EPICS-TCA, a Node.js Library for EPICS Channel Access and PV Access Client

Hao Hao

Oak Ridge National Laboratory

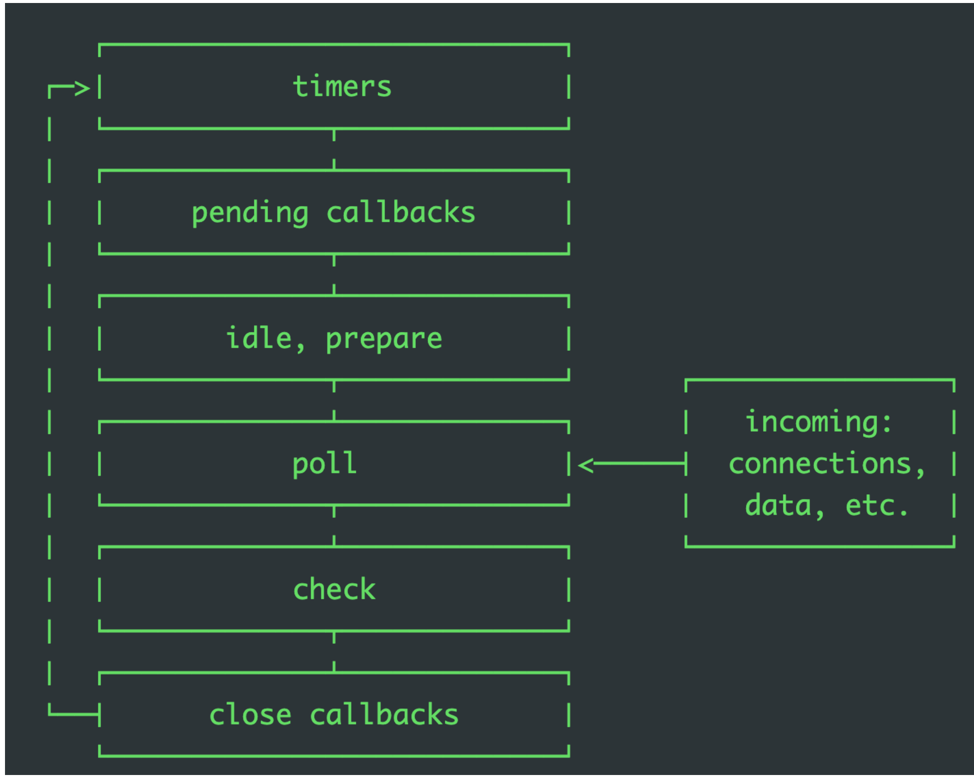
April 27, 2023

About Node.js

- Node.js is a "desktop" version of JavaScript
  - Independent of web browser
  - Allowed to use resources on the computer: TCP/UDP protocols, read/write files, …
  - Derived from Google V8 engine, which is used in Chromium (Google Chrome, Microsoft Edge, …)
  - Cross platform
  - Can be possibly converted to a web application
  - Development environment
    - npm
    - yarn
    - …

Asynchronous Model in Node.js

- Event loop model enables rich asynchronous features
  - Promise
  - EventEmitter
  - await-async
- Single threaded
  - All operations in a function are atomic
  - Can be multi-threaded by using e.g. worker thread



