Internet технологии

ЛЕКЦИЯ №8

ОРГАНИЗАЦИЯ ПОЛНОДУПЛЕКСНОГО ОБМЕНА ДАННЫМИ МЕЖДУ БРАУЗЕРОМ И ВЕБ-СЕРВЕРОМ С ПРИМЕНЕНИЕМ ПРОТОКОЛА WEBSOCKET

Содержание

- Концепция Rich Internet Application.
- Классический polling с применением протокола HTTP 1.1.
- Модели разработки приложений, поддерживающих постоянное HTTP-соединение.
- Особенности протокола **WebSocket**.
- Node.js и Socket.io API.
- Пример приложения: мониторинг погоды.

RIA – Rich Internet Application

Rich Internet application (RIA, «насыщенное (богатое) интернет-приложение») – это приложение, доступное через Интернет, обладающее функциональностью настольных приложений, которая предоставляется спецификой браузера, или через плагин, или путём использования «песочницы».

В отличие от обычных веб-сервисов, использующих клиент-серверную архитектуру с тонким клиентом, RIA используют архитектуру с толстым клиентом.

Наиболее распространенными платформами для RIA являются Adobe Flash, JavaFX и Microsoft Silverlight

Как правило, приложение RIA передаёт веб-клиенту необходимую часть пользовательского интерфейса, оставляя большую часть данных (ресурсы программы, данные и пр.) на сервере. Запускается локально в среде безопасности, называемой «песочница» (sandbox).

RIA. Преимущества

- RIA предлагают пользовательский **интерфейс**, не ограниченный лишь использованием языка HTML, применяемого в стандартных веб-приложениях.
- запускается в **браузере** и не требует дополнительной установки ПО;
- **интерактивность**;
- клиентская часть приложения обладает возможностями **кэширования** данных и работы без подключения к сети;
- Использование вычислительных ресурсов клиента и сервера лучше **сбалансировано**. Поэтому сервер не должен быть «рабочей лошадкой», как в традиционных веб-приложениях, что освобождает вычислительные **ресурсы сервера**.
- **Асинхронная коммуникация**, клиентская часть может взаимодействовать с сервером, не дожидаясь, пока пользователь совершит действие в приложении, нажав на кнопку или ссылку.

RIA. Недостатки

- требуется **JavaScript** или другие скриптовые языки;
- некоторые RIA используют скриптовый язык на стороне клиента, например, JavaScript, с частичной потерей **производительности**;
- **время загрузки** скрипта;
- поисковые системы могут оказаться не в состоянии **проиндексировать** содержимое приложения RIA. Часто, однако, индексирование в полной мере и не требуется.

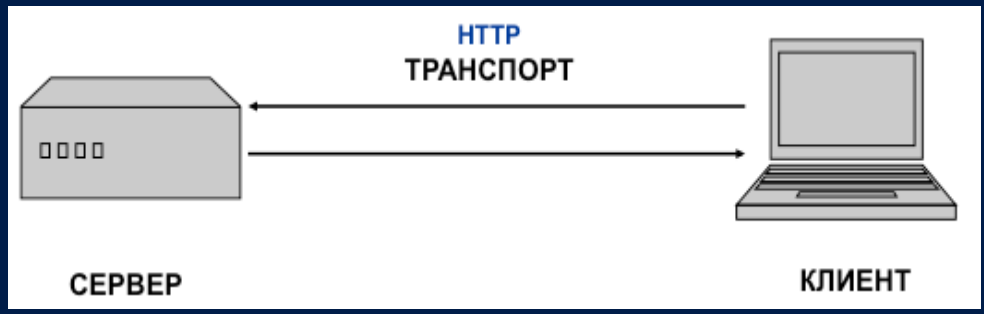
Типичные real-time приложения

- веб-приложения с **интенсивным обменом** данными, требовательные к скорости обмена;
- комплексные приложения со множеством различных **асинхронных** блоков на странице;
- **многопользовательские** веб-приложения в реальном времени;
- **кроссдоменные** приложения.

Например, мгновенный обмен сообщениями, электронная почта, аукционы, букмекерство, игры, отображение спортивных результатов, мониторинг состояния датчиков.

Классический опрос (polling) посредством HTTP/1.1.

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:
1.Стартовая строка — определяет тип сообщения;
2.Заголовки — характеризуют тело сообщения, параметры передачи и прочие сведения;
3.Тело сообщения — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.



