

Урок №12

# Real-Time сообщения

---

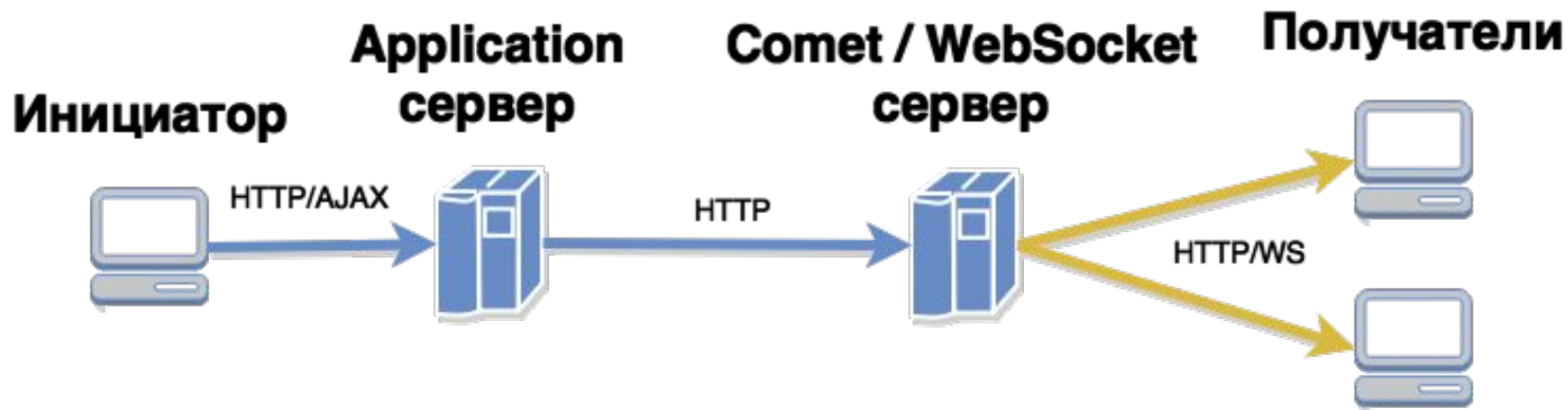
# Содержание занятия

1. Polling
2. Comet
3. Server Push
4. WebSocket
5. Centrifugo

# Примеры использования

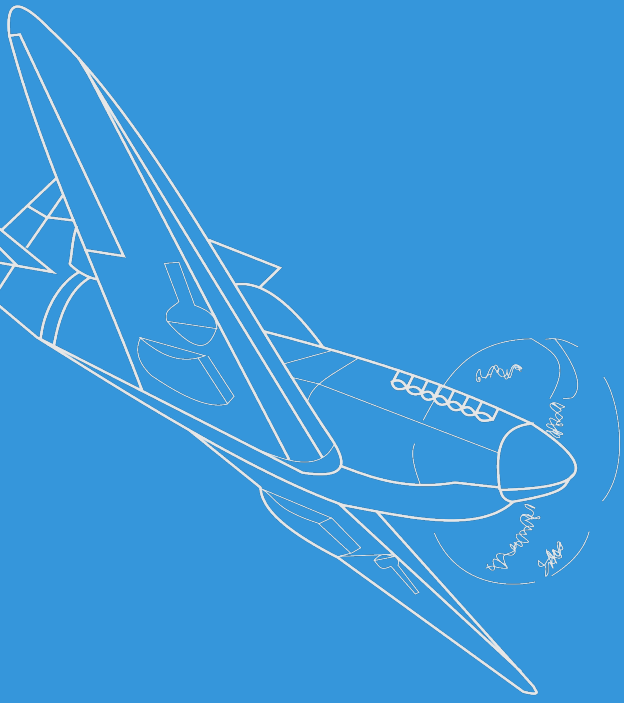


- Чаты и мессенджеры
- Отображение котировок
- Прямые трансляции (a-la twitter)
- Push уведомления
- Сетевой обмен в играх на HTML





- **Polling** — периодический опрос сервера
- **Comet (Long polling)** — polling с долгоживущими запросами
- **Server Push** — бесконечный запрос
- **WebSocket** — специализированный протокол



