How to use GET method to access remote api using XMLHttpRequest with React client and Express server

井民全, Jing, mqjing@gmail.com

You can use XMLHttpRequest easily to send GET requests with parameters to the server. Here is an example, I used React as a client and Express as a server.

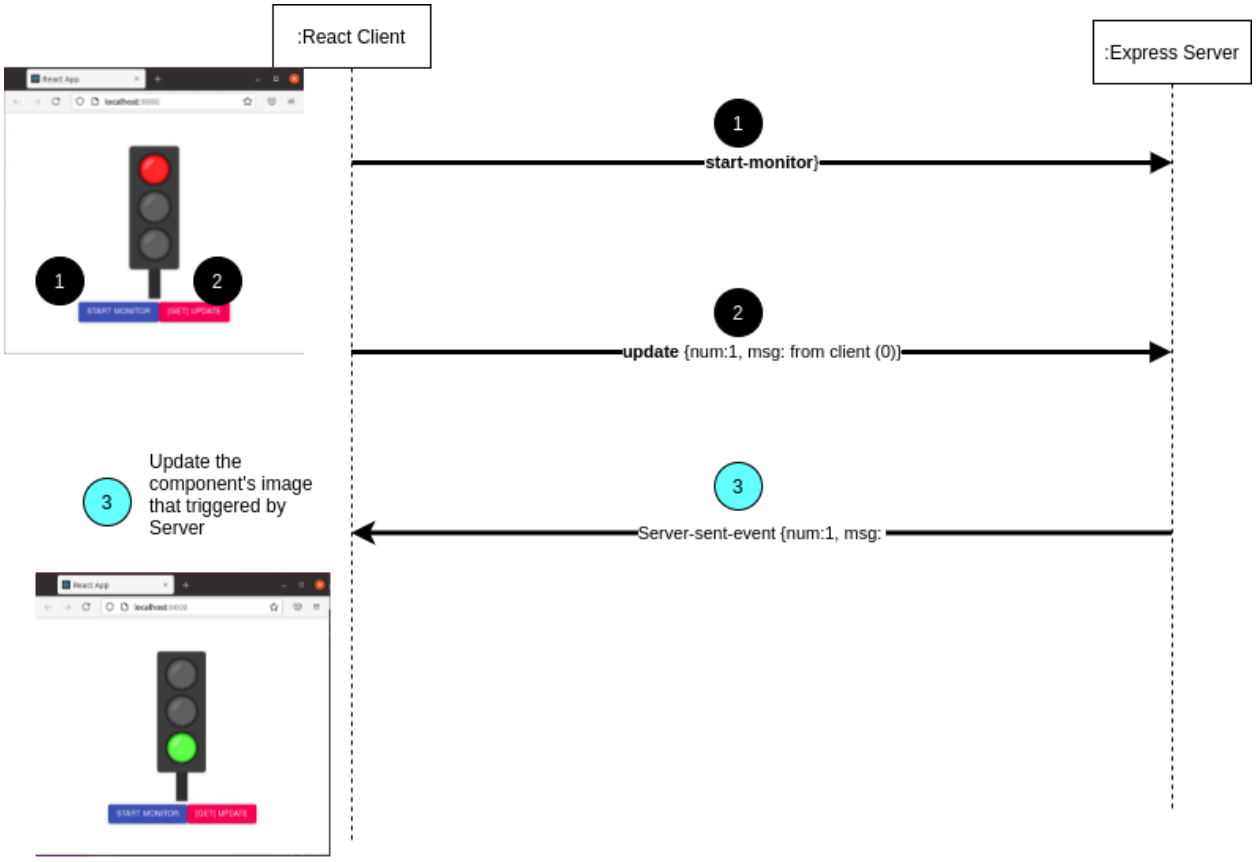


Fig. The system flow. (Edit)

Enjoy!

By Jing.