Запрос от браузера:

GET / HTTP/1.1
Host: webgyry.info
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive

Ответ сервера:

HTTP/1.1 200 OK
Date: Sun, 10 Feb 2013 03:51:41 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=5
Server: Apache
X-Konkurentam: Preved

<!DOCTYPE html>
<html >…</html>

Классический опрос (polling) посредством HTTP/ 1.1.

Запрос от браузера:
GET /wiki/CORBA HTTP/1.1
Host: ru.wikipedia.org
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: text/html
Connection: close

Разберем заголовки запроса детальнее:
GET – метод HTTP-запроса. GET-запросы могут кэшироваться.
/wiki/CORBA – это URL той страницы или файла, которые мы хотим получить.
HTTP/1.1 – наименование протокола и его номер версии.
User-Agent – содержит строку-идентификатор клиента.
Список заголовков заканчивается ещё одной комбинацией символов CR-LF (то есть, пустой строкой). После заголовков идёт тело запроса. В данном примере тело запроса пустое (нулевой длины).

Классический опрос (polling) посредством HTTP/ 1.1.

HTTP-ответ:
HTTP/1.1 200 OK
Server: nginx/0.5.35
Date: Tue, 22 Apr 2008 10:18:08 GMT
Content-Type: text/html;
charset=windows-1251
Connection: close

Last-Modified: Fri, 30 Nov 2007 12:46:53 GMT
ETag: ``27e74f-43-4750063d''
Content-Length: 34
(пустая строка)
(далее следует запрошенная страница в HTML)

Некотрые заголовки ответа:
Стартовая строка HTTP/1.1 200 OK
200 – код ответа (запрошенный файл или страница обнаружены и отданы клиенту).
OK – человеко-читабельное описание кода ответа.
Accept – список допустимых форматов ресурса.
ETag – тег, используемый при кэшировании.
Connection – сведения о проведении соединения.
Referer – URI ресурса, после которого клиент сделал текущий запрос.

Оптимизация передачи данных посредством HTTP

- **HTTP keep-alive**. Один раз подключившись к серверу, клиент обменивается с ним запросами и ответами, не разрывая соединение. Клиент может запросить разрыв соединения после ответа, передав в запросе заголовок Connection: close. Ваше веб-приложение должно правильно выставлять заголовок Content-Length, а иначе keep-alive соединения не будут работать.
- **HTTP-pipelining**. Отправка подряд нескольких запросов перед началом обработки ответов. Плохая поддержка клиентами.
- **HTTP-кэширования**. Используется заголовок ответа Cache-Control. Проблема обратной совместимости с HTTP 1.0.
- **Компрессия**. Например, удаление лишних пробелов и переводов строк из HTML-файлов и CSS-файлов, CSS-spriting, минификация (вместе с обфускацией) JavaScript.

Классический опрос (polling) посредством HTTP/ 1.1.

Проблемы:

- **Синхронность** (низкая производительность и интерактивность).
- Жесткое **распределение ролей** «клиент» и «сервер» – сервер не имеет возможности инициировать отправку данных
- **Низкая эффективность** при передаче большого количества маленьких по объему данных.
- **Масштабируемость** (сложности при увеличении количества пользователей).

Пути решения:

- Минимальная латентность и накладные расходы.
- Асинхронность.

Модели разработки real-time приложений

Частый опрос (Polling)

Браузер через регулярные промежутки времени отправляет HTTP-запрос на сервер. Например, тег <meta http-equiv= "refresh" content="5"> или методы setInterval и setTimeout. Большой overhead на http-запросы, тем более в случае если частота сообщений меньше частоты запросов.

Удержание соединения (Comet via Long-Polling)

Запрос удерживается сервером на протяжении определенного промежутка времени. Например, до момента выполнения какого-то события. Нерационально используется полоса пропускания.

Стриминг (Streaming)