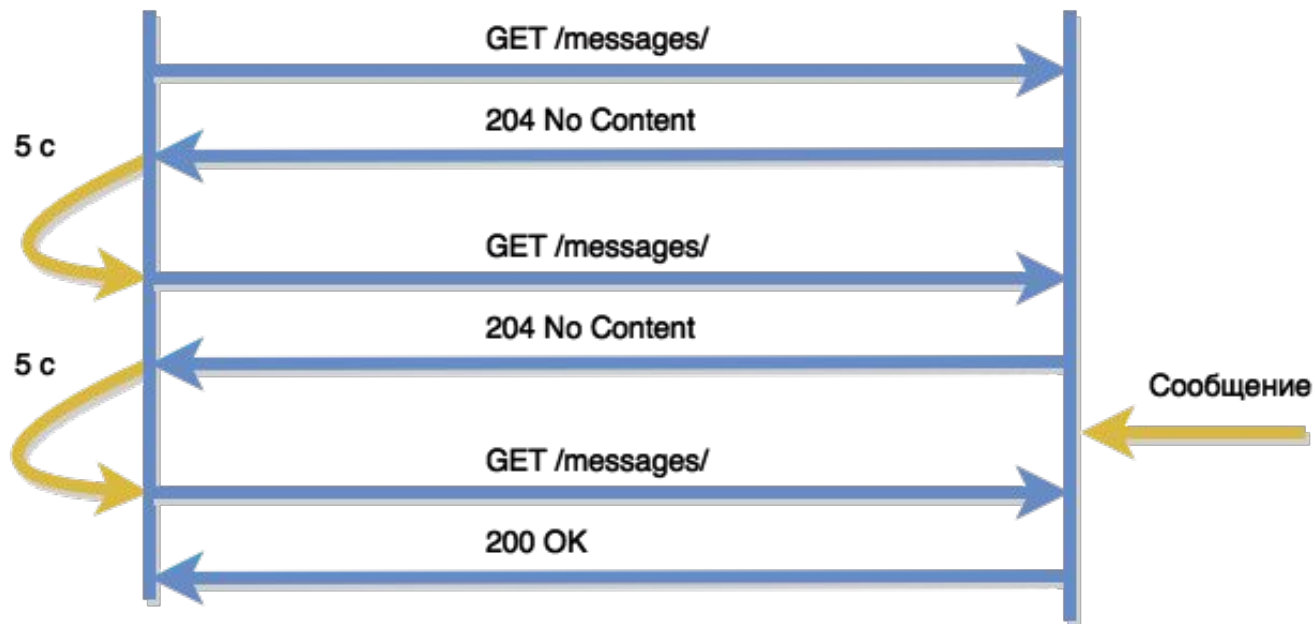
# Polling

# Polling — периодический опрос



**Web-клиент**

**Web-сервер**



# Polling на клиенте



```
var since = 0;

setInterval(function() {
    $.ajax({
        type: 'GET',
        url: '/messages/',
        data: { channel_id: 5, since: since },
    }).success(function(resp) {
        if (!resp.messages || !resp.messages.length) {
            return;
        }
        handleMessages(resp.messages);
        since = resp.messages[0].id;
    }); }, 5000);
```