https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick

Asynchronous Model in EPICS-TCA

- EPICS applications are usually not computationally intensive, and most functions are asynchronous: caget(), camonitor() …
- TCA uses Promise and async-await to "block" the asynchronous call, then resolve the Promise to proceed

```
let promise = new Promise<any>((resolve, reject) => {
    resolveFunc = resolve;
    rejectFunc = reject;
});
```

```
// handle CA_PROTO_SEARCH (0x06) reply message
private handle_CA_PROTO_SEARCH = async (message: Buffer, info: Record<string, any>) => {
    const serverTcpPort: number = message.readUInt16BE(CA_MSG_HEADER_OFFSETS.dataType);
    const serverAddress: string = info.address;
    const channelClientID: number = message.readUInt32BE(CA_MSG_HEADER_OFFSETS.parameter2);

    // if channel is not resolved yet
    if (this.context.unresolvedChannelsByClientID[channelClientID.toString()] !== undefined) {
        const channel = this.context.unresolvedChannelsByClientID[channelClientID.toString()];
        // lift the blocking of Channel.createTCP(), next step is to create the TCP based on
        // the info received in this UDP packet
        channel.promises.resolve("CA_PROTO_SEARCH", [serverAddress, serverTcpPort]);
        return;
    } else {
        // the channel is already resolved: do nothing
    }
};
```

Blocked in async function

Lift the blocking

```
// resolve after receiving CA_PROTO_SEARCH reply via UDP
const [serverAddress, serverTcpPort] = await this.promises.get("CA_PROTO_SEARCH");
```

Project Overview

- Uses “npm” to manage the project
  - Easily adopt 3rd party libraries
- Uses TypeScript for coding
  - A “typed” JavaScript, improve development efficiency and avoid runtime errors
- Most code are encapsulated inside classes

```
"dependencies": {
  "buffer": "^6.0.3",
  "jest": "^29.3.1",
  "ts-jest": "^29.0.3",
  "ws": "^8.11.0"
}
```

```
export type type_dbrData = Record<string, any> & {
    value: string | string[] | number | number[];
};
```

Classes

- class Context
  - Initialize program: create CA repeater thread, add UDP and TCP listeners
  - Process Beacon
  - Manage channels
- class UDPTransport, class TCPTransport
  - Send and receive UDP and TCP messages, invoke listeners upon a new message arrival
- class Channel
  - Lifecycle management of a channel: search, connect, get, put, and destroy
  - Encode and decode most CA messages
  - Manage monitors (class ChannelMonitor)
- class ChannelMonitor
  - Subscribe, unsubscribe a CA monitor.

Application – get a value

- Create an NPM project: “npm init --yes”
- Install epics-tca package: “npm install epics-tca”
- Create a Node.js program, e.g. test01.js, with the following contents

```
const epicsTca = require("epics-tca");
const context = new epicsTca.Context({});
context.initialize()
context.createChannel("val1").then((channel) => {
    channel.get().then(console.log).then(() => {
        context.destroyHard();
    });
});
```

- Run the program:

```
$ node test01.js
{ value: 445 }
```

Application – tcaGet()

- Using TypeScript and async-await
  - More robust and intuitive

```
import {Context, type_dbrData} from 'epics-tca';

const tcaGet = async (name: string): Promise<type_dbrData | undefined> => {
    const context = new Context({});
    context.initialize();
    const channel = await context.createChannel(name);
    const result = await channel?.get();
    context.destroyHard();
    return result;
}

tcaGet("val1").then(console.log)
```

```
$ node test02.js
{ value: 724 }
```

Application – tcaMonitor()

- ChannelMonitor can be created by a Channel, with a callback function and desired data type