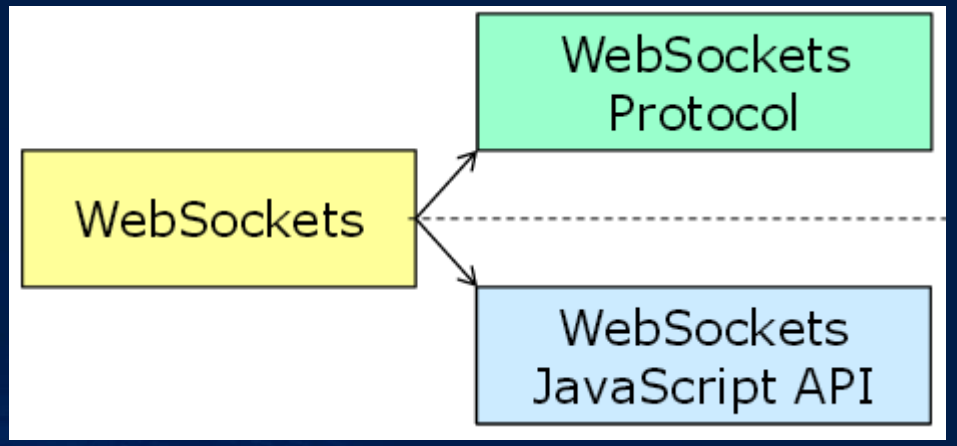
Запрос может удерживаться сервером бесконечно долго. Проблема с http proxy – могут пытаться буфферизировать ответ, сводя на нет преимущества.

Протокол WebSocket

WebSocket - протокол полнодуплексной связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени. Websockets является альтернативой использованию технологии polling.

Базовые идеи двухсторонней коммуникации:

- клиент или сервер могут послать сообщение;
- клиент или сервер могут выполнить код на основе сообщения по Javascript событию.

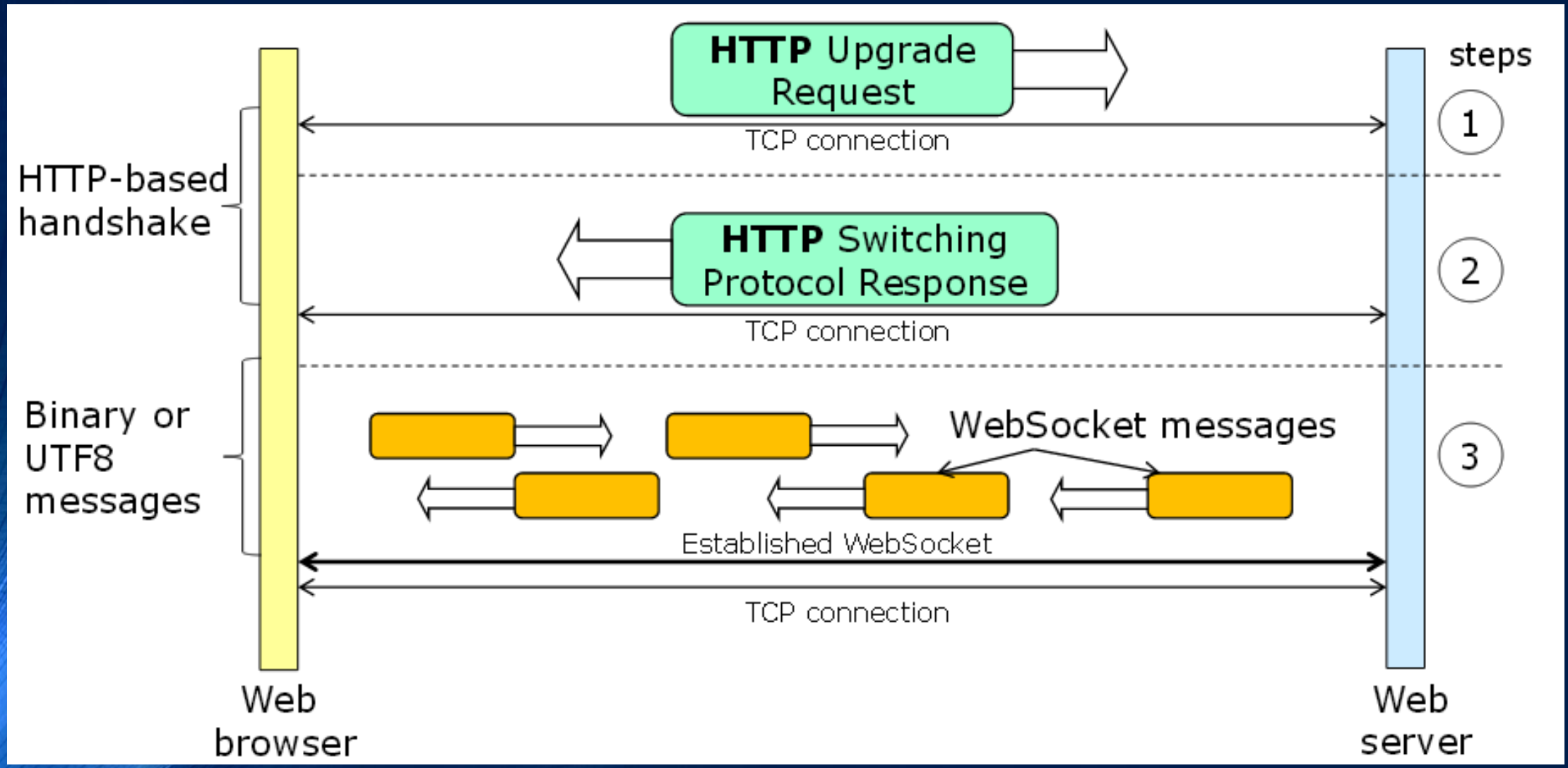


WebSocket. Преимущества

Преимущества технологии WebSocket:

- низкие требования к сетевым ресурсам, максимальный КПД передачи данных, минимум накладных расходов;
- входит в стандарт HTML5;
- двунаправленная передача данных;
- асинхронность;
- время жизни канала в **неактивном** состоянии;
- возможность работы с разными доменами;
- неограниченное количество подключений к одному домену;
- очень простое событийно-ориентированное API.

Протокол WebSocket



WebSocket Protocol Handshake

Клиент

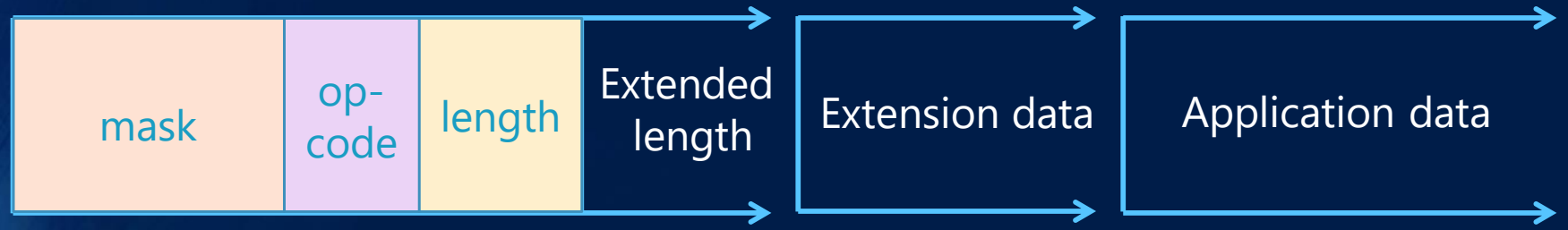
Обязательные:
GET /chat HTTP/1.1
Host: server.domain.com
Upgrade: websocket
Connection: upgrade
Sec-WebSocket-Key: 16-byte nonce, base64
Sec-WebSocket-Version: 13
Опциональные:
Origin: http://domain.com
Sec-WebSocket-Protocol: protocol
Sec-WebSocket-Extensions: extension
Cookie: cookie content

Сервер

Обязательные:
HTTP/1.1 101 Switching Protocols
Upgrade: **websocket**
Connection: upgrade
Sec-WebSocket-Accept: 20-byte MD5 hash in base64
Опциональные:
Sec-WebSocket-Protocol: protocol
Sec-WebSocket-Extension: extension

WebSocket. Формат фреймов

Каждый фрейм содержит несколько заголовочных байтов, определяющих, фрагментировано ли сообщение, тип передаваемых данных, размер данных, маску и другие управляющие данные (ping, pong…).
Данные могут пересылаться как в текстовом так и в бинарном виде.



Нефрагментированное текстовое сообщение Hello без маски:

0x81 0x05 0x48 0x65 0x6c 0x6c 0x6f

WebSocket. Поддержка браузерами

- Chrome 4.0+
- Safari 5.0 & iOS 4+
- Firefox 4+
- Opera 10.7+
- Internet Explorer 10+

На клиенте:

```
var webSocket = new
WebSocket('ws://localhost/echo');
webSocket.onopen = function(event) {
  alert('onopen');
  webSocket.send("Hello Web Socket!");
};
webSocket.onmessage = function(event) {
  alert('onmessage, ' + event.data);
};
webSocket.onclose = function(event) {
  alert('onclose');
};
```

WebSocket. Серверные реализации

Основные серверные реализации, базирующиеся на событийно-ориентированной модели:

- **Javascript**: Node.js with Socket.IO
- **Java**: Jetty
- **Erlang**: MochWeb, Misultin
- **Perl**: AnyEvent, Coro, POE
- **Python**: Twisted, Tornado
- **Ruby**: EventMachine
- **PHP**: phpDaemon
- IISNode on IIS 8

Пакеты для разработки с помощью Node.JS

- **npm** – Node Packaged Modules, менеджер пакетов для node. Устанавливается вместе с node. npm install express mongoose jade less
- node-inspector и nodemon – отладка и авторестарт разрабатываемых приложений.
- node-validator – библиотека для проверки, фильтрация и санитизации строк.
- bcrypt – библиотека для хеширования паролей.
- mongoose – mongodb для node.
- node_redis – клиент для Redis для node.
- Jade – шаблонизатор для node.
- Nodemailer – модуль для отправки электронной почты с помощью node.
- express – node-фреймворк для построения одно- и многостраничных веб приложений.
- **socket.io** – унифицированное средство обмена данными.

