React - AXIOS

Установка
Экземпляр AXIOS
Конфигурация запроса
Схема ответа
Конфигурация по умолчанию
Перехват запросов
Обработка ошибок
Отмена запросов
URL кодирующие параметры
JWT токен (авторизация)

Что такое AXIOS

Axios - это HTTP-клиент, основанный на Promise для node.js и браузера. Он изоморфный (он может работать в браузере и node.js с той же базой кодов). На стороне сервера он использует нативный node.js http-модуль, тогда как на стороне клиента (браузер) он использует XMLHttpRequests.

Команда установки

npm install axios

импорт axios в js

import axios from “axios”

Экземпляр AXIOS

Вы можете создать новый экземпляр axios с пользовательской конфигурацией.

axios.create([config])

```
const instance = axios.create({
  baseURL: 'https://some-domain.com/api/',
  timeout: 1000,
  headers: {'X-Custom-Header': 'foobar'}
});
```

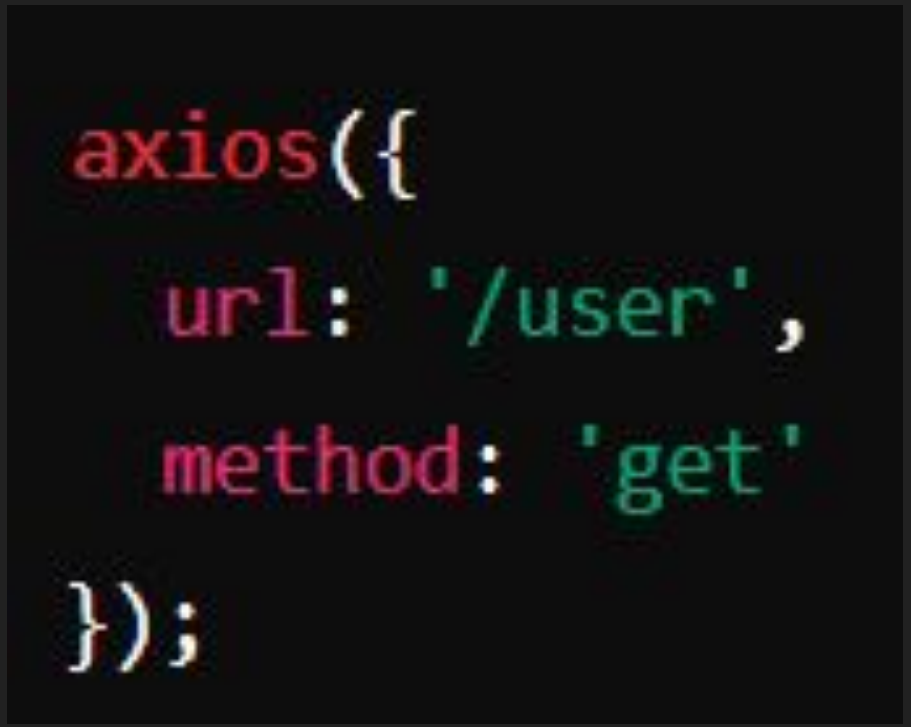
Конфигурация запросов

Конфигурация запроса в Axios — это набор параметров, который определяет, как запрос будет отправлен, какие данные будут переданы, как они будут обработаны и какие параметры взаимодействия с сервером применяются. Она позволяет детально управлять поведением запросов, обрабатывать ошибки и добавлять заголовки, параметры, тайм-ауты и многое другое.

url

Описание: URL-адрес сервера для запроса.

Здесь отправляется GET-запрос на /user.



method

Описание: Указывает метод HTTP-запроса. По умолчанию — GET.

Это отправит POST-запрос на /user.

baseURL

Описание: Добавляется к относительным URL-адресам.

```
const instance = axios.create({
  baseURL: 'https://api.example.com'
});

instance.get('/users'); // Запрос на https://api.example.com/users
```

transformRequest

Описание: Позволяет изменять данные запроса перед отправкой.

```
axios({
  url: '/user',
  method: 'post',
  data: { name: 'Alex' },
  transformRequest: [(data, headers) => {
    data.name = data.name.toUpperCase();
    return JSON.stringify(data);
  }],
  headers: { 'Content-Type': 'application/json' }
});
```

transformResponse

Описание: Изменяет ответ перед передачей его в then/catch.

```
axios({
  url: '/user',
  method: 'get',
  transformResponse: [function (data) {
    return JSON.parse(data).map(user => user.name.toUpperCase());
  }]
}).then(response => {
  console.log(response.data); // Преобразованные данные
});
```

