### ЛЕКЦІЯ 10. Реалізація WebSocket. Порівняння HTTP та WebSocket. Архітектура WebSocket. Приклади використання WebSocket для реального часу

У сучасних веб-додатках часто виникає потреба в реальному часі передавати інформацію між сервером і клієнтом, що є ключовою вимогою для багатьох сервісів, таких як онлайн-чати, ігри, фінансові системи або системи моніторингу. Протокол WebSocket став потужним рішенням для вирішення цієї задачі, дозволяючи двосторонню комунікацію між сервером і клієнтом у реальному часі. У цій лекції ми детально розглянемо архітектуру WebSocket, порівняємо його з HTTP і наведемо приклади реального використання WebSocket у веб-додатках.

**1. Основи та порівняння HTTP та WebSocket**

**1.1 Що таке HTTP?**

HTTP (Hypertext Transfer Protocol) — це протокол, який використовується для передачі інформації між клієнтом (наприклад, веб-браузером) і сервером. Основна модель HTTP полягає в тому, що клієнт ініціює запит на сервер, а сервер відповідає на цей запит.

Однак класичний HTTP має певні обмеження, які не дозволяють створювати динамічну, інтерактивну комунікацію в реальному часі:

* **Запит-відповідь**: Клієнт надсилає запит, сервер відповідає, після чого зв'язок завершується. Якщо клієнту потрібно отримати нові дані, він повинен повторно надсилати запити (Polling або Long Polling).
* **Одностороння комунікація**: Комунікація відбувається лише тоді, коли клієнт надсилає запит.

**1.2 Що таке WebSocket?**

WebSocket — це протокол для двосторонньої комунікації між клієнтом і сервером, який працює поверх TCP. Після встановлення з'єднання, сервер і клієнт можуть обмінюватися повідомленнями без необхідності постійно ініціювати нові запити.

**1.3 Порівняння HTTP та WebSocket**

| **Параметр** | **HTTP** | **WebSocket** |
| --- | --- | --- |
| **Тип комунікації** | Одностороння (запит-відповідь) | Двостороння |
| **З'єднання** | Кожен запит відкриває нове з'єднання | Постійне з'єднання |
| **Підтримка реального часу** | Не підтримується | Підтримується |
| **Продуктивність** | Високі накладні витрати на відкриття запиту | Менші накладні витрати після встановлення з'єднання |
| **Використання ресурсів** | Вимагає більше ресурсів через часті запити | Ефективніше для тривалих зв'язків |

Основна відмінність між HTTP і WebSocket полягає в тому, що HTTP використовує модель "запит-відповідь", де кожен запит ініціює нове з'єднання, тоді як WebSocket забезпечує постійне двостороннє з'єднання між клієнтом і сервером. Це дає змогу передавати дані в реальному часі без затримок, що є критичним для багатьох сучасних веб-додатків.

**2. Архітектура WebSocket**

Архітектура WebSocket дозволяє двосторонню передачу даних без розриву з'єднання після кожного запиту. Після початкового запиту на з'єднання через HTTP (це так званий "WebSocket Handshake"), комунікація продовжується через WebSocket.

**2.1 Як працює WebSocket?**

1. **Handshake**: Початковий HTTP-запит від клієнта містить заголовок, який вказує серверу, що клієнт хоче встановити WebSocket-з'єднання.

Наприклад:

makefile

GET /chat HTTP/1.1

Host: example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

Sec-WebSocket-Version: 13

Сервер відповідає на цей запит, підтверджуючи готовність до з'єднання:

Makefile

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

1. **Постійне з'єднання**: Після успішного "Handshake" з'єднання залишається відкритим, і клієнт та сервер можуть надсилати один одному повідомлення у будь-який час без необхідності повторного встановлення з'єднання.
2. **Обмін повідомленнями**: Повідомлення надсилаються в обох напрямках у форматі "фреймів" (frames). Це дозволяє передавати як текстові, так і двійкові дані. Наприклад:
	* Клієнт може надіслати повідомлення серверу:

Arduino

Client: Hello, server!

* + Сервер може відповісти:

arduino

Server: Hello, client!

**2.2 Важливі аспекти архітектури WebSocket**

* **Низькі накладні витрати**: Після встановлення WebSocket-з'єднання немає необхідності щоразу відкривати нове з'єднання для кожного повідомлення, що зменшує накладні витрати на передачу.
* **Постійний зв'язок**: З'єднання між клієнтом і сервером може залишатися відкритим протягом тривалого часу, що забезпечує миттєву передачу повідомлень у обидві сторони.
* **Подія "onmessage"**: Для обробки повідомлень використовується подія onmessage, яка обробляє всі повідомлення, що надходять від сервера до клієнта.

**3. Приклади використання WebSocket для реального часу**

**3.1 Онлайн-чати**

Одним з найпоширеніших прикладів використання WebSocket є онлайн-чати. У цьому сценарії клієнт і сервер обмінюються повідомленнями в режимі реального часу. Завдяки WebSocket повідомлення від користувачів миттєво передаються всім іншим учасникам чату без необхідності оновлювати сторінку або надсилати нові запити.