# Polling на сервере

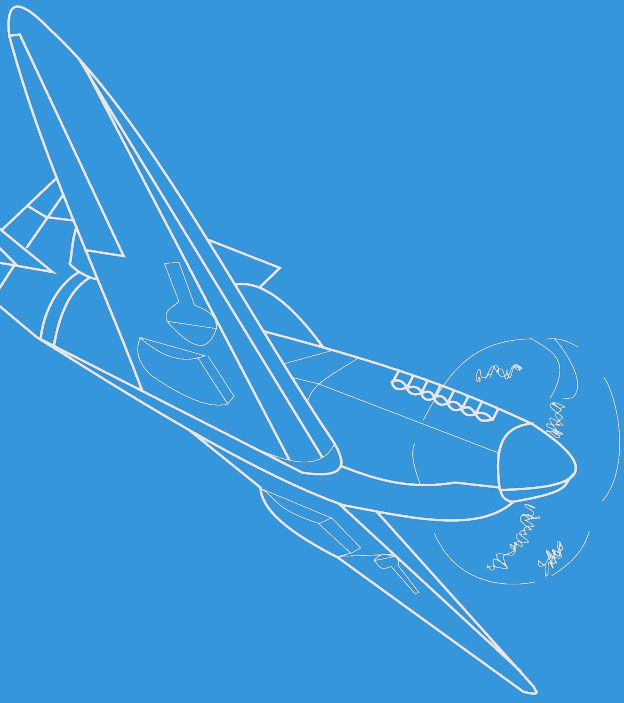


```
def messages(request):  
    cid = request.GET.get('channel_id')  
    since = request.GET.get('since', 0)  
    messages = Messages.filter(  
        channel_id = channel_id,  
        id__gt = since,  
    ).order_by('-id')  
    messages = [m.as_data() for m in messages]  
    return HttpResponseAjax(messages = messages)
```

# Плюсы и минусы Polling



- + Простота и надежность реализации;
- + Не требуется дополнительного ПО;
- Сообщения приходят с задержкой до N секунд;
- Избыточное число HTTP запросов  $RPS = CCU / N$ ;
- Ограничение по числу пользователей.



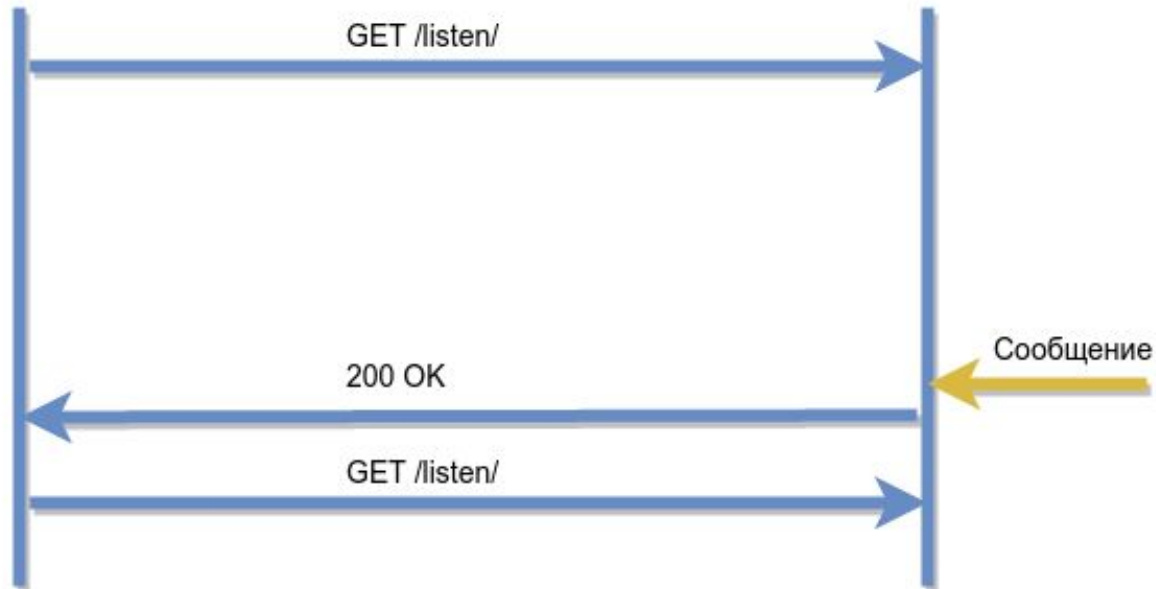
Comet

# Comet — долгоживущие запросы



**Web-клиент**

**Comet-сервер**



# Comet на клиенте



```
function getComet() {  
    $.ajax({  
        type: 'GET',  
        url: '/listen/',  
        data: { channel_id: 5 },  
    }).success(function(resp) {  
        handleMessages(resp.messages);  
        getComet();  
    }).error(function() {  
        setTimeout(getComet, 10000);  
    });  
}  
getComet();
```

# Comet на сервере



В технологии comet сервер должен поддерживать одновременно открытыми большое количество соединений, причем каждое соединение находится в ожидании сообщений для него. По этой причине мы не можем использовать классический application- сервер в роли comet-сервера.