Пакет socket.io

Socket.io – унифицированное средство обмена данными, библиотека для кросс-браузерной поддержки WebSockets.

Предоставляет легкий и удобный уровень абстракции, позволяющий использовать все возможные технологии обмена данными броузера с сервером в реальном времени (WebSocket, Adobe® Flash® Socket, AJAX long polling, AJAX multipart streaming, Forever Iframe, JSONP Polling), в т.ч. обеспечивает работу в старых IE и т.д.

Пакет socket.io. Поддержка

Поддерживаемые транспорты:

- WebSocket
- Adobe® Flash® Socket
- AJAX long polling
- AJAX multipart streaming
- Forever Iframe
- JSONP Polling

Поддерживаемые браузеры:

- Internet Explorer 5.5+
- Safari 3+
- Google Chrome 4+
- Firefox 3+
- Opera 10.61+
- iPhone Safari
- iPad Safari
- Android WebKit
- WebOs WebKit

socket.io. Опции конфигурации на стороне сервера

- heartbeats (по умолчанию включено) – используется ли режим heartbeats для проверки состояния соединения socket.io;
- transports – **транспорт** (по умолчанию websocket, htmlfile, xhr-polling, jsonp-polling);
- log level (по умолчанию равен 3) – данные, выводимые регистратору: 0 – ошибка, 1 – предупреждения, 2 – информация, 3 – отладка;
- close timeout (по умолчанию 60 секунд) – тайм-аут для клиента, в течении которого в случае закрытия соединение имеется возможность повторного его открытия. Это значение посылают клиенту после успешного рукопожатия;
- heartbeat timeout (по умолчанию 60 секунд) – тайм-аут для клиента, в течении которого он должен отправить новое значение heartbeat на сервер. Это значение посылают клиенту после успешного рукопожатия.

socket.io. Выбор транспорта передачи данных

```
// подключаем модуль для создания сервера
// и ставим на прослушивание 80-порта
var io = require('socket.io').listen(80);

//Способ 1
//ограничиваем транспорт только протоколом WebSocket
io.set('transports', ['websocket']);
//Способ 2
io.configure(function () {  io.set('transports', ['websocket',
'flashsocket', 'xhr-polling']);});
```

socket.io. Опции конфигурации на стороне клиента

- connect timeout (по умолчанию 10000 мс) – задержка перед попыткой подключения к серверу с использованием другого транспорта;
- reconnect (по умолчанию включена) – повторное подключение в случае, если socket.io обнаружит разрыв связи или тайм-аут.
- reconnection delay (по умолчанию 500 мс) – задержка перед началом восстановления соединения;
- maxReconnectionAttempts – максимальное количество попыток переподключения.

Обработка событий на стороне сервера

io.sockets.on('**connection**', function(socket) {}) – начальное соединение от клиента. Аргумент socket должен быть использован в дальнейшей коммуникации с клиентом.

socket.on('**message**', function (message, callback) {}) – обработка сообщения.

socket.on('**disconnect**', function() {}) – событие разъединения срабатывает во всех случаях, когда соединение клиент-сервер закрыто. Срабатывает в случаях желательного, нежелательного, мобильного, не мобильного, клиентского и серверного отключения. Не существует события **восстановления** связи. Вы должны использовать событие "connection " для восстановления управляемости.

Закрыть соединение имеется возможность **у любой из сторон**, как сервера и/или браузера, поскольку в итоге существует только одно соединение.

Обработка событий на стороне клиента

socket.on('**connect**', function () {}) – когда сокет успешно подключен к серверу;

socket.on('**connecting**', function () {}) – когда сокет пытается подключиться;

socket.on('**disconnect**', function () {}) – когда сокеты отключены;

socket.on('**reconnecting**', function () {}) – при попытке сокета восстановить связь;

socket.on('**connect_failed**', function () {}) – socket.io не может установить соединение с сервером и не имеет других вариантов транспорта;

socket.on('**reconnect_failed**', function () {}) – когда socket.io не удается восстановить рабочую связь после того как подключение было прекращено;

socket.on('**reconnect**', function () {}) – socket.io успешно повторно подключился;

socket.on('**error**', function () {}) – при возникновении ошибки.

socket.on('**message**', function (message, callback) {}) – обработка сообщения.