headers

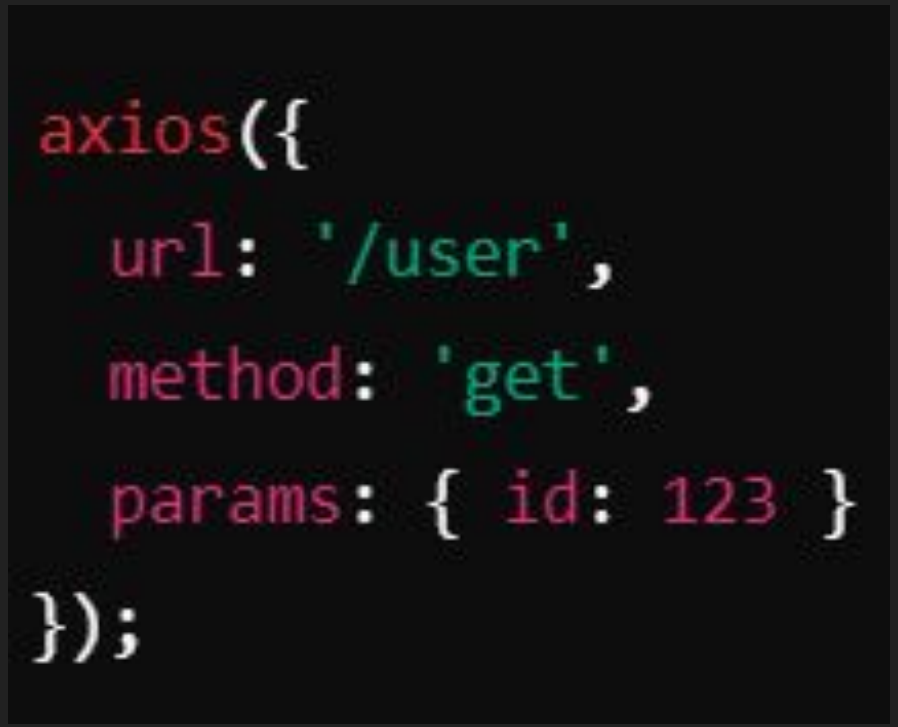
Описание: Устанавливает пользовательские заголовки.

Добавляет заголовок авторизации.

```
axios({
  url: '/user',
  method: 'get',
  headers: { 'Authorization': 'Bearer token123' }
});
```

params

Описание: URL-параметры для запроса.



paramsSerializer

Описание: Позволяет настраивать сериализацию параметров.

```
axios({
  url: '/user',
  method: 'get',
  params: { id: [1, 2, 3] },
  paramsSerializer: function(params) {
    return Qs.stringify(params, { arrayFormat: 'brackets' });
  }
});
```

data

Описание: Данные, отправляемые в теле запроса. Применимо для POST, PUT, DELETE, PATCH.

```
axios({
  url: '/user',
  method: 'post',
  data: { name: 'Alex', age: 30 }
});
```

timeout

Описание: Максимальное время ожидания запроса.

Запрос будет отменен, если сервер не ответит за 5 секунд.

```
axios({
  url: '/user',
  method: 'get',
  timeout: 5000 // 5 секунд
});
```

withCredentials

Описание: Указывает, должны ли отправляться куки и другие учетные данные при кросс-доменных запросах.

Кросс-доменный запрос с отправкой учетных данных.

```
axios({
  url: 'https://example.com/user',
  method: 'get',
  withCredentials: true
});
```

auth

Описание: Для базовой HTTP-аутентификации.

Отправляет запрос с HTTP-аутентификацией.

```
axios({
  url: '/user',
  method: 'get',
  auth: {
    username: 'user',
    password: 'password'
  }
});
```

responseType

Описание: Определяет тип данных ответа (json, blob, text, arraybuffer).