Для comet-сервера необходима отдельная технология, например **nginx + push-stream-module**

<https://github.com/wandenberg/nginx-push-stream-module>

# Nginx + push-stream-module



```
location /publish/ {
    push_stream_publisher normal;          # включаем отправку
    push_stream_channels_path $arg_cid;    # id канала
    push_stream_store_messages off;        # не храним сообщения
    allow 127.0.0.1;
    deny all;
}

location /messages/ {
    push_stream_subscriber long-polling;  # включаем доставку
    push_stream_channels_path $arg_cid;    # id канала
    default_type application/json;         # MIME тип сообщения
}
```

# Nginx + http-push (старый модуль)



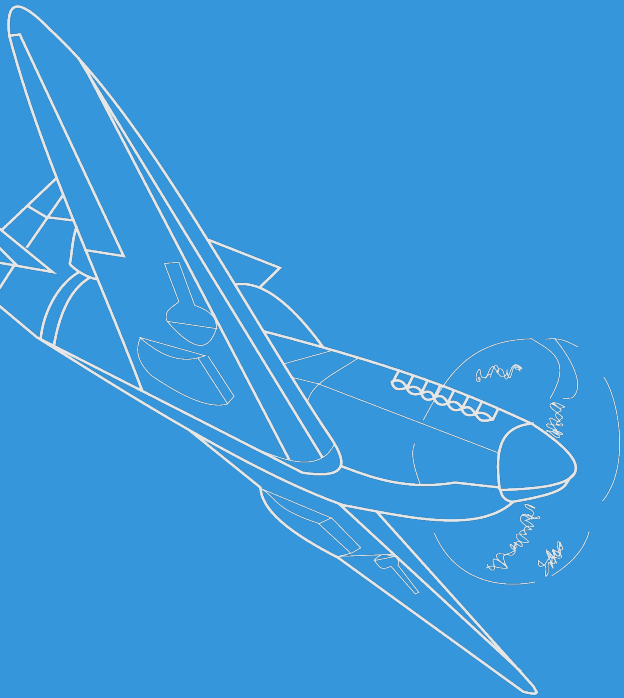
```
location /publish/ {  
    set $push_channel_id $arg_cid; # id канала  
    push_store_messages off;        # не храним сообщения  
    push_publisher;                 # включаем отправку  
    allow    127.0.0.1;  
    deny     all;  
}  
  
location /listen/ {  
    push_subscriber_concurrency broadcast; # всем!  
    set $push_channel_id $arg_cid;         # id канала  
    default_type application/json;         # MIME тип сообщения  
    push_subscriber;                       # включаем доставку  
}
```



# Плюсы и минусы Comet



- + Поддержка всеми браузерами
- + Поддержка большого числа пользователей
- + Относительная простота реализации
- Избыточные HTTP запросы
- Half-duplex



