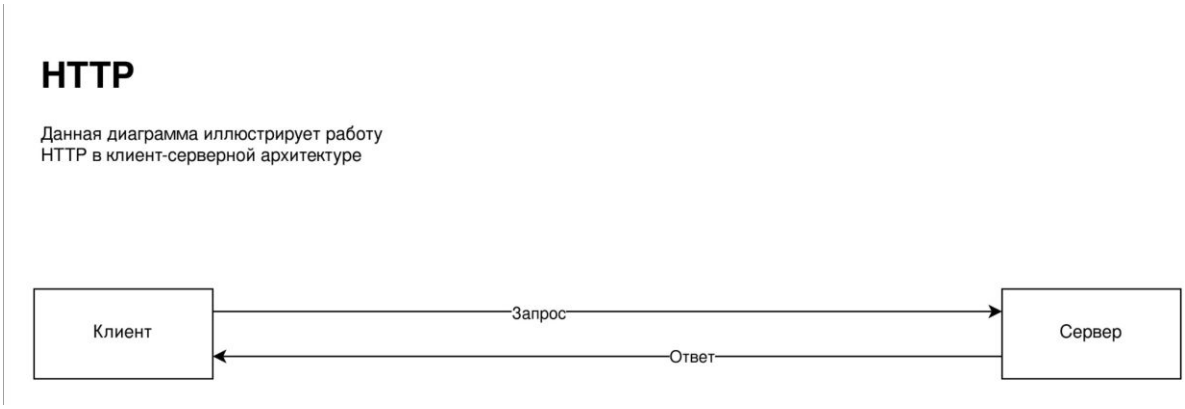
Протокол HTTP - это протокол *запроса-ответа*. Это означает, что только клиент может отправлять HTTP-запросы на сервер. Сервер может обслуживать HTTP-запросы, отправляя обратно HTTP-ответы, но сервер не может отправлять незапрошенные HTTP-ответы клиенту.



Особенности **HTTP**

- HTTP не поддерживает соединение, после того, как отдает ответ на запрос.
- HTTP обязует клиентов заранее оговаривать действие, которое клиент хочет

  сделать в заголовке (HTTP Headers) - GET, POST, PUT, DELETE
- Мы отправляем заголовок что хотим сделать каждый раз, как обраемся к серверу

HTTP изначально был разработан для передачи ресурсов типа запрос-ответ в распределенных системах гипермедиа, но не для одновременной двунаправленной связи.

Протокол **WebSocket** разработан для замены существующих механизмов обхода **HTTP** и обеспечения эффективного протокола для одновременной двунаправленной связи с низкой задержкой между браузерами и серверами по одному **TCP**-соединению.

Механизмы на основе HTTP

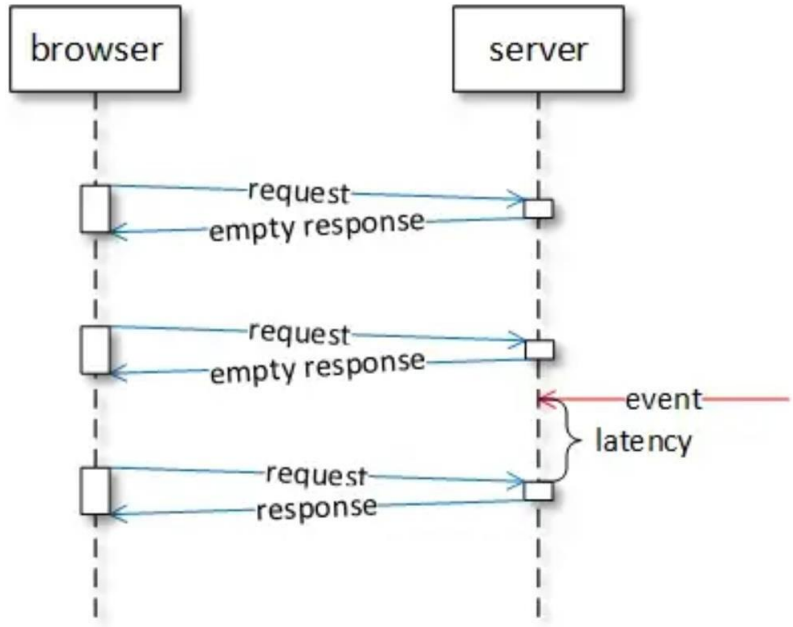
Поскольку HTTP не был разработан для поддержки сообщений, инициируемых

сервером, было разработано несколько механизмов для достижения этой цели,

каждый из которых имеет свои преимущества и недостатки:

1)**HTTP-опрос**

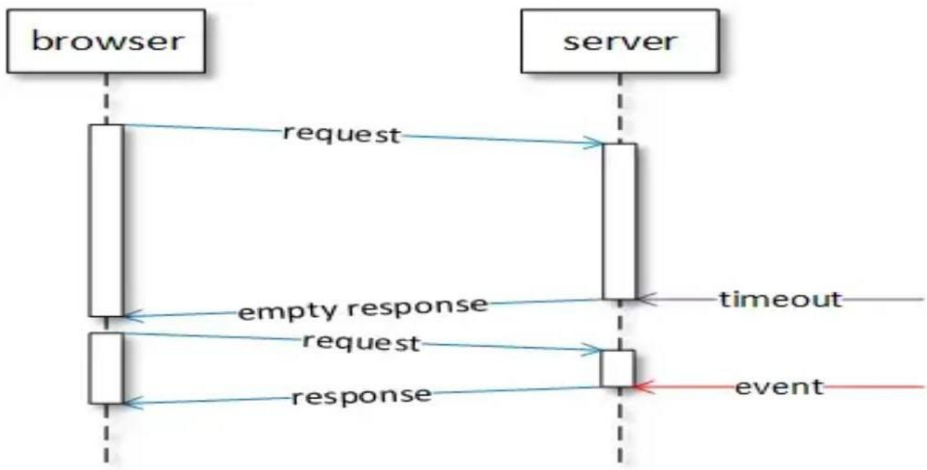
Во время механизма *опроса* клиент отправляет периодические запросы на сервер, и сервер немедленно отвечает. Если есть новые данные, сервер возвращает их, в противном случае сервер возвращает пустой ответ. После получения ответа клиент некоторое время ждет, прежде чем отправить другой запрос.



Опрос может быть эффективным, если мы знаем период обновления данных на сервере. В противном случае клиент может опрашивать сервер либо слишком редко (добавляя дополнительную задержку при передаче данных с сервера клиенту), либо слишком часто (тратя впустую серверную обработку и сетевые ресурсы).

Длинный опрос HTTP

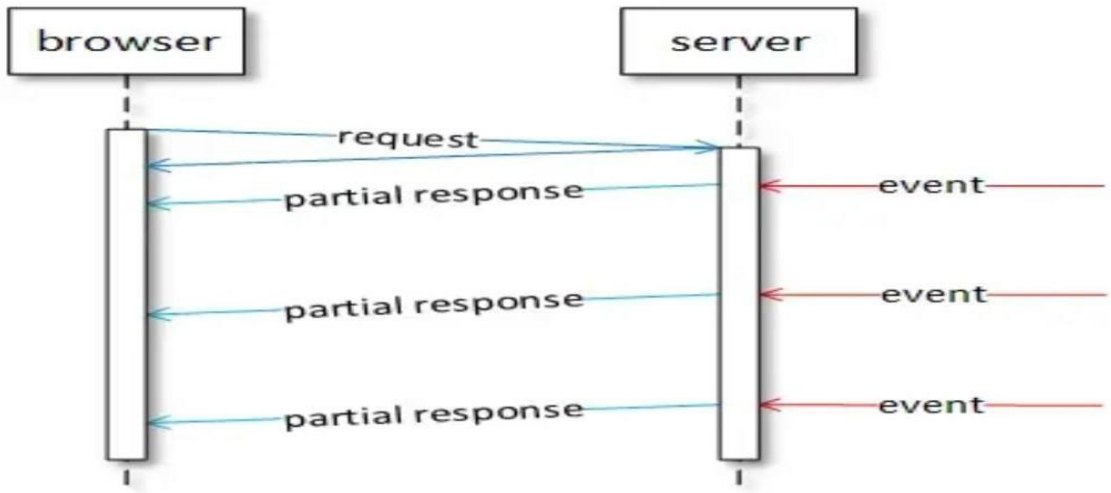
Во время механизма *длительного опроса* клиент отправляет запрос на сервер и начинает ждать ответа. Сервер не отвечает, пока не поступят новые данные или не наступит тайм-аут. Когда новые данные становятся доступными, сервер отправляет ответ клиенту. После получения ответа клиент немедленно отправляет другой запрос.



Длительный опрос сокращает использование серверных и сетевых ресурсов для получения обновлений данных с низкой задержкой, особенно там, где новые данные становятся доступными с нерегулярными интервалами. Однако сервер должен отслеживать несколько открытых запросов. Кроме того, время ожидания длительных запросов может истечь, и новые запросы должны периодически отправляться, даже если данные не обновляются.

Потоковая передача HTTP

Во время механизма *потоковой* передачи клиент отправляет запрос на сервер и сохраняет его открытым на неопределенный срок. Сервер не отвечает, пока не поступят новые данные. Когда новые данные становятся доступными, сервер отправляет их обратно клиенту как часть ответа. Данные, отправленные сервером, не закрывают запрос.