**Приклад реалізації онлайн-чату з WebSocket:**

1. Клієнт відкриває WebSocket-з'єднання:

javascript

const socket = new WebSocket('ws://example.com/chat');

1. Клієнт надсилає повідомлення серверу:

Javascript

socket.send('Hello, everyone!');

1. Сервер приймає повідомлення та відправляє його іншим клієнтам:

javascript

socket.onmessage = function(event) {

 console.log('Message from server ', event.data);

};

**3.2 Мультиплеєрні ігри**

У реальному часі, WebSocket дозволяє гравцям взаємодіяти один з одним, передаючи дані про стан гри, рухи гравців та інші дії миттєво між клієнтами і сервером.

**Приклад ігрового сценарію з WebSocket:**

1. Сервер передає дані про координати гравця всім іншим учасникам гри:

javascript

socket.send(JSON.stringify({ player: 'Player1', position: { x: 10, y: 15 } }));

1. Клієнти отримують оновлені дані та оновлюють інтерфейс гри:

javascript

socket.onmessage = function(event) {

 const data = JSON.parse(event.data);

 updatePlayerPosition(data.player, data.position);

};

**3.3 Фінансові сервіси та біржові торги**

Багато бірж та фінансових сервісів використовують WebSocket для передачі котирувань у реальному часі. Це дозволяє користувачам отримувати оновлення про зміни цін на активи без затримки, що є критично важливим для трейдерів.

**Приклад використання WebSocket для фінансових даних:**

1. Клієнт підписується на котирування певного активу через WebSocket:

javascript

const socket = new WebSocket('ws://example.com/marketdata');

socket.send(JSON.stringify({ subscribe: 'AAPL' }));

1. Сервер надсилає оновлення про зміну ціни:

javascript

socket.onmessage = function(event) {

 const data = JSON.parse(event.data);

 console.log(`New price for AAPL: ${data.price}`);

};

ДОДАТКОВО

Архітектура WebSocket — це ключова частина сучасних веб-технологій, що дозволяє створювати інтерактивні додатки з двостороннім зв'язком між клієнтом і сервером у режимі реального часу. WebSocket забезпечує постійне з'єднання, що дозволяє обмінюватися даними без затримки, яка зазвичай виникає в протоколах, заснованих на принципі "запит-відповідь" (як у випадку з HTTP).

**1. Основні компоненти WebSocket**

**1.1 Клієнтська сторона**

Клієнт WebSocket — це програма (найчастіше браузер або мобільний додаток), яка ініціює з'єднання з сервером WebSocket і підтримує це з'єднання. Клієнтська сторона відповідає за обробку вхідних і вихідних повідомлень, а також за контроль над з'єднанням.

Ключові елементи на клієнтській стороні:

* **Ініціація з'єднання**: Клієнт ініціює з'єднання через HTTP-запит із використанням механізму "WebSocket Handshake".
* **Обробка подій**: Клієнтська програма слухає події, що надходять від сервера, та виконує відповідні дії.
* **Надсилання та отримання повідомлень**: Після встановлення з'єднання клієнт може безперервно надсилати та отримувати повідомлення.

**1.2 Серверна сторона**

Сервер WebSocket відповідає за обробку вхідних з'єднань від клієнтів та подальший обмін даними. Це може бути як спеціалізований сервер WebSocket, так і інтегрований модуль в існуючий сервер, такий як Nginx або Apache.

Ключові елементи на серверній стороні:

* **Прийом запиту на WebSocket-з'єднання**: Сервер отримує HTTP-запит від клієнта і виконує "WebSocket Handshake", щоб встановити постійне з'єднання.
* **Управління підключеними клієнтами**: Сервер може мати одночасні підключення від декількох клієнтів і управляти ними.
* **Надсилання повідомлень**: Сервер може ініціювати відправку повідомлень клієнтам без необхідності отримувати запит.

**2. WebSocket Handshake**

Процес "рукостискання" (Handshake) є важливою частиною архітектури WebSocket і виконується через стандартний HTTP-запит. Це означає, що ініціація WebSocket-з'єднання починається з відправки HTTP-запиту клієнтом і отримання відповіді від сервера.

Кроки Handshake:

1. **HTTP-запит від клієнта**: Клієнт надсилає HTTP-запит з заголовками, які вказують на перехід на WebSocket-з'єднання. Основні заголовки включають:
	* Upgrade: websocket — повідомляє серверу, що клієнт хоче перейти на WebSocket.
	* Connection: Upgrade — додаткова вказівка на бажання перейти на інший протокол.
	* Sec-WebSocket-Key — унікальний ключ, який сервер використовує для верифікації.
	* Sec-WebSocket-Version — вказує версію WebSocket, що підтримується клієнтом (звичайно 13).
2. **Відповідь сервера**: Якщо сервер підтримує WebSocket, він відповідає з заголовком HTTP 101 ("Switching Protocols"). Це означає, що сервер погоджується на зміну протоколу на WebSocket, і зв'язок тепер буде постійним.
3. **Встановлення з'єднання**: Після завершення рукостискання встановлюється постійне з'єднання, яке може використовуватись для двосторонньої передачі даних.