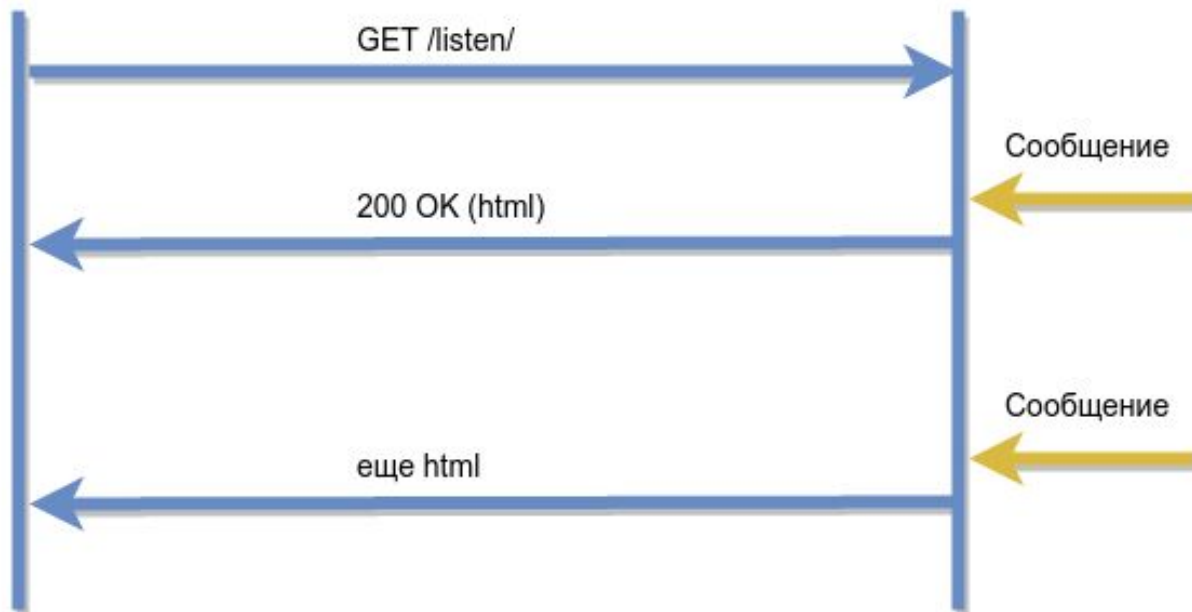
Server push

# Server push - бесконечный запрос



**Web-клиент**

**Push-сервер**



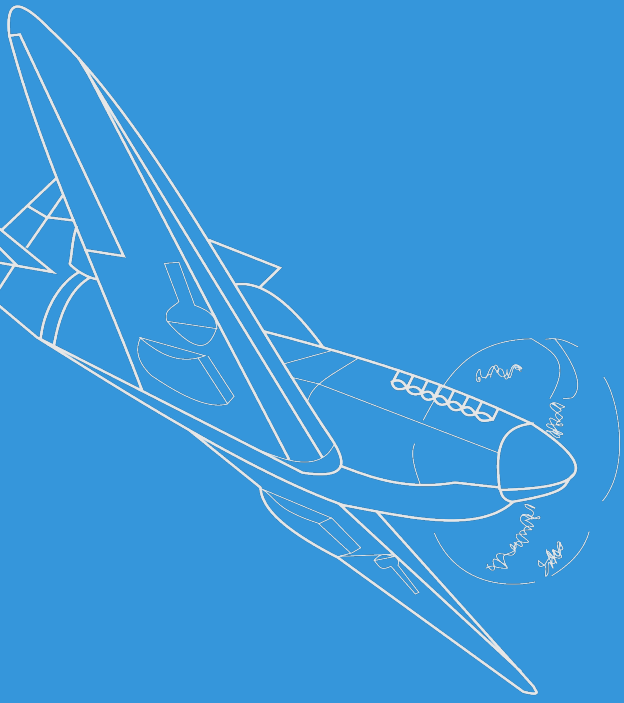
# Server push на клиенте



```
<script>
    function handle(message) {
        // любая логика
    }
</script>
<iframe src='/listen/?cid=123'></iframe>
```