Потоковая передача основана на способности сервера отправлять несколько фрагментов данных в одном ответе, не закрывая запрос. Этот механизм значительно снижает задержку в сети, поскольку клиенту и серверу не нужно отправлять и получать новые запросы. Однако клиент и сервер должны договориться о том, как интерпретировать поток ответов, чтобы клиент знал, где заканчивается одна часть данных и начинается другая. Кроме того, сетевые посредники могут нарушать потоковую передачу — они могут буферизировать ответ и вызывать задержки или отключать соединения, которые остаются открытыми в течение длительного времени.

WebSocket

Предварительные требования

WebSocket предназначен для преодоления ограничений механизмов на основе HTTP (опрос, длительный опрос, потоковая передача) при *полнодуплексной* связи между браузерами и серверами:

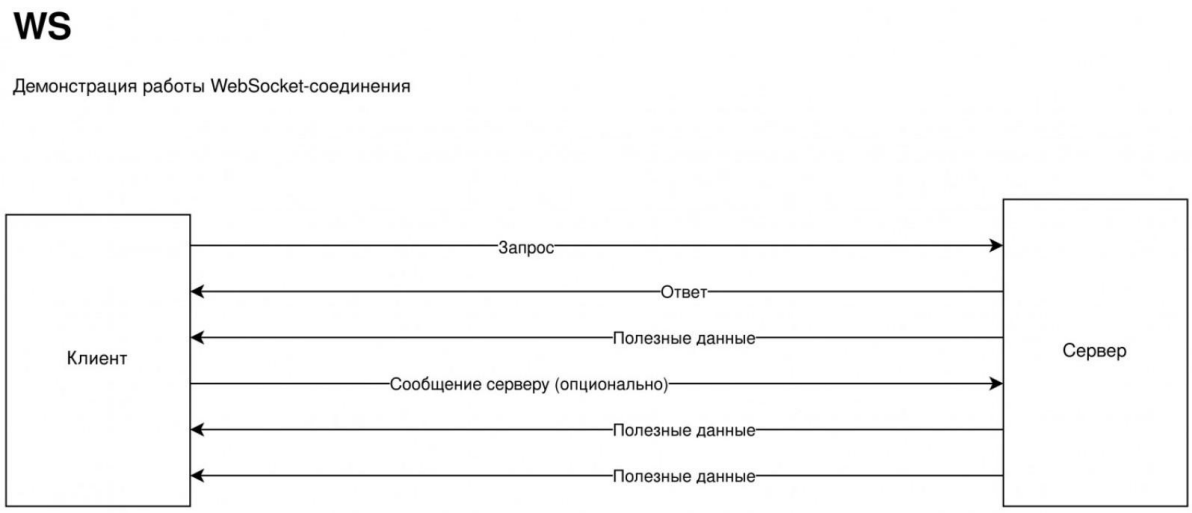
При *полнодуплексной* связи обе стороны могут отправлять и получать сообщения в обоих направлениях одновременно.

При *полудуплексной* связи обе стороны могут отправлять и получать сообщения в обоих направлениях, но одновременно в одном направлении.

HTTP позволяет осуществлять *полудуплексную* связь между браузером и сервером: браузер может либо отправлять запросы на сервер, либо получать ответы от сервера, но не оба одновременно. Чтобы преодолеть эти ограничения, несколько механизмов Comet используют два одновременных HTTP-соединения для восходящей и нисходящей связи между браузером и сервером, что приводит к дополнительной сложности.

Вот основные различия между **HTTP** и **WebSocket**:

- **HTTP** - это текстовый протокол, **WebSocket** - двоичный протокол (двоичные протоколы передают меньше данных по сети, чем текстовые протоколы)
- **HTTP** имеет заголовки запросов и ответов, сообщения **WebSocket** могут иметь формат, подходящий для конкретных приложений (ненужные метаданные не передаются по сети)
- **HTTP** - это полудуплексный протокол, **WebSocket** - это полнодуплексный протокол (сообщения с низкой задержкой могут передаваться одновременно в обоих направлениях)



Браузер и сервер должны договориться об общем формате сообщений, прежде
чем эта семантика сможет быть выражена. Существует несколько

протоколов, которые могут помочь.

Simple/Streaming Text-Oriented Messaging Protocol (STOMP)

STOMP — протокол обмена сообщениями, созданный предельно простым.

