از کدامیک باید استفاده کنیم - فرانت کست : فرانت کست ،Fetch و Axios

اگر برنامه نویس جاوااسکریپت هستید حتما با یکی از APIهای Axios و Fetch کارکرده‌اید و با آنها آشنا هستید. هرچند که هرکدام از این APIها کارکرد مخصوص خود را دارند اما سوال اصلی این است که بهتر است از کدام یک از آنها استفاده کنیم؟ برخی از توسعه دهندگان به خاطر سهولت پیاده‌سازی Axios، استفاده از آن را ترجیح می‌دهند. اما برخی دیگر این API را بسیار سطح بالا دانسته و استفاده از آن را ضروری نمی‌دانند. از طرفی APIای مانند Fetch علاوه بر اینکه ویژگی‌های کلیدی Axios را دارد، به راحتی در کلیه مرورگرهای مدرن و به روز دنیا نیز در دسترس است.

در این مقاله Fetch و Axios را با یکدیگر مقایسه خواهیم کرد تا بدانیم که چطور می‌توان از آن‌ها برای انجام وظایف مختلف استفاده کرد. امیدواریم در پایان مقاله، درک بهتری از هر دو API داشته باشید. لطفا تا انتهای مطلب با ما همراه شوید.

دستورات پایه

اجازه بدهید قبل از اینکه درباره ویژگی‌های پیشرفته Axios صحبت کنیم، دستورات پایه آن را با Fetch مقایسه کنیم. در قطعه کد زیر مشاهده می‌کنید که چگونه می‌توان با استفاده از Axios یک درخواست POST را با Headerهای سفارشی به یک نشانی اینترنتی ارسال کرد.

```
// axios

const options = {
  url: 'http://localhost/test.htm',
  method: 'POST',
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json;charset=UTF-8'
  },
  data: {
    a: 10,
    b: 20
  }
};

axios(options)
  .then(response => {
    console.log(response.status);
  });
```

اکنون همین کد را که با Fetch پیاده سازی شده و در نهایت نتیجه یکسانی تولید می‌کند، ببینید.

```
// fetch()

const url = 'http://localhost/test.htm';
const options = {
  method: 'POST',
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json;charset=UTF-8'
  },
```

```
  body: JSON.stringify({
    a: 10,
    b: 20
  })
};

fetch(url, options)
  .then(response => {
    console.log(response.status);
  });
```

با مقایسه دو قطعه کد بالا می‌توان موارد زیر را دریافت:

- برای ارسال داده Fetch از ویژگی body و Axios از ویژگی data استفاده می‌کند.
- داده‌های Fetch به حالت رشته‌ای از کاراکترها هستند.
- آدرس اینترنتی در Fetch به صورت یک آرگومان درنظر گرفته شده است اما در Axios به عنوان شی‌ای از options است.

سوابق سازگاری APIها با مرورگرها

اگر نگاهی به سابقه استفاده از این دو API در مرورگرهای مختلف داشته باشیم، درمی‌یابیم که Axios از مروگرهای بسیاری پشتیبانی می‌کند و حتی بر روی مرورگرهای قدیمی مثل IE۱۱ بدون کوچکترین مشکل کار می‌کند. اما Fetch اینگونه نیست و فقط از مرورگرهای گوگل کروم نسخه ۴۲ به بالا، فایرفاکس نسخه ۳۹ به بالا، edge نسخه ۱۴ به بالا و سافاری ۱۰٫۱ به بالا پشتیبانی می‌کند.

مدت زمان پاسخگویی

سادگی تنظیم زمان‌بندی در Axios یکی دیگر از دلایلی است که برخی از توسعه دهندگان را ترغیب به استفاده از این API می‌کند. در Axios شما می‌توانید با استفاده از ویژگی timeout در شی پیکربندی شده، برای تنظیم بازه زمانی قبل از قطع درخواست برحسب میلی‌ثانیه استفاده کنید. برای مثال:

```
axios({
  method: 'post',
  url: '/login',
  timeout: 4000,    // 4 seconds timeout
  data: {
    firstName: 'David',
    lastName: 'Pollock'
  }
})
.then(response => {/* handle the response */})
.catch(error => console.error('timeout exceeded'))
```