Ответ сервера:

```
<script>parent.handle({ message: 'hello' })</script>
```



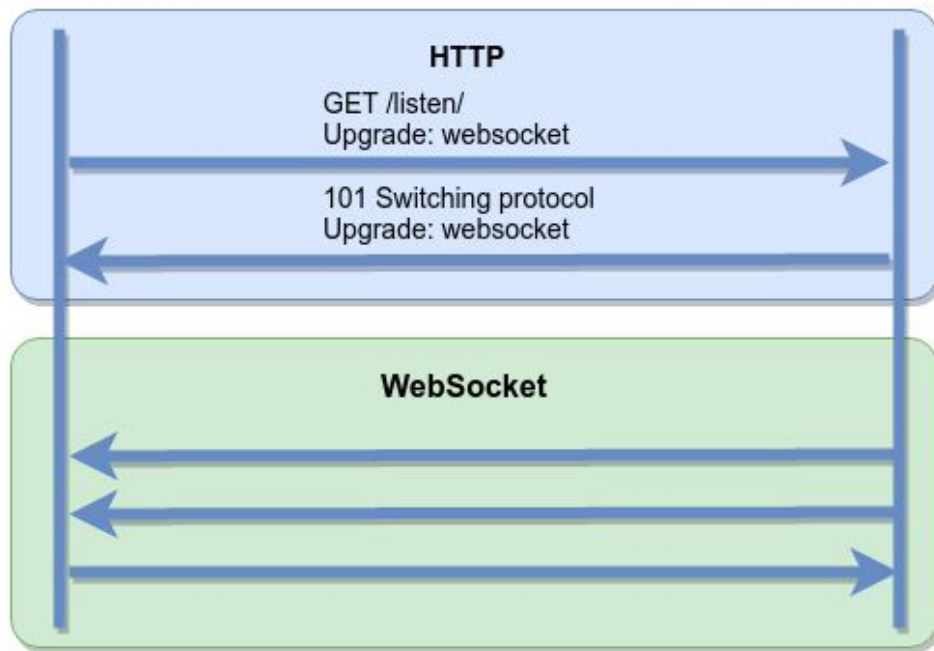
# WebSocket

# WebSocket



**Web-клиент**

**WebSocket-сервер**



# WebSocket handshake. Запрос



**GET** /listen HTTP/1.1

Host: server.example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

Sec-WebSocket-Origin: http://example.com

Sec-WebSocket-Protocol: chat, superchat

Sec-WebSocket-Version: 13

# WebSocket handshake. Ответ



**HTTP/1.1 101 Switching Protocols**

Upgrade: **websocket**

Connection: **Upgrade**

Sec-WebSocket-Accept: **s3pPLMBiTxaQ9kYGzzhZRbK+x0o=**

Sec-WebSocket-Protocol: **chat**



# WebSocket на стороне клиента



```
var socket = new WebSocket('ws://host/echo');

socket.onopen = function(event) {
    console.log('ws opened');
    var data = JSON.stringify({ message: "Hello WebSocket" });
    socket.send(data);
};

socket.onmessage = function(event) {
    var resp = JSON.parse(event.data);
    console.log('ws message', resp.message);
};

socket.onclose = function(event) {
    console.log('ws closed');
};
```

# WebSocket на стороне сервера



```
class EchoWebSocket(tornado.websocket.WebSocketHandler):  
    def open(self):  
        print("WebSocket opened")  
  
    def on_message(self, message):  
        self.write_message(message)  
  
    def on_close(self):  
        print("WebSocket closed")
```

# Плюсы и минусы WebSocket



- + Минимальный объем трафика
- + Минимальная нагрузка на сервер
- + Поддержка большого числа пользователей
- + Full-duplex
- Нет поддержки IE<10, OperaMini, Android<4.4
- Требуется специальный WebSocket-сервер
- Плохо работает с прокси-серверами

# Софт для Real Time сообщений



- Real Time Web Technologies Guide

<https://www.leggetter.co.uk/real-time-web-technologies-guide/>

- Real Time libraries and frameworks

<https://deepstream.io/blog/realtime-framework-overview/>

- Centrifugo

<https://github.com/centrifugal/centrifugo>



1. Устанавливаем  
<https://github.com/centrifugal/centrifugo/blob/master/docs/content/server/install.md>
2. Генерируем конфиг  
`centrifugo genconfig`
3. В настройках бекенда регистрируем Centrifugo secret and Centrifugo API key (дефолтный адрес <http://localhost:8000/api>)
4. Подключаем библиотеку для клиента  
<https://github.com/centrifugal/centrifuge-js>
5. Подписываем клиенты на каналы

## Домашнее задание № 12



1. Установить и поднять centrifugo;
2. Подключить centrifugo к проекту на стороне клиента и сервера;
3. Организовать отправку/получение сообщений с помощью centrifugo.

---

## Рекомендуемая литература



Для саморазвития (опционально)

Чтобы не набирать двумя  
пальчиками

Спасибо за  
внимание!

**Антон Кухтичев**



[a.kukhtichev@mail.ru](mailto:a.kukhtichev@mail.ru)



[@toshunster](https://www.instagram.com/toshunster)