Основан на фреймах по образцу HTTP. Фрейм состоит из команды, необязательных заголовков и необязательного тела. Клиент пошлет фрейм

SUBSCRIBE, где заголовок «destination» показывает на что конкретно он

подписывается:

```
SUBSCRIBE
id:sub-1
destination:/topic/price.stock.*
```

Как только ресурс становятся доступными, сервер отправляет фрейм MESSAGE с соответствующим «destination» и идентификатором подписки, а

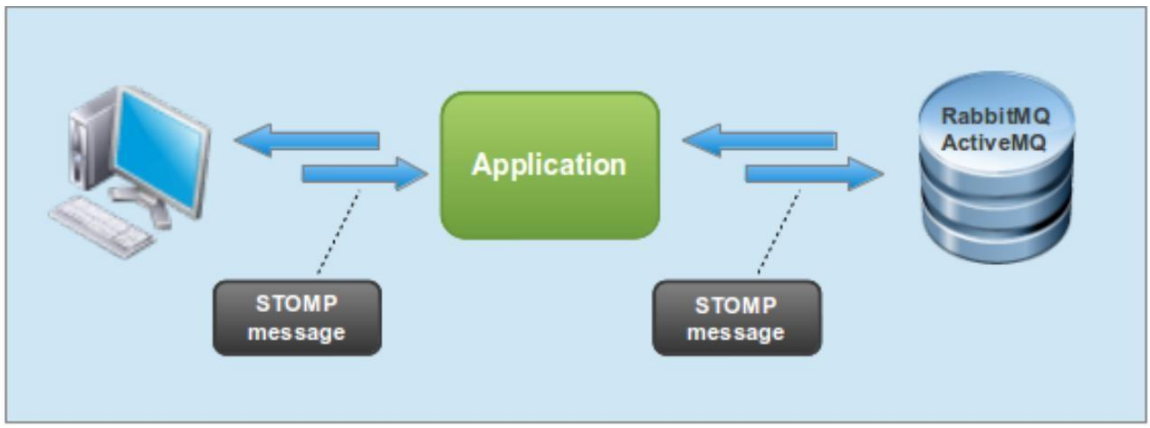
также заголовок «content-type» и тело:

```
MESSAGE
subscription:sub-1
message-id:wm2si1tj-4
content-type: application/json
destination:/topic/stocks.PRICE.STOCK.NASDAQ.EMC

{\"ticker\":\"EMC\",\"price\":24.19}
```

Приложение + Message-broker

Другой подход заключается в том, чтобы сделать приложение, обрабатывающее входящие сообщения и выступающее в качестве посредника между веб-клиентами и брокером. Сообщения от клиентов к брокеру можно отправлять через приложение, обратное сообщение также пройдет через
приложение к клиенту. Это позволяет приложению определить тип сообщения и заголовок «destination», после чего решить обрабатывать самостоятельно
и



AMQP (Advanced Message Queuing Protocol) — открытый протокол для передачи сообщений между компонентами системы. Основная идея состоит в том, что отдельные подсистемы (или независимые приложения) могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет маршрутизацию, возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений.

Загрузка портфеля позиций

- Клиент запрашивает портфель позиций
- Приложение обрабатывает запрос путем загрузки и возврата данных для подписки
- Message-broker не участвует в этом взаимодействии

Подписка на котировки акций

- Клиент отправляет запрос на подписку
- Приложение передает сообщение брокеру
- Message-broker передает сообщение всем подписанным клиентам

Получение котировок акций

- QuoteService посылает брокеру сообщение с котировками акций
- Message-broker передает сообщение всем подписанным клиентам

Проведение сделки

- Клиент отправляет торговый запрос
- Приложение не обрабатывает его, все сделки проходят через TradeService
- Message-broker не участвует в этом взаимодействии

Особенности протокола

О плюсах WebSocket:

1. Поддерживает двусторонний обмен данными: может одновременно получать и передавать информацию.
2. Отправляет данные быстрее, чем HTTP.
3. Кроссплатформенная совместимость. Для сервера веб-сокета не имеет значения, кто выступает в качестве клиента: веб-сайт или мобильные приложение.
4. HTTP требует до 2000 байтов накладных расходов, тогда как веб- сокет — всего 2 байта.
5. Кратковременное отсутствие связи не прерывает соединение. Если интернет будет работать нестабильно, клиенту и серверу не придётся заново устанавливать WebSocket-соединения, они смогут

обмениваться данными в обычном режиме, пока связь не восстановится.

6. Протокол позволяет работать в асинхронном режиме вместо привычной для веба работы «Запрос-ответ». Так, у клиента и сервера равноправные роли в процессе обмена данными, и они действуют автономно.

О минусах:

1. Высокие требования к серверному оборудованию.
2. Молчаливый отвал соединения. При отправке пакета в WebSocket вы не узнаете о том, доставлен ли он или нет, пока не пройдёт время таймаута — по умолчанию 75 секунд. Зачастую приходится вводить дополнительные механизмы общения между клиентом и сервером, чтобы быстро понять, отвечает ли клиент.
3. Смена сети клиентом. Бывают ситуации, когда клиент находится в дороге и меняется сеть, в которой находится устройство. В таких случаях сервер ничего не знает о смене адреса, если клиент не закрыл соединение при переподключении к другой сети.