Приклад запиту від клієнта:

makefile

Копіювати код

GET /chat HTTP/1.1

Host: example.com

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==

Sec-WebSocket-Version: 13

Приклад відповіді від сервера:

makefile

Копіювати код

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

**3. Обмін даними в WebSocket**

Після того, як з'єднання встановлено, клієнт і сервер можуть обмінюватися повідомленнями у форматі "фреймів" (frames). Повідомлення передаються в одному з двох форматів: **текстовий** або **бінарний**.

**3.1 Фрейми**

* **Фрейм початку з'єднання** (Connection Establishment Frame): це перший фрейм, який обидві сторони передають для відкриття WebSocket-з'єднання.
* **Текстовий фрейм**: містить дані у вигляді рядка (стрічки символів UTF-8).
* **Бінарний фрейм**: використовується для передачі двійкових даних, таких як зображення, відео або файли.
* **Фрейм завершення з'єднання**: якщо одна із сторін вирішує закрити з'єднання, вона надсилає спеціальний фрейм завершення.

**3.2 Робота з фреймами**

Кожне повідомлення надсилається у вигляді фрейму, що складається з кількох полів, таких як:

* **FIN** — вказує на завершення повідомлення.
* **Opcode** — вказує тип повідомлення (наприклад, текстовий або бінарний).
* **Payload** — основні дані повідомлення.

Ці фрейми дозволяють передавати повідомлення різного розміру та типу без потреби встановлювати нове з'єднання для кожного запиту. Як тільки фрейм доставляється, інша сторона негайно обробляє його та може відправити відповідь.

**4. Двостороння комунікація**

Основною перевагою WebSocket є **двостороння комунікація**, що означає, що після встановлення з'єднання як клієнт, так і сервер можуть надсилати дані без очікування запитів.

**4.1 Клієнт -> Сервер**

Клієнт може ініціювати передачу даних у будь-який момент після встановлення з'єднання. Наприклад, клієнт надсилає повідомлення про оновлення статусу користувача на сервер.

**4.2 Сервер -> Клієнт**

Сервер, у свою чергу, може надсилати дані на клієнт, наприклад, надсилати оновлення в реальному часі про зміни в системі без потреби чекати запиту від клієнта. Це значно підвищує продуктивність і знижує навантаження на мережу.

**5. Використання протоколу WebSocket**

Протокол WebSocket найчастіше використовується в ситуаціях, де потрібне постійне оновлення даних в реальному часі. Ось декілька прикладів:

**5.1 Онлайн-чати**

WebSocket ідеально підходить для систем онлайн-чатів, де повідомлення надходять і передаються без затримки, а сервер може миттєво відправити повідомлення всім підключеним клієнтам.

**5.2 Мультиплеєрні ігри**

Багатокористувацькі онлайн-ігри використовують WebSocket для передачі інформації про дії гравців у реальному часі, дозволяючи відображати ці дії для всіх учасників гри одночасно.

**5.3 Біржові та фінансові системи**

WebSocket також використовується для передачі фінансових даних у реальному часі, таких як зміни котирувань акцій. Клієнти можуть отримувати оновлення про ціни на активи без затримки, що є критичним для трейдерів.

**5.4 Системи моніторингу**

WebSocket може використовуватися для моніторингу систем у реальному часі, наприклад, для відстеження стану серверів або інших системних показників.

**6. Переваги та недоліки WebSocket**

**6.1 Переваги**

* **Двостороння комунікація**: забезпечує обмін даними в обох напрямках без затримок.
* **Менше накладних витрат**: не потрібно встановлювати нове з'єднання для кожного запиту, як у випадку з HTTP.
* **Реальний час**: ідеальний для додатків з потребою в оновленнях даних у реальному часі.
* **Масштабованість**: можливість підтримувати одночасні з'єднання з великою кількістю клієнтів.

**6.2 Недоліки**

* **Складність реалізації**: потребує додаткової конфігурації серверів і клієнтської сторони.
* **Безпека**: постійне з'єднання може бути більш вразливим для атак, таких як DoS або CSRF.
* **Обмеження в мережах**: не всі мережі або проксі-сервери підтримують протокол WebSocket.

WebSocket є потужним інструментом для створення інтерактивних додатків у реальному часі. Він забезпечує двосторонню комунікацію з меншими накладними витратами порівняно з HTTP та дозволяє досягати високої продуктивності. Завдяки WebSocket стали можливими багато сучасних додатків, таких як онлайн-чати, ігри, фінансові сервіси та інші. Розуміння архітектури WebSocket і його можливостей допомагає ефективніше розробляти сучасні веб-додатки та сервіси для реального часу.