Как создать собственный OPC UA сервер с использованием Node.js

Pasha ⋮⋮ 18.12.2024

suprunchuk 18 дек 2024 в 22:42

4 мин

2.1K

Промышленное программирование*Программирование*

Туториал

В данной статье мы рассмотрим процесс создания простого OPC UA сервера с использованием популярной библиотеки node-opcua.

Что такое OPC UA?

OPC UA (Unified Architecture) — это стандарт промышленной автоматизации, который обеспечивает обмен данными между оборудованием, системами управления и приложениями. Он активно используется в системах IoT и промышленности 4.0.

Зачем использовать Node.js?

Node.js — это платформа, основанная на JavaScript, которая идеально подходит для разработки серверных приложений благодаря своей асинхронной модели ввода-вывода. С помощью библиотеки **node-opcua** мы можем быстро разрабатывать надежные OPC UA серверы.

Подготовка к работе

Перед началом убедитесь, что Node.js установлен на вашем компьютере. Установить Node.js можно, следуя инструкциям на официальном сайте.

Создание проекта

1. Создайте новую директорию для проекта:

```
mkdir myserver
cd myserver
```

2. Инициализируйте проект Node.js:

```
npm init -y
```

3. Установите библиотеку **node-opcua**:

```
npm install node-opcua --save
```

Теперь проект готов к разработке.

Написание скрипта сервера

Создайте файл `server.js` и начните с подключения необходимых модулей:

```
const { OPCUAServer, Variant, DataType, StatusCodes } = require("node-
opcua");
const os = require("os");
```

Шаг 1: Создание экземпляра сервера

Инициализируйте сервер:

```
const server = new OPCUAServer({
  port: 4334,
  resourcePath: "/UA/MyLittleServer",
  buildInfo: {
    productName: "MySampleServer",
    buildNumber: "1",
    buildDate: new Date()
  }
});
```

Шаг 2: Инициализация сервера

Инициализация сервера выполняется асинхронно:

```
(async () => {
  await server.initialize();
  console.log("Сервер инициализирован.");
})();
```

Шаг 3: Добавление объектов и переменных

Мы создадим объект `MyDevice` и добавим в него переменные.

Добавление объекта

```
const addressSpace = server.engine.addressSpace;
const namespace = addressSpace.getOwnNamespace();
const device = namespace.addObject({
```

```
  organizedBy: addressSpace.rootFolder.objects,
  browseName: "MyDevice"
});
```

Добавление переменной для чтения

```
let variable1 = 1;
setInterval(() => {
  variable1 += 1;
}, 500);
namespace.addVariable({
  componentOf: device,
  browseName: "MyVariable1",
  dataType: "Double",
  value: {
    get: () => new Variant({ dataType: DataType.Double, value: variable1 })
  }
});
```

Добавление переменной для чтения и записи

```
let variable2 = 10.0;
namespace.addVariable({
  componentOf: device,
  browseName: "MyVariable2",
  nodeId: "ns=1;s=MyVariable2",
  dataType: "Double",
  value: {
    get: () => new Variant(
      { dataType: DataType.Double, value: variable2 }),
    set: (variant) => {
      variable2 = parseFloat(variant.value);
      return StatusCodes.Good;
    }
  }
});
```

Добавление переменной для отображения свободной памяти

```
namespace.addVariable({
  componentOf: device,
  browseName: "FreeMemory",
  nodeId: "s=free_memory",
  dataType: "Double",
  value: {
```

```
    get: () => new Variant({
      dataType: DataType.Double,
      value: os.freemem() / os.totalmem() * 100.0
    })
  }
});
```

Шаг 4: Запуск сервера

После настройки всех переменных запустите сервер:

```
(async () => {
  await server.start();
  console.log("Сервер запущен.");
  console.log("URL:",
  server.endpoints[0].endpointDescriptions()[0].endpointUrl);
})();
```

Тестирование сервера

Сохраните файл `server.js` и выполните его:

```
node server.js
```

После запуска сервер будет доступен по адресу, указанному в консоли. Теперь его можно подключить к OPC UA клиенту для тестирования.

Полный код доступен здесь:

▶ Скрытый текст

Заключение

Мы создали базовый OPC UA сервер с помощью Node.js и библиотеки **node-opcua**. Вы можете расширить этот сервер, добавляя собственные объекты и переменные, чтобы удовлетворить ваши требования.