Quick

Client

```
let url =   'status-realtime/update' + '?parm1=value1&parm2=value2';
let xhr = new XMLHttpRequest();
xhr.open('GET', url);
xhr.send();
```

Ex

```
let url =   'status-realtime/update' + '?num=1&msg=hello';
let xhr = new XMLHttpRequest();
xhr.open('GET', url);
xhr.send();
```

Server

File: routes/xxxapi.ts

```
// api: /update
router.get('/update', function(req:Request, res:Response, next) {
   // Step 1: Retrieve the parameters from client
   let receivedNum = req.query.num;   // get the para: num
   let receivedMsg = req.query.msg;     // get the para: msg

   status.num = parseInt(receivedNum as string); // update
   status.msg = receivedMsg as string; // update
   console.log('/update. status = ' + JSON.stringify(status));

   // Step 2: Response to client
   res.send('ok')
});
```

Table of contents

1. Client (React)

File: App.tsx

```
// Pass object type, return the GET parameters format
function getParamUrl<T>(para:T):string{
  var url = '?';
  let bFirst = false;

  for (var key in para) {
     if ((para as Object).hasOwnProperty(key)) {
        url += (bFirst ? '&' : '') + key + "=" + para[key as keyof T];
     }
     bFirst = true;
  }

  return url;
}

function App() {
  const urlUpdateApi:string = 'status/update';  // API: status/update

  interface Status {
    num: number;
    msg: string;
  }

  let mystatus:Status = {num: 0, msg: 'init'};

  const onUpdate = async () => {
    // Step 1: Update status
    mystatus.num++;
    mystatus.msg = 'from onUpdate. (`${mystatus.num}`)';
    console.log('[client] onUpdate, ' + JSON.stringify(mystatus));

    // Step 2: Send to the server
    let url = urlUpdateApi + getParamUrl(mystatus);
    let xhr = new XMLHttpRequest();
    xhr.open('GET', url);
    xhr.send();

    xhr.onload = function() {
      if (xhr.status != 200) { // HTTP error?
        console.log('[client] Error: xhr.status != 200' + xhr.status)
        return;
```

```
    }
    // get the response from xhr.response
    console.log('[client] xhr.response = ' + xhr.response);
  };
  xhr.onprogress = function(event) {
    var percentComplete = event.loaded / event.total * 100;
    console.log(`[client] xhr progress ${percentComplete}% complete.`);
  };

  xhr.onerror = function() {
    console.log('[client] handle non-HTTP error (e.g. network down)');
  };
 }

return (
  <div className="App">
    <header className="App-header">
      <p>
        <Button onClick={onUpdate} variant="contained" color="secondary"> [GET] Update
Status </Button>
      </p>
    </header>
  </div>
 );
)
```

2. Server (Express)

File: routes/status-api.ts

```
import express, {Request, Response} from 'express';

interface Status  {
  num: number;
  msg: string;
}

let status:Status = {num: 0, msg: 'init'};

var router = express.Router();

// api: /status/update
router.get('/update', function(req:Request, res:Response, next) {
  // Step 1: Retrieve the parameters from client
```

```
    let receivedNum = req.query.num;   // get the para: num
    let receivedMsg = req.query.msg;   // get the para: msg
    status.num = parseInt(receivedNum as string); // update
    status.msg = receivedMsg as string; // update
    console.log('/update. status = ' + JSON.stringify(status));

    // Step 2(opt): Sent back to client (SSE part)
    if (resStream == null){
        let strMsg:string = '[server] Warn::SSE do not setup, yet. Please click [Start Monitor]. Act:
do not thing.';
        console.log(strMsg);
        res.send(strMsg)
        return;
    }
    resStream.write(`data: ${JSON.stringify(status)}\n\n`);

    // Step 3: Response to client
    let strMsg:string = '[server] status update ok.';
    console.log(strMsg);
    res.send(strMsg)
});

// api: /status/start-monitor: Create server-sent-event back channel for client update status
let resStream:Response | null = null;
router.get('/start-monitor', function(req:Request, res:Response, next) {
    if (resStream != null){
        console.log('[Error] SSE already setup. Act: do not thing.');
        return;
    }

    console.log('start-monitor');
    resStream = res;
    resStream.setHeader('Cache-Control', 'no-cache');
    resStream.setHeader('Content-Type', 'text/event-stream');
    resStream.setHeader('Access-Control-Allow-Origin', '*'); // ok for local
    resStream.setHeader('Connection', 'keep-alive');
    resStream.flushHeaders(); // flush the headers to establish SSE with client
    resStream.write(`data: ${JSON.stringify(status)}\n\n`);


    // If client closes connection, stop sending events
    resStream.on('close', () => {
        console.log('client dropped me');
        // clearInterval(interValID);
        (resStream as Response).end();
        resStream = null;
        console.log('res.end()');
```

```
    });
});


module.exports = router;
```

