The "Server-Sent Events Documentation Repository"

Release 0.1

Hector Miranda

Jul 04, 2023

CONTENTS

```mermaid graph LR A[Client] –> B((EventSource)) B – Subscribe –> C[Server] C – Send events –> B B – Event handling –> A
```

CHAPTER

ONE

CONTENTS

1.1 Table of Contents

1.2 Introduction to Server-Sent Events

1.2.1 Server-Sent Events (SSE)

Server-Sent Events (SSE) is a standard that allows a web page to receive updates from a server over a single, long-lived HTTP connection. It provides a simple and efficient way to push data from the server to the client in real-time.

1.3 Receiving events from the server

To receive events from the server using SSE, the client needs to establish a connection and listen for incoming events. This can be done by creating an EventSource instance.

1.4 Creating an EventSource instance

To create an EventSource instance in JavaScript, you can use the following code:

```
const eventSource = new EventSource('/events');
```

In the above code, */events* represents the URL from which the server sends events. You can replace it with the appropriate server endpoint.

1.5 Listening for message events

The *message* event is fired when the server sends a new event. To listen for message events, you can use the following code:

```
eventSource.addEventListener('message', function(event) {
  const data = event.data;
  // Handle the received event data
})
```

1.6 Listening for custom events

Apart from the standard *message* event, the server can also send custom events. To listen for custom events, you can use the following code:

```
eventSource.addEventListener('myevent', function(event) {
  const data = event.data;
  // Handle the received custom event data
})
```

In the above code, *myevent* represents the name of the custom event sent by the server.

1.7 Sending events from the server

The server can send events to connected clients using the SSE protocol. The events are typically sent in a specific format, which includes the event type, data, and optional fields like retry interval.

1.8 Error handling

In case of errors or connection issues, the server can send an error event to the client. The client can handle these errors by listening for the *error* event.

```
eventSource.addEventListener('error', function(event) {
  const error = event.error;
  // Handle the error
})
```

1.9 Closing event streams

To close the connection and stop receiving events, the client can call the *close()* method on the EventSource instance.

```
eventSource.close();
```

1.10 Event stream format

The event stream format used by Server-Sent Events is a text-based format with a specific structure. It consists of lines of text, where each line can represent an event field such as event type, data, or retry interval.

Here's an example of a basic event stream:

```
event: message
data: Hello, world!

event: customEvent
data: Some custom data
```

The above example demonstrates two events: a standard *message* event and a custom *customEvent* with their respective data.

These are the main concepts of using Server-Sent Events. By following these guidelines, you can easily implement real-time updates in your web applications.

1.11 Table of Contents

1.12 Introduction to Server-Sent Events

1.12.1 Server-Sent Events (SSE)

Server-Sent Events (SSE) is a standard that allows a web page to receive updates from a server over a single, long-lived HTTP connection. It provides a simple and efficient way to push data from the server to the client in real-time.

1.13 Receiving events from the server

To receive events from the server using SSE, the client needs to establish a connection and listen for incoming events. This can be done by creating an EventSource instance.

1.14 Creating an EventSource instance

To create an EventSource instance in JavaScript, you can use the following code:

```
const eventSource = new EventSource('/events');
```

In the above code, */events* represents the URL from which the server sends events. You can replace it with the appropriate server endpoint.

1.15 Listening for message events

The *message* event is fired when the server sends a new event. To listen for message events, you can use the following code:

```
eventSource.addEventListener('message', function(event) {
  const data = event.data;
  // Handle the received event data
})
```

1.16 Listening for custom events

Apart from the standard *message* event, the server can also send custom events. To listen for custom events, you can use the following code:

```
eventSource.addEventListener('myevent', function(event) {
  const data = event.data;
  // Handle the received custom event data
})
```

In the above code, *myevent* represents the name of the custom event sent by the server.

1.17 Sending events from the server

The server can send events to connected clients using the SSE protocol. The events are typically sent in a specific format, which includes the event type, data, and optional fields like retry interval.

1.18 Error handling

In case of errors or connection issues, the server can send an error event to the client. The client can handle these errors by listening for the *error* event.

```
eventSource.addEventListener('error', function(event) {
  const error = event.error;
  // Handle the error
})
```

1.19 Closing event streams

To close the connection and stop receiving events, the client can call the *close()* method on the EventSource instance.

```
eventSource.close();
```

1.20 Event stream format

The event stream format used by Server-Sent Events is a text-based format with a specific structure. It consists of lines of text, where each line can represent an event field such as event type, data, or retry interval.

Here's an example of a basic event stream:

```
event: message
data: Hello, world!

event: customEvent
data: Some custom data
```

The above example demonstrates two events: a standard *message* event and a custom *customEvent* with their respective data.

These are the main concepts of using Server-Sent Events. By following these guidelines, you can easily implement real-time updates in your web applications.