Пример мониторинга погоды. Разметка

```
Выберете населенный пункт для слежения:
<select name="station_name" autofocus
onchange="stationSelect(this.options[this.selectedIndex].value)">
  <option>Харьков</option>
  <option>Москва</option>
</select>
Частота обновления:
<input type="number" id="quantity" min="1000" max="60000" value="1000">
<table border="1" id="messages"> <!-- Таблица для вывода информации -->
  <thead>
    <tr><th>Метеостанция</th><th>Время</th><th>Температура</th></tr>
  </thead>
  <tbody></tbody>
</table>
```

Пример мониторинга погоды. Подключение к серверу

```
<script src="http://localhost:8080/socket.io/socket.io.js"></script>
<script>
var socket;
function connect_server() {
  var serverURL = 'http://localhost:8080'; //адрес сервера
  socket = io.connect(serverURL); // открываем соединение
  socket.on('connect', function clientConnected() {
    //Определяем обработчик приема сообщений от сервера.
    socket.on('message', function messageReceived(msg) {
      var msgObj = JSON.parse(msg); //разбор JSON-сообщения
      addRow(msgObj.station, msgObj.time, msgObj.temperature);
    });
  });
}
</script>
```

Пример мониторинга погоды. Обработка на клиенте

```
/* Обработка выбора станции наблюдения: подключение к серверу,
периодический опрос */
function stationSelect(station_name) {
  if(station_name != "-") {
    connect_server();
    var quantity = document.getElementById('quantity').value; //интервал опроса
    setInterval(function() {
      socket.send(JSON.stringify({type : 'request', station : station_name}));
    }, quantity);
  }
}
```

Пример мониторинга погоды. Обработка на клиенте

```
/*Добавить новые данные в таблицу результатов */
function addRow(station, time, temperature) {
  // Находим нужную таблицу
  var tbody =
document.getElementById('messages').getElementsByTagName('TBODY')[0];
  // Создаем строку таблицы и добавляем ее
  var row=tbody.insertRow(0);
  var td1=row.insertCell(0); var td2=row.insertCell(1); var td3=row.insertCell(2);
  // Наполняем ячейки
  td1.innerHTML = station; td2.innerHTML = time; td2.innerHTML = time;
}
```

Пример мониторинга погоды. Подключение на сервере

```
var port = 8080;
var io = require('socket.io').listen(port); // подключаем модуль для создания сервера и ставим на прослушивание 8080-порта
io.set('log level', 1); // отключаем вывод полного лога
io.set('transports', ['websocket']); //ограничиваем транспорт только веб-сокетом
io.sockets.on('connection', function (socket) {
  //Добавляем обработчик принятого сообщения
  socket.on('message', function messageReceived(msg) {
    var msgObj = JSON.parse(msg); //разбор JSON-сообщения, получаем JavaScript-объект
    if (msgObj.type === 'request')
      SendWeather(socket, msgObj.station);
  });
});
```

Пример мониторинга погоды. Обработка на сервере

```
function SendWeather(socket, station) {
  var dt = (new Date).toLocaleTimeString();
  var msgData = {
    station : station,
    time: dt,
    temperature : GetTemperature(-40, 40)
  }
  socket.send(JSON.stringify(msgData));
}

function GetTemperature(min, max) {
   var temperature = Math.round(Math.random() * (max - min + 1) + min);
   return temperature;
}
```

Вопросы

- Особенности Rich Internet Application.
- Преимущества и недостатки Rich Internet Application.
- Структура HTTP-запроса. Заголовки HTTP.
- Способы оптимизации передачи данных при использовании HTTP.
- Недостатки опроса посредством HTTP/1.1.
- Модели разработки real-time приложений.
- Протокол WebSocket и его преимущества. WebSocket Protocol Handshake.
- Известные реализации серверов, поддерживающие WebSocket.
- Node.js и его отличительные особенности. Что такое MEAN?
- Назначение пакета socket.io. Транспорты, поддерживаемые пакетом socket.io.
- Конфигурирование socket.io на стороне клиента и сервера.
- Обработка событий socket.io на стороне клиента и сервера.
- Методы socket.io для отправки сообщений.