Основные возможности – Приоритезация**
2. **Внешний вид Фрейма Headers при открытии соединения**

Каждый поток имеет приоритет, используемый для того, чтобы показать другому участнику обмена, какие потоки считать более важными.

После того, как сообщение HTTP можно разделить на несколько отдельных фреймов, точный порядок, в котором кадры чередуются и доставляются в соединении, может быть оптимизирован для дальнейшего улучшения производительности приложения. Следовательно, необязательное 31-битное значение приоритета: 0 представляет поток с наивысшим приоритетом; 231-1 представляет поток с наименьшим приоритетом.

1. **Управление приоритезацией**

Приоритеты могут динамически меняться при обмене, что позволит браузеру быть уверенным, что когда пользователь просматривает страницу заполненную картинками, он сможет указать какие изображения являются наиболее важными, или когда вы переключаете вкладки, он может повысить приоритет потокам, которые неожиданно попали в фокус.

1. **Влияние Приоритета и Зависимости на поток**

Каждый поток может иметь:

- вес – 1…256

- зависимость от другого потока

Комбинация веса и зависимостей позволяет построить «дерево приоритетов».

Зависимость объявляется путем ссылки на идентификатор другого потока (родитель).

Если нету – родитель ROOT-поток.

Зависимость указывает, что родительский поток следует обработать и отправить раньше зависимого потока (если это возможно – рекомендация)

Потоки могут иметь общего родителя. В таком случае приоритет определяется весом потока.

Пример: Если поток А имеет вес 12, а В – 4, то поток А должен получить 2/3 доступных ресурсов.

Примеры:

1. Ни А, ни В не имеют поток-родитель. Это значит, что они зависят от корневого потока. Приоритет А – 12, В – 4. Поэтому поток А должен получить 3/4 доступных ресурсов, а В – 1/4.
2. Д – зависит от корневого, С – зависит от Д. Поэтому сначала поток Д отправляет все ресурсы, а потом только С. Вес не имеет значения.
3. Д – впереди всех, потом С, дальше А и В соответственно ¾ и ¼.
4. Д – впереди всех, Е и С – поровну делят ресурсы. После полной обработки Е и С – отправляются А и В – соответственно ¾ и ¼.

Как видно, HTTP/2 предоставляет возможность гибкой настройки приоритезации обработки и отправки ресурсов.

Более того, HTTP/2 позволяет управлять приоритетами прямо во время обработки и отправки ресурсов – в ответ на действия пользователя.

1. **Основные возможности – Управление передачей данных**
2. **Пример Управление передачей данных (Flow Control)**

Мультиплексирование большого количества потоков через одно ТСР соединение приводит к ситуации конкуренции за пропускную способность. Выставление приоритетов потоков может помочь определить относительный порядок доставки данных, но сами по себе приоритеты неэффективны в контроле над тем, как ресурсы распределены между различными фреймами.

Для решения этой задачи в НТТР 2.0 ввели механизм под названием Flow Control (Управление Потоком). Управление потоком базируется на Фреймах WINDOW\_UPDATE. Получатель рекомендует сколько байт данных (полезная нагрузка фрейма Дата) он готов принять для одного потока и для целого соединения.

Управление потоком производится через фрейм WINDOW\_UPDATE фрейм, который определяет идентификатор потока и значение увеличения количества передаваемых данных. Управление потоком может быть отключено получателем.

1. **Основные возможности – Сжатие заголовков**

Элементами технологии HPACK являются следующие:

• Статическая таблица - предопределенная таблица, чья запись представляет собой либо имя заголовка, либо пару имени заголовка и значения заголовка, выбранного в спецификации. К каждой записи можно получить доступ по индексу. Поскольку типичная длина индексов равна семи бит, то отправляя индексы вместо строк, можно существенно сократить размер метаданных.

• Динамическая таблица - аналогична статической таблице, но с динамически зарегистрированными записями. Динамическая таблица имеет верхний предел для своего размера. Если регистрация новой записи вызывает переполнение, старые записи удаляются. Каждая конечная точка использует две динамические таблицы для отправки и получения для каждого соединения.