Fetch عملکرد مشابهی را از طریق واسط AbortController فراهم می‌کند. با این حال استفاده از این واسط به سادگی کار با Axios نیست.

```
const controller = new AbortController();
const options = {
  method: 'POST',
  signal: controller.signal,
  body: JSON.stringify({
    firstName: 'David',
    lastName: 'Pollock'
  })
};
const promise = fetch('/login', options);
const timeoutId = setTimeout(() => controller.abort(), 4000);

promise
  .then(response => {/* handle the response */})
  .catch(error => console.error('timeout exceeded'));
```

در قطعه کد بالا با استفاده از AbortController.AbortController() شی AbortController را که به ما اجازه قطع درخواست را می‌دهد ایجاد کردیم. signal تنها ویژگی خواندنی AbortController است که وسیله‌ای برای برقراری ارتباط با یک درخواست یا قطع آن است. اگر سرور در کمتر از ۴ ثانیه پاسخ ندهد، controller.abort فراخوانی می‌شود و عملیات خاتمه می‌یابد.

برای درک بهتر مدیریت درخواست‌ها با استفاده از شی AbortController، قسمت ۹۳ دوره جامع MERN Stack را بررسی کنید.

تبدیل خودکار داده‌های JSON

همان طور که قبلاً گفتیم، Axios به طور خودکار داده‌ها را هنگام ارسال درخواست به صورت رشته‌ای ارسال می‌کند. اگر چه شما می‌توانید این حالت پیش‌فرض را نادیده بگیرید و مکانیزم تغییر متفاوتی را تعریف کنید. با این حال، هنگام استفاده از Fetch باید به صورت دستی این کار را انجام دهید. دو قطعه کد زیر را باهم مقایسه کنید.

```
// axios
axios.get('https://api.github.com/orgs/axios')
  .then(response => {
    console.log(response.data);
  }, error => {
    console.log(error);
  });

// fetch()
fetch('https://api.github.com/orgs/axios')
  .then(response => response.json())    // one extra step
  .then(data => {
    console.log(data)
  })
  .catch(error => console.error(error));
```

تبدیل اتوماتیک داده‌ها ویژگی خوبی است، اما باز هم چیزی نیست که بتوانید با Fetch آن را انجام دهید.

پیگیری درخواست‌های HTTP

یکی از ویژگی‌های مهم Axios، توانایی آن در پیگیری درخواست‌های HTTP است. پیگیری درخواست‌های HTTP در مواقعی که نیاز به بررسی یا تغییر درخواست‌های HTTP از خود برنامه به سرور و یا بالعکس (مثلا ورود به سیستم، تایید اعتبار و غیره) مفید است. با استفاده از رهگیرها دیگر لازم نیست که برای هر درخواست HTTP کدی جداگانه بنویسید. در قطعه کد زیر نحوه درخواست یک رهگیر در Axios را می‌توانید ببینید.

```
axios.interceptors.request.use(config => {
  // log a message before any HTTP request is sent
  console.log('Request was sent');

  return config;
});

// sent a GET request
axios.get('https://api.github.com/users/sideshowbarker')
  .then(response => {
    console.log(response.data);
  });
```

در این کد، از متد axios.interceptors.request.use() برای تعریف کدی که قبل ارسال درخواست HTTP اجرا می‌شود استفاده شده است.

به طور پیش‌فرض در Fetch هیچ راهی برای توقف درخواست‌ها فراهم نشده، اما برای رفع این مشکل روشی وجود دارد. در این روش می‌توانید با بازنویسی Fetch، رهگیر خود را به صورت زیر تعریف کنید.

```
fetch = (originalFetch => {
  return (...arguments) => {
    const result = originalFetch.apply(this, arguments);
      return result.then(console.log('Request was sent'));
  };
})(fetch);

fetch('https://api.github.com/orgs/axios')
  .then(response => response.json())
  .then(data => {
    console.log(data)
  });
```