File: index.ts

```
import express from 'express';
import path from 'path';


var statusRouter = require('./routes/status-api');
// var cors = require("cors");

const ReactBuildPath:string = '../client/build';  // the React client build folder
const app = express();

// handle cors
// app.use(cors());
// Register a middleware to serve files from the React production build folder: ../build
app.use(express.static(path.join(__dirname, ReactBuildPath)));

// Register a router statusAPIRouter on the virtual path /statusI.
app.use('/status', statusRouter);

// Register a middleware that handle GET request on the '/' to response React production
index.html
app.get('/', function (req, res) {
  res.sendFile(path.join(__dirname, ReactBuildPath + '/index.html'));
});

app.listen(9000);
console.log('Server: ok')
console.log('Test home: gio open http://localhost:9000/')
```

3. Install, Run, Test

3.1. Install

```
yarn --cwd ./server install
```

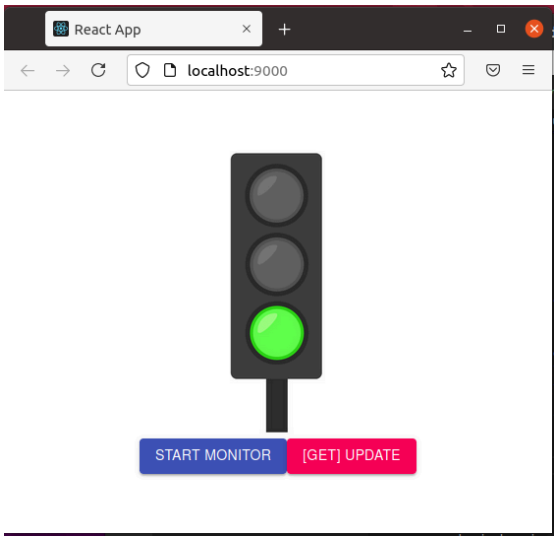
```
yarn --cwd ./client install
yarn --cwd ./client build
```

3.2. Run

```
cd server
tsc
node index.js
```

3.3. Test

```
gio http://localhost:9000
```



3.4. Make clean

```
yarn --cwd ./server clean
yarn --cwd ./client clean
```

3.4.1. Clean script: Server

File: ./server/package.json

```
"clean": "rm -fr node_modules; find . -name \"*.js\" -type f|xargs rm -f; find . -name \"*.map\" -type f|xargs rm -f"
```

3.4.2. Clean script: Client

File: ./client/package.json

```
"clean": "(rm -fr node_modules; rm -fr build; find . -name \"*.js\" -type f|xargs rm -f; find . -name \"*.map\" -type f|xargs rm -f)"
```

4. References

1. Express 4.x - API Reference, http://expressjs.com/en/api.html#req.query
2. How To Use the req Object in Express, https://www.digitalocean.com/community/tutorials/nodejs-req-object-in-expressjs
3. XMLHttpRequest, https://javascript.info/xmlhttprequest
4. Using XMLHttpRequest, https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest

5. Further Information

1. [react, express, deploy] How to deploy the React App with Express (view)
2. [react, express, restfulAPI] How to implement a restfulAPI using Express and React (view)
3. [react, express, sse] How to create server that streams events to the front-end client using Express and React**** (view), (blog)
4. [html, xhr, get] How to use GET method to access remote api using XMLHttpRequest with React client and Express server*** (view) <--- this document