LR_15

1. **Что такое AJAX?**

   AJAX (Asynchronous JavaScript and XML) - это технология веб-разработки, которая позволяет обмениваться данными между браузером и сервером асинхронно. С использованием AJAX можно отправлять запросы на сервер и получать данные без перезагрузки всей веб-страницы. Это обеспечивает более динамический и отзывчивый пользовательский интерфейс.

2. **Что такое протокол WebSocket?**

   Протокол WebSocket представляет собой стандарт обмена данными между веб-браузером и веб-сервером в режиме реального времени. Он обеспечивает полнодуплексное соединение, что позволяет как серверу, так и браузеру отправлять данные друг другу в любой момент времени без необходимости повторной инициации подключения.

3. **Какие дополнительные возможности предоставляет Web Sockets API?**

   Дополнительные возможности Web Sockets API включают:

   - **Бинарные данные:** Возможность передачи бинарных данных, а не только текстовых.
   - **Расширения:** Возможность использования расширений для улучшения функциональности протокола.
   - **Поддержка подпротоколов:** Возможность использования подпротоколов для дополнительной функциональности поверх стандартного протокола WebSocket.
   - **Безопасность:** Встроенные механизмы обеспечения безопасности, такие как поддержка шифрования.

Что такое API WebSocket?

WebSocket API — это стандартизированный протокол и интерфейс прикладного программирования (API) , который обеспечивает непрерывную двустороннюю связь между клиентом и сервером. Он использует единое долговременное соединение, которое позволяет отправлять и получать данные в режиме реального времени, обеспечивая взаимодействие с малой задержкой и эффективную связь.

Традиционная модель HTTP «запрос-ответ» может привести к задержке из-за затрат на установление и завершение нескольких соединений между клиентом и сервером. WebSocket API решает эту проблему, поддерживая постоянное соединение, сокращая накладные расходы и обеспечивая более быстрое реагирование. Это особенно полезно в приложениях обмена данными в реальном времени, таких как онлайн-игры, финансовые торговые платформы и чат-приложения. API WebSocket поддерживается современными веб-браузерами, что упрощает разработчикам реализацию функций реального времени на различных платформах.

События WebSocket и обработчики событий

События WebSocket запускаются браузером асинхронно на различных этапах жизненного цикла соединения WebSocket, указывая текущее состояние соединения. К этим событиям относятся открытие, закрытие и получение сообщения. Обработчики событий — это функции JavaScript , назначенные этим событиям и определяющие поведение приложения в ответ на них. Основные события WebSocket и соответствующие им обработчики событий следующие:

1. `onopen` : срабатывает, когда соединение успешно открыто. На этом этапе вы можете начать отправлять сообщения на сервер. Пример:

```
socket.onopen = (event) => { console.log('WebSocket connectio
```

2. `onclose` : срабатывает, когда соединение закрывается из-за успешного закрытия соединения, сбоя или неожиданного завершения. Пример:

```
socket.onclose = (event) => { console.log(`WebSocket connecti
```

3. `onmessage` : срабатывает при получении сообщения от сервера. Объект события, передаваемый обработчику событий, включает свойство `data` , содержащее данные полученного сообщения. Обратите внимание, что сообщения можно получать в текстовом или двоичном формате. Пример:

```
socket.onmessage = (event) => { console.log('Received message
```

4. `onerror` : срабатывает, когда возникает ошибка во время связи через WebSocket. За этим событием может последовать событие `onclose` , если ошибка приводит к разрыву соединения. Пример:

```
socket.onerror = (event) => { console.log('WebSocket error en
```

Назначив соответствующие функции этим обработчикам событий, вы можете определить, как ваше приложение будет реагировать на различные события, и обеспечить бесперебойную связь через WebSocket.

Отправка и получение сообщений

API WebSocket обеспечивает двустороннюю связь между клиентом и сервером в режиме реального времени. Процесс отправки и получения сообщений лежит в основе этого общения. В этом разделе мы рассмотрим методы, используемые для отправки и получения сообщений, а также обработки различных типов данных.

Каковы практические применения WebSocket API?

WebSocket API обычно используется в приложениях, требующих передачи данных в реальном времени и взаимодействия с малой задержкой, таких как онлайн-игры, финансовые обновления в реальном времени, платформы для общения и совместной работы в реальном времени, мониторинг устройств IoT и потоковая передача событий в реальном времени.

Что такое события WebSocket и обработчики событий?

События WebSocket запускаются браузером асинхронно, указывая состояние соединения WebSocket, например открытие, закрытие или получение сообщения. Обработчики событий — это функции JavaScript, назначенные этим событиям для определения поведения приложения в ответ на эти события.

Каковы основные компоненты WebSocket API?

Основными компонентами API WebSocket являются: объект соединения WebSocket, события и обработчики событий WebSocket, отправка и получение сообщений, а также закрытие соединения WebSocket.

Как закрыть соединение WebSocket?

Чтобы закрыть соединение WebSocket, вызовите метод close() объекта WebSocket. При желании вы также можете передать код состояния и причину в качестве параметров для более подробного подтверждения закрытия.

Что такое API WebSocket?