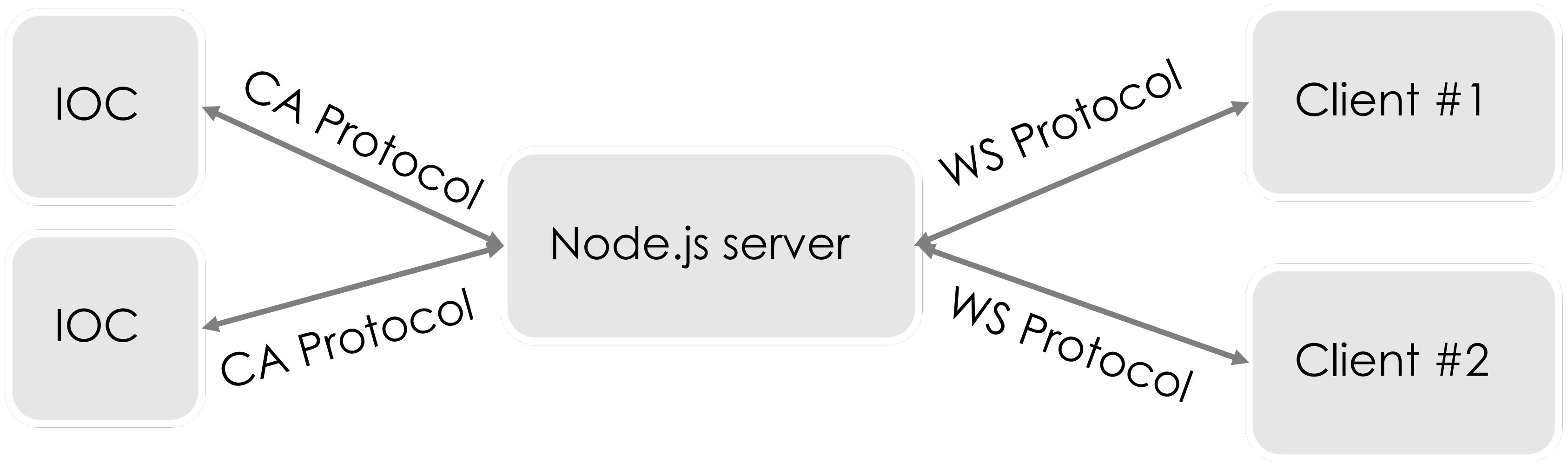
```
import {Context, type_dbrData} from 'epics-tca';

const tcaMonitor = async (name: string): Promise<void> => {
    const context = new Context({});
    context.initialize();
    const channel = await context.createChannel(name);
    const dbrType = channel?.getDbrTypeNum_TIME();
    const monitor = await channel?.createMonitor(() => {
        console.log(channel.getDbrData());
    }, dbrType)
    monitor?.subscribe();
}

tcaMonitor("val1")
```

```
$ node test02.js
{
  status: 0,
  severity: 0,
  secondsSinceEpoch: 1050258716,
  nanoSeconds: 82980000,
  value: 174
}
{
  status: 0,
  severity: 0,
  secondsSinceEpoch: 1050258716,
  nanoSeconds: 181029000,
  value: 175
}
```

Application – a WebSocket Server based on Node.js



- Start a WebSocket sever:

```
import {WSServer} from "epics-tca";
const server = new WSServer();
```

- Start client (Node.js code):

```
import WebSocket from "ws";
const ws = new WebSocket("ws://localhost:8080");
ws.on("open", () => {
    ws.send("MONITOR val1");
});
ws.on('message', (data) => {
  console.log('received: %s', JSON.parse(data.toString()));
});
```

```
$ node test02.js
received: { channelName: 'val1', value: 977, webSocketCommand: 'GET' }
received: {
  channelName: 'val1',
  value: 978,
  subscriptionId: 3,
  webSocketCommand: 'MONITOR'
}
received: {
  channelName: 'val1',
  value: 979,
  subscriptionId: 3,
  webSocketCommand: 'MONITOR'
}
```

Performance

- 100,000 PVs, connect to a local soft IOC, read value, and close
  - ~ 6 s on M1 Pro Max Processor, ~ 60 microseconds per channel
  - 1.3 GB memory, about 13.6 kB/channel
- 50,000 updates each second, monitor values
  - 85% CPU, ~ 17 microseconds for each update
  - 750 MB memory, ~ 15.4 kB/channel

Outlook

- CA Protocol
  - IPv6 …
- PV Access Protocol
  - Introspection data encoding/decoding is almost finished
  - Network protocol is the next
- Optimize performance
  - Reduce memory footprint: carefully design data structure and logic, less GC
- More unit and integrated tests

Thanks!

https://code.ornl.gov/1h7/epics-tca