**Чтобы понять, какие события применять к разным типам потоков в Node.js, важно знать, как работает каждый тип потока (чтение, запись, дуплексные и трансформирующие потоки) и какие события характерны для каждого из них.**

**Типы потоков:**

1. **Потоки чтения** (Readable Stream): Потоки, которые **читают данные** (например, из файлов, сетевых соединений).
   * Пример: fs.createReadStream().
2. **Потоки записи** (Writable Stream): Потоки, которые **записывают данные** (например, в файлы, сетевые соединения).
   * Пример: fs.createWriteStream().
3. **Дуплексные потоки** (Duplex Stream): Потоки, которые могут **и читать, и записывать** данные (например, сетевые соединения).
   * Пример: net.Socket.
4. **Трансформирующие потоки** (Transform Stream): Потоки, которые могут **и читать, и записывать** данные, при этом трансформируя их по мере обработки.
   * Пример: zlib.createGzip() для сжатия данных.

**Основные события для каждого типа потоков:**

**1. Чтение (Readable Stream):**

Применяй следующие события к потокам чтения:

* **data**: Срабатывает, когда поток выдает кусок данных.
* **end**: Когда поток завершил чтение всех данных (например, файл полностью прочитан).
* **close**: Когда поток закрыт и все ресурсы освобождены (может быть после успешного завершения или при ошибке).
* **error**: Когда произошла ошибка во время чтения данных.

Пример для потока чтения:

javascript

**код:**

const readStream = fs.createReadStream("file.txt");

readStream.on("data", (chunk) => {

console.log(`Received chunk: ${chunk}`);

});

readStream.on("end", () => {

console.log("Reading finished");

});

readStream.on("close", () => {

console.log("Stream closed");

});

readStream.on("error", (err) => {

console.error("Error:", err);

});

**2. Запись (Writable Stream):**

Применяй следующие события к потокам записи:

* **drain**: Когда поток готов принять новые данные после того, как буфер освободился.
* **finish**: Когда все данные были записаны и поток завершил работу.
* **close**: Когда поток закрыт и все ресурсы освобождены.
* **error**: Когда произошла ошибка во время записи данных.

**Пример для потока записи:**

**Код:**

const writeStream = fs.createWriteStream("output.txt");

writeStream.on("finish", () => {

console.log("Write finished");

});

writeStream.on("close", () => {

console.log("Stream closed");

});

writeStream.on("error", (err) => {

console.error("Error:", err);

});

writeStream.end("Some data to write");

**3. Дуплексные и трансформирующие потоки:**

Они поддерживают как события для чтения, так и события для записи, так как они могут выполнять обе операции.

* Для чтения используются: **data**, **end**, **close**, **error**.
* Для записи: **drain**, **finish**, **close**, **error**.

**Пример для дуплексного потока:**

javascript

**код:**

const net = require("net");

const socket = new net.Socket();

// Чтение данных

socket.on("data", (chunk) => {

console.log("Received data:", chunk);

});

// Запись данных

socket.write("Hello, server!");

socket.on("finish", () => {

console.log("Write finished");

});

// Ошибка

socket.on("error", (err) => {

console.error("Error:", err);

});

**Резюме: Как понять, какие события применять к потоку?**

* **Чтение (Readable Stream)**: Применяй события data, end, close, error.
* **Запись (Writable Stream)**: Применяй события drain, finish, close, error.
* **Дуплексные и трансформирующие потоки**: Поддерживают события для обоих типов — и для чтения, и для записи.

Таким образом, при работе с потоками важно понимать, что потоки чтения и записи имеют свои уникальные события, а дуплексные и трансформирующие потоки объединяют оба набора событий.