WebSocket API — это стандартизированный протокол и API, который обеспечивает двустороннюю связь между клиентом и сервером через одно длительное соединение. Он позволяет отправлять и получать данные в режиме реального времени, обеспечивая взаимодействие с малой задержкой и эффективную связь.

Как создать соединение WebSocket?

Соединение WebSocket инициируется путем создания объекта WebSocket на стороне клиента с передачей URL-адреса сервера WebSocket в качестве параметра. Этот объект представляет соединение и предоставляет методы и свойства для взаимодействия с сервером.

Как отправлять и получать сообщения с помощью WebSocket API?

Чтобы отправить сообщение от клиента на сервер, используйте метод send() объекта WebSocket. Чтобы получать сообщения, отправленные с сервера, назначьте функцию обработчику событий onmessage объекта WebSocket, который будет запускаться при получении сообщения.

Каковы некоторые соображения по безопасности WebSocket?

Убедитесь, что вы используете протокол WebSocket Secure (WSS) для зашифрованной связи, проверки и очистки входных данных, реализации механизмов аутентификации и авторизации и защиты от атак типа «отказ в обслуживании».

Установление WebSocket-соединения

Протокол `WebSocket` работает *над* TCP.

Это означает, что при соединении браузер отправляет по HTTP специальные заголовки, спрашивая: «поддерживает ли сервер WebSocket?».

Если сервер в ответных заголовках отвечает «да, поддерживаю», то дальше HTTP прекращается и общение идёт на специальном протоколе WebSocket, который уже не имеет с HTTP ничего общего.

Установление соединения

Пример запроса от браузера при создании нового объекта `new WebSocket("ws://server.example.com/chat")`:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Origin: http://javascript.ru
Sec-WebSocket-Key: Iv8io/9s+lYFgZWcXczP8Q==
Sec-WebSocket-Version: 13
```

Описания заголовков:

GET, HostСтандартные HTTP-заголовки из URL запроса**Upgrade, Connection**Указывают, что браузер хочет перейти на websocket.**Origin**Протокол, домен и порт, откуда отправлен запрос.**Sec-WebSocket-Key**Случайный ключ, который генерируется браузером: 16 байт в кодировке Base64.**Sec-WebSocket-Version**Версия протокола. Текущая версия: 13.

Все заголовки, кроме `GET` и `Host`, браузер генерирует сам, без возможности вмешательства JavaScript.

Ответ сервера, если он понимает и разрешает `WebSocket`-подключение:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: hsBlbuDTkk24srzEOTBUlZAlC2g=
```

Здесь строка `Sec-WebSocket-Accept` представляет собой перекодированный по специальному алгоритму ключ `Sec-WebSocket-Key`. Браузер использует её для проверки, что ответ предназначается именно ему.

Затем данные передаются по специальному протоколу, структура которого («фреймы») изложена далее. И это уже совсем не HTTP.

Расширения и подпротоколы

Также возможны дополнительные заголовки `Sec-WebSocket-Extensions` и `Sec-WebSocket-Protocol`, описывающие расширения и подпротоколы (subprotocol), которые поддерживает данный клиент.

Посмотрим разницу между ними на двух примерах:

- Заголовок `Sec-WebSocket-Extensions: deflate-frame` означает, что браузер поддерживает модификацию протокола, обеспечивающую сжатие данных.

  Это говорит не о самих данных, а об улучшении способа их передачи. Браузер сам формирует этот заголовок.

- Заголовок `Sec-WebSocket-Protocol: soap, wamp` говорит о том, что по WebSocket браузер собирается передавать не просто какие-то данные, а данные в протоколах SOAP или WAMP («The WebSocket Application Messaging

Protocol»). Стандартные подпротоколы регистрируются в специальном каталоге IANA.

При наличии таких заголовков сервер может выбрать расширения и подпротоколы, которые он поддерживает, и ответить с ними.

Например, запрос:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Origin: http://javascript.ru
Sec-WebSocket-Key: Iv8io/9s+lYFgZWcXczP8Q==
Sec-WebSocket-Version: 13

Sec-WebSocket-Extensions: deflate-frame
Sec-WebSocket-Protocol: soap, wamp
```

Ответ:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: hsBlbuDTkk24srzEOTBUlZAlC2g=

Sec-WebSocket-Extensions: deflate-frame
Sec-WebSocket-Protocol: soap
```

В ответе выше сервер указывает, что поддерживает расширение `deflate-frame`, а из запрошенных подпротоколов – только SOAP.

WSS

Соединение `WebSocket` можно открывать как `WS://` или как `WSS://`.
Протокол `WSS` представляет собой WebSocket над HTTPS.

Кроме большей безопасности, у `WSS` есть важное преимущество перед обычным `WS` – большая вероятность соединения.

Дело в том, что HTTPS шифрует трафик от клиента к серверу, а HTTP – нет.

Если между клиентом и сервером есть прокси, то в случае с HTTP все WebSocket-заголовки и данные передаются через него. Прокси имеет к ним доступ, ведь они никак не шифруются, и может расценить происходящее как нарушение протокола HTTP, обрезать заголовки или оборвать передачу.

А в случае с `WSS` весь трафик сразу кодируется и через прокси проходит уже в закодированном виде. Поэтому заголовки гарантированно пройдут, и общая вероятность соединения через `WSS` выше, чем через `WS`.