• Кодировка Хоффмана - кодирование наиболее часто используемых букв в именах заголовков и значениях заголовка с меньшим количеством бит. Отображение статически определено.

1. **Заголовки в НТТР 1.1 – сколько они занимают**

Каждая передача HTTP содержит набор заголовков, которые используются для описания переданного ресурса. В HTTP 1.x эти метаданные всегда отправляются в виде открытого текста и обычно добавляют от 500 до 800 байт служебных данных для каждого запроса и часто намного больше, если необходимы файлы cookie HTTP.

1. **Заголовки в НТТР/2 – сколько они занимают**

Чтобы уменьшить накладные расходы и повысить производительность, HTTP 2.0 сжимает метаданные заголовков:

Вместо повторной передачи одних и тех же данных для каждого запроса и ответа HTTP 2.0 использует таблицы заголовков как на клиенте, так и на сервере для отслеживания и хранения ранее отправленных пар ключ-значение заголовка.

• Таблицы заголовков сохраняются для всего соединения HTTP 2.0 и постепенно обновляются как клиентом, так и сервером.

• Каждая новая пара ключ-значение заголовка либо добавляется к существующей таблице, либо заменяет предыдущее значение в таблице.

1. **Отслеживание ссылок**

В результате обе стороны соединения HTTP 2.0 знают, какие заголовки были отправлены, и их предыдущие значения, что позволяет кодировать новый набор заголовков как простое различие (рисунок 7) из предыдущего набора.

1. **Как работает сжатие заголовков**

Общие пары ключ-значение, которые редко изменяются на протяжении всего срока службы соединения (например, пользовательский агент, заголовок принятия и т. Д.), Должны передаваться только один раз. Фактически, если никакие заголовки не изменяются между запросами (например, запрос опроса для того же ресурса), тогда служебные данные заголовка-заголовка равны нулю, все заголовки автоматически унаследованы от предыдущего запроса.

1. **Как выглядят статические таблицы заголовков**
2. **Как происходит сжатие – Пример 1**
3. **Как происходит сжатие – Пример 2**
4. **Основные возможности – Server Push**
5. **Как работает Push Promise**

Если клиент запрашивает один ресурс, сервер часто может прогнозировать, что ему понадобятся другие ресурсы, указанные на странице. В HTTP / 1.1 для предоставления этих ресурсов клиентам как часть первого ответа использовался Inlining. У Inlining есть свои недостатки - особенно в том, что встроенный ресурс нельзя кэшировать для использования на других страницах, которые также могут на него ссылаться.

Это позволяет серверам предоставлять преимущества Inlining, но в форме, которую клиент может кэшировать и повторно использовать на других страницах.

1. **Сравнение времени получения HTTP-ресурсов**
2. Последовательные запросы (НТТР 1.0)
3. Конвейерные запросы (НТТР 1.0)
4. Чередующийся запросы (НТТР/2)
5. Чередующийся запросы с Push (HTTP/2)
6. **Сравнение ключевых особенностей загрузки страниц 1.1 и 2**

|  |  |
| --- | --- |
| **HTTP/1.1 Page Load** | **HTTP/2 Page Load** |
| Создание 6-13 параллельных соединений | Создается одно соединение |
| Запрос HTML страницы | Запрос HTML страницы |
| Получение HTML страницы | Получение HTML страницы |
| Декодирование HTML страницы | Декодирование HTML страницы |
| Запрос первых 6-13 файлов, вложенных в HTML страницу, никаких приоритетов или зависимостей  Заголовки в запросах не сжаты, передаются в виде простого текста | Запрашиваются все файлы, вложенные в HTML страницу, с приоритетами и зависимостями  Заголовки в запросах сжаты, имеют бинарный формат |
| В каждом соединении ожидается полная загрузка запрашиваемого ресурса | Ресурсы возвращаются, разбитые на множество фреймов, по готовности |
| Запрос следующего файла по освободившемуся соединению | - |
| Repeat 6-7 for each remaining file | - |
| Закрытие 6-13 соединений | Закрытие одного соединения |

Шаги, которые быстрее и эффективнее в НТТР/2:

1. Создание и закрытие соединений – НТТР/2 создает одно соединение, в отличие от НТТР 1.1, где число соединений 6-13.
2. Ожидании для делания запроса – НТТР/2 запрашивает все ресурсы одновременно, с использованием мультиплексирования и приоритезации для получения высокоприоритетных ресурсов настолько быстро насколько возможно. НТТР 1.1 делает 6-13 запросов, потом ожидает получения перед тем, как отправлять дополнительные запросы, один за раз для каждого соединения как только оно стает доступным.
3. Отправка меньше заголовков – в отличие от 1.1. Кроме того, заголовки в НТТР/2 сжаты и соответственно происходит передача меньшего числа байт.
4. **Как понять поддерживает ли данный хост НТТР/2 соединение**

Запрос от клиента отправляется по протоколу 1.1 со специальным заголовком Upgrade:h2c

При чем, h2 означает шифрованное соединение TLS, h2с – соответственно не шифрованное.

Если сервер не поддерживает протокол 2, то он просто проигнорирует эти заголовки.

Если же поймет, то тогда ответ также будет отправлен по протоколу 1.1 со специальным заголовком HTTP/1.1 101 Switching Protocols. После этого и клиент и сервер начинают взаимодействовать, используя протокол НТТР/2.

1. **Конфигурация Tomcat для использования с НТТР/2**
2. **Maven зависимость для использования НТТР/2**
3. **Ключевые особенности НТТР/2, привнесенные благодаря Servlet 4.0**

API-интерфейс Servlet хорошо подходит для использования оптимизаций, привнесенных HTTP/2 и позволяет использовать функциональную возможность PushServer.

Какими возможностями НТТР/2 можно управлять с помощью API Servlet 4.0:

- ServerPush

- Выставление приоритетов потоков

- Мультиплексирование запросов-ответов

1. **Новые классы**
2. **Новые методы**
3. **Что происходит при использовании PushBuilder**

Запрос ресурса GET /index.html

Сервер обнаруживает, что запрашиваемый ресурс также нуждается в еще двух файлах: style.css, common.js

Из объекта HttpServletRequest запрашивается PushBuilder, которому присваивается путь к ресурсу и вызывается метод push()

1. **Структура интерфейса PushBuilder**
2. **Пример использования PushBuilder в Фильтре**

Пример того, как можно использовать PushBuilder в фильтрах, делая push ресурсов в зависимости от каких-либо характеристик запроса, например запрашиваемого ресурса.

1. **Заключение: что мы получаем вместе с HTTP/2**

Как НТТР/2 повлияет на обычных людей?

НТТР/2 ещё широко не представлен и не используется. Мы не можем сказать точно как всё сложится.

НТТР/2 уменьшает количество необходимых сетевых приёмов-передач, полностью избегает дилеммы блокировки начала очереди за счёт мультиплексирования и быстрого отклонения нежелательных потоков.

Он позволяет работать множеству параллельных потоков, число которых может превышать число соединений даже у наиболее активно использующих **шардинг** современных сайтов.

С приоритетами, корректно используемыми на потоках, шансы получить важные данные раньше менее важных значительно выше.

Собрав всё это вместе, я скажу, что очень высоки шансы, что это приведёт к ускорению загрузки страниц и повысит отзывчивость веб-сайтов. Коротко: лучше ощущения от веб-серфинга.

Насколько быстрее и насколько лучше, мы увидим. Я не думаю, что мы пока готовы сказать.

Во-первых технология по-прежнему ещё молода, а во-вторых, мы ещё не видели аккуратных реализаций клиентов и серверов, которые по-настоящему используют всю мощь, которую предоставляет новый протокол.

1. **THANK YOU**

**Слайд 16 Сжатие заголовков**

[HPACK](http://tools.ietf.org/html/draft-ietf-httpbis-header-compression-07) – формат сжатия для http2-заголовков (отдельная спецификация).

In HTTP/1.x, header data is sent as plain text. In HTTP/2, header data for the single,

multiplexed connection is compressed according to the new HPACK standard, which uses

Huffman coding. Header data is also sent in binary format, further reducing file size.