پیشرفت دانلود

شاخص‌های پیشرفت، هنگام بارگیری درخواست‌هایی با حجم بزرگ بسیار مفید هستند، به خصوص برای کاربرانی که سرعت اینترنتشان کند است. قبلاً، برنامه نویسان جاوااسکریپت از XMLHttpRequest.onprogress برای اجرای شاخص‌های پیشرفت استفاده می‌کردند. Fetch کنترل کننده onprogress ندارد. در عوض با استفاده از خاصیت body، شی response نمونه‌ای از ReadableStream را ارائه می‌دهد. مثال زیر استفاده از ReadableStream را برای گرفتن بازخورد سریع هنگام دانلود تصویر توسط کاربران رانشان می‌دهد:

```
<div id="progress" src="">progress</div>
<img id="img">

<script>

'use strict'

const element = document.getElementById('progress');

fetch('https://fetch-progress.anthum.com/30kbps/images/sunrise-baseline.jpg')
  .then(response => {

    if (!response.ok) {
      throw Error(response.status+' '+response.statusText)
    }

    // ensure ReadableStream is supported
    if (!response.body) {
      throw Error('ReadableStream not yet supported in this browser.')
    }

    // store the size of the entity-body, in bytes
    const contentLength = response.headers.get('content-length');

    // ensure contentLength is available
    if (!contentLength) {
      throw Error('Content-Length response header unavailable');
    }

    // parse the integer into a base-10 number
    const total = parseInt(contentLength, 10);

    let loaded = 0;

    return new Response(

      // create and return a readable stream
      new ReadableStream({
        start(controller) {
          const reader = response.body.getReader();

          read();
          function read() {
            reader.read().then(({done, value}) => {
              if (done) {
                controller.close();
                return;
              }
```

```
            loaded += value.byteLength;
            progress({loaded, total})
            controller.enqueue(value);
            read();
          }).catch(error => {
            console.error(error);
            controller.error(error)
          })
        }
      }
    })
  );
})
.then(response =>
  // construct a blob from the data
  response.blob()
)
.then(data => {
  // insert the downloaded image into the page
  document.getElementById('img').src = URL.createObjectURL(data);
})
.catch(error => {
  console.error(error);
})

function progress({loaded, total}) {
  element.innerHTML = Math.round(loaded/total*100)+'%';
}

</script>
```

اجرای یک شاخص پیشرفت در Axios ساده‌تر است، به خصوص اگر از ماژول Axios Progress Bar استفاده کنید. اول، باید اسکریپت‌های زیر را به پروژه اضافه کنید.

```
<link rel="stylesheet" type="text/css" href="https://cdn.rawgit.com/rikmms/progress-bar-4-axios/0a3acf92/dist/nprogress.css" />

<script src="https://cdn.rawgit.com/rikmms/progress-bar-4-axios/0a3acf92/dist/index.js"></script>
```

سپس می‌توانید نوار پیشرفت را به این شکل پیاده سازی کنید:

```
<img id="img">

<script>

loadProgressBar();

const url = 'https://fetch-progress.anthum.com/30kbps/images/sunrise-baseline.jpg';

function downloadFile(url) {
  axios.get(url, {responseType: 'blob'})
    .then(response => {
      const reader = new window.FileReader();
      reader.readAsDataURL(response.data);
      reader.onload = () => {
        document.getElementById('img').setAttribute('src', reader.result);
      }
    })
    .catch(error => {
      console.log(error)
    });
}

downloadFile(url);

</script>
```

این کد از FileReader برای خواندن همزمان تصویر دانلود شده استفاده می‌کند. تابع readAsDataURL، داده‌های تصویر را به صورت یک رشته کدگذاری شده از نوع base۶۴ باز می‌گرداند، که در ادامه برای نشان دادن تصویر در صفت src در تگ img درج می‌شود.