**Слайд 17**

**Слайд Сжатие данных**

Непосредственно перед выпуском черновика 12, была добавлена поддержка DATA-фреймов сжатых gzip. Каждый фрейм сжимается индивидуально, поэтому между ними нет общего контекста, но это немного снижает уровень сжатия. Эта возможность соответствует использованию gzip в Transfer-Encoding в HTTP 1.1. Возможность, которая редко используется, но часто обсуждается, как нечто провальное в протоколе, по крайне мере для браузеров.

**Слайд Server push (посылка сервера)**

Эта возможность также известна как «посылка в кэш». Идея в том, что если клиент запрашивает ресурс X, а сервер предполагает, что клиент наверняка затем попросит ресурс Z, отправляет этот ресурс клиенту без просьбы с его стороны. Это помогает клиенту поместить Z в свой кэш, и он будет на месте, когда потребуется.  
Посылка сервера – это то, что клиент явно должен разрешить серверу, и даже если он разрешил, он может по своему выбору быстро отменить посланный поток с помощью RST\_STREAM, если он ему оказался не нужен.

**Слайд Альтернативные сервисы**

После адаптации http2 есть причины ожидать, что TCP-соединения будут более продолжительными и дольше сохраняться в рабочем состоянии, чем это было с HTTP 1.x соединениями. Клиент сможет делать многое, что он захочет в рамках одного соединения к каждому хосту/сайту и это соединение вероятно будет открыто очень долго.  
Это повлияет на работу HTTP-балансировщиков, и могут возникнуть ситуации, когда сайт захочет предложить клиенту подключиться к другому хосту. Это может быть как по причинам производительности, но также и необходимости отключения для обслуживания или подобных целей.

Сервер отправляет заголовок Alt-Svc (или ALTSVC-фрейм в http2), сообщая клиенту о наличии альтернативного сервиса. Дополнительный маршрут к такому же контенту, используя другой сервис, хост и номер порта.

Ожидается, что клиент попытается асинхронно подключиться к сервису и начать использовать альтернативный сервис, если он нормально работает.

**Слайд Выводы о http/2.0**

**Слайд 19 Push Builder from HTTP Request**

Simply you get the push builder from the HTTP Request object and set the path to the resource and push.

There are two things to note in this sequence diagram,

the push builder can be reused. In the example, I use the push builder to push two resources the CSS file and the JavaScript file.

the second thing is that the index.html is returning to the browser after the push resource.

The reason is that if the index returns before the pushed resources, the browser will analysis it and see that it needs the two resources. It will look in the cache and see that it does not have those resources and it will request them. At this point, the browser cache will not be prepopulated. So the pushed resources must be returned first before the index is sent.

Push Promise

One of the frames types mentioned earlier was a RST\_STREAM this is how the client can decline a push promise. So if the server pushes a resource and the browser already has it in the cache then rather than let the server send the file it will send an RST\_STREAM frame saying that it already has the files file so don’t send it.

**Слайд 20**

The only required setting is the URI path to be used for the push request. This must be called before every call to push().

If the path includes a query string, the query string will be appended to the existing query string (if any) and no de-duplication will occur.

Paths beginning with ‘/’ are treated as absolute paths. All other paths are treated as relative to the context path of the request used to create this builder instance. The path may include a query string.

The resource is pushed by calling the push() method on the pushBuilder instance.

**Disable/Reject Server Push**

The client can explicitly disabled server push by sending a SETTINGS\_ENABLE\_PUSH setting value of 0 (zero).

In addition to allowing clients to disable server push with the SETTINGS\_ENABLE\_PUSH setting, servlet containers must honor a client’s request to not receive a pushed response on a finer grained basis by heeding the CANCEL or REFUSED\_STREAM code that references the pushed stream’s stream identifier. One common use of this interaction is when a browser already has the resource in its cache.

Это реализовано путем эффективного мультиплексирования, передачи данных с низкой задержкой через одно соединение (single connection). Теперь это позволит разработчикам отказаться от большинства различных хаков, которые используются в вебе, чтобы обойти ограничения HTTP 1.1.

С ростом сложности веб-приложений этих хаков уже недостаточно для обеспечения повышенных требований к производительности и отзывчивости.

И тогда, разработчики из Гугла обратили внимание на лежащий в основе транспортный протокол, ответственных за перемещение данных.