درخواست‌های همزمان

برای ایجاد چندین درخواست همزمان، Axios روش Axios.all() را ارائه می‌دهد. کافیست درخواست‌ها را در قالب یک آرایه فراهم آورده، سپس از axios.spread() برای اختصاص ویژگی‌های آرایه پاسخ به متغیرهای جداگانه استفاده کنید.

```
axios.all([
  axios.get('https://api.github.com/users/iliakan'),
  axios.get('https://api.github.com/users/taylorotwell')
])
.then(axios.spread((obj1, obj2) => {
  // Both requests are now complete
  console.log(obj1.data.login + ' has ' + obj1.data.public_repos + ' public repos on GitHub');
  console.log(obj2.data.login + ' has ' + obj2.data.public_repos + ' public repos on GitHub');
}));
```

شما می‌توانید با استفاده از متد built-in به همان نتیجه برسید. هنگام استفاده از Fetch همه درخواست‌ها را به عنوان یک آرایه به Promise.all() ارسال کنید. سپس، با استفاده تابع async پاسخ را مدیریت کنید. مانند کد زیر:

```
Promise.all([
  fetch('https://api.github.com/users/iliakan'),
  fetch('https://api.github.com/users/taylorotwell')
])
.then(async([res1, res2]) => {
  const a = await res1.json();
  const b = await res2.json();
  console.log(a.login + ' has ' + a.public_repos + ' public repos on GitHub');
  console.log(b.login + ' has ' + b.public_repos + ' public repos on GitHub');
})
.catch(error => {
  console.log(error);
});
```

نتیجه‌گیری

استفاده از Axios برای اکثر نیازهای ارتباطی HTTP، بسیار ساده‌تر است. با این حال، اگر شما ترجیح می‌دهید که با APIهای محلی کار کنید، هیچ چیز شما را از استفاده از ویژگی‌های Axios منع نمی‌کند.

همانطور که در این مقاله مورد بحث قرار گرفت، امکان باز تولید ویژگی‌های کلیدی کتابخانه Axios با استفاده از متد Fetch توسط مرورگرهای وب وجود دارد.

برای درک بهتر کار با Axios و Fetch، بررسی دوره جامع و پیشرفته جاوااسکریپت را پیشنهاد می‌کنیم.

sjln گفت:

آبان ۸, ۱۳۹۹ در ۲:۵۰ ب.ظ

ممنون از توضیحاتتون.

و اما سوال بنده این هست که برتری fetch نسبت به axios چی هست که عده ای از اون استفاده میکنن؟ اگر axios ساده تر هست، چرا به سمت fetch میرن؟

پاسخ

هادی گفت:

تیر ۱, ۱۳۹۹ در ۱۰:۰۱ ق.ظ

درود

من مشکلی با axios دارم

زمانی که json های فارسی را لود میکنم از یک حدودی بیشتر که میشه خطای Access-Control-Allow-Origin می دهد در حالی که وقتی داده های فارسی کمتر هست یا انگلیسی هستند مشکلی نداره

آیا راهی برای این مورد هست؟

پاسخ

مسعود صدری گفت:

تیر ۱, ۱۳۹۹ در ۱۱:۴۲ ق.ظ

سلام

این خطا مربوط به تنظیم Headerها در بک‌اند هست.

لطفا کدتون و خطا رو توی تلگرام یا واتس‌اپ بفرستین تا بررسی کنم.

همین طور تنظیمات Header در بک‌اند.

پاسخ

MinA گفت:

اردیبهشت ۱۴, ۱۳۹۹ در ۱۰:۴۹ ق.ظ

سلام خیلیی مفید بود. مرسی از روژین برای مقاله های خوبش .

پاسخ

مسعود صدری گفت:

اردیبهشت ۱۴, ۱۳۹۹ در ۱۱:۲۳ ق.ظ

سلام

خیلی ممنون که مطالعه کردین.

پاسخ

امید گفت:

اردیبهشت ۱۱, ۱۳۹۹ در ۸:۳۰ ق.ظ

سلام مقاله واقعا خوبی بود و ممنون

فقط یه مشکلی وجود داره قسمت ۹۳ دوره MERN هنوز نیومده که بریم ببینیم طریقه کار با AbortController بریم ببینیم چه جوریه 😀

پاسخ

مسعود صدری گفت:

اردیبهشت ۱۱, ۱۳۹۹ در ۸:۳۷ ق.ظ

سلام

ممنون از شما.

لطفا دوباره بررسی کنید، تقریبا دو روز پیش به روز رسانی انجام شده.

برای اطلاع از به روز رسانی دوره‌ها لطفا کانال تلگرام رو دنبال کنید.

پاسخ

با عنوان leilan وارد شده‌اید. خارج می‌شوید؟

دیدگاه

فرستادن دیدگاه