Запрос для получения бинарных данных.

```
axios({
  url: '/file',
  method: 'get',
  responseType: 'blob'
});
```

onUploadProgress

Описание: Обработчик событий загрузки (для браузера).

```
axios({
  url: '/upload',
  method: 'post',
  data: formData,
  onUploadProgress: function(progressEvent) {
    console.log('Загружено: ' + Math.round((progressEvent.loaded * 100) / progressEvent.total));
  }
});
```

cancelToken

Описание: Используется для отмены запроса.

```
const CancelToken = axios.CancelToken;
const source = CancelToken.source();

axios({
  url: '/user',
  cancelToken: source.token
});

// Отмена запроса
source.cancel('Запрос отменен пользователем');
```

Схема ответа

```
{
  // `data`- это ответ, предоставленный сервером
  data: {},

  // `status`- это код состояния HTTP-запроса
  status: 200,

  // `statusText`- это сообщение о состоянии HTTP-запроса
  statusText: 'OK',

  // `headers` заголовки HTTP-запроса, на которые ответил сервер
  // Все имена заголовков написаны в нижнем регистре, и к ним можно получить доступ, используя кв
  // Например: `response.headers['content-type']`
  headers: {},

  // `config` - это конфигурация, которая была предоставлена `axios` для запроса
  config: {},

  // `request` - это запрос, который сгенерировал этот ответ
  // Это последний экземпляр ClientRequest в node.js (в перенаправлениях)
  // и экземпляр XMLHttpRequest в браузере.
  request: {}
}
```

```
axios.get('/user/12345')
  .then(function (response) {
    console.log(response.data);
    console.log(response.status);
    console.log(response.statusText);
    console.log(response.headers);
    console.log(response.config);
  });
```

Конфигурация по умолчанию

Глобальные значения по умолчанию для axios через defaults у axios

Пользовательский экземпляр axios через defaults у экземпляра

Конфигурация будет объединена в порядке приоритета. Порядок: значения по умолчанию для библиотеки, найденные в lib/defaults.js, затем свойства defaults для экземпляра и, наконец, аргумент config для запроса. Последнее будет иметь приоритет над первым.

```
axios.defaults.baseURL = 'https://api.example.com';
axios.defaults.headers.common['Authorization'] = AUTH_TOKEN;
axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';
```

```
// Установка настройки по умолчанию при создании экземпляра
const instance = axios.create({
  baseURL: 'https://api.example.com'
});

// Изменение значения по умолчанию после создания экземпляра
instance.defaults.headers.common['Authorization'] = AUTH_TOKEN;
```

```
// Создайте экземпляр, используя настройки по умолчанию, предоставленные библиотекой.
// На данный момент значение времени ожидания равно «0», что является значением по умолчанию для библиотеки.
const instance = axios.create();

// Переопределите время ожидания по умолчанию для библиотеки
// Теперь все запросы, использующие этот экземпляр, будут ждать 2,5 секунды до истечения времени ожидания.
instance.defaults.timeout = 2500;

// Переопределение времени ожидания для этого запроса, поскольку известно, что он занимает много времени
instance.get('/longRequest', {
  timeout: 5000
});
```

Перехват запросов

Вы можете перехватывать запросы или ответы до того, как они будут then или catch.

```
// Добавляем перехват запросов
axios.interceptors.request.use(function (config) {
    // Здесь можете сделать что-нибудь с перед отправкой запроса
    return config;
  }, function (error) {
    // Сделайте что-нибудь с ошибкой запроса
    return Promise.reject(error);
  });

// Добавляем перехват ответов
axios.interceptors.response.use(function (response) {
    // Любой код состояния, находящийся в диапазоне 2xx, вызывает срабатывание этой функции
    // Здесь можете сделать что-нибудь с ответом
    return response;
  }, function (error) {
    // Любые коды состояния, выходящие за пределы диапазона 2xx, вызывают срабатывание этой функции
    // Здесь можете сделать что-то с ошибкой ответа
    return Promise.reject(error);
  });
```

Работа с перехватчиками

Если вам нужно, вы можете удалить перехватчик

Вы можете добавить перехватчики в пользовательский экземпляр axios.

```
const myInterceptor = axios.interceptors.request.use(function () {/*...*/});
axios.interceptors.request.eject(myInterceptor);
```

```
const instance = axios.create();
instance.interceptors.request.use(function () {/*...*/});
```

Обработка ошибок

Обработка ошибки классическим отловом через catch.

Используя toJSON, вы получаете объект с дополнительной информацией об ошибке HTTP.

```
axios.get('/user/12345')
  .catch(function (error) {
    console.log(error.toJSON());
  });
```

```
axios.get('/user/12345')
  .catch(function (error) {
    if (error.response) {
      // Запрос был сделан, и сервер ответил кодом состояния, который
      // выходит за пределы 2xx
      console.log(error.response.data);
      console.log(error.response.status);
      console.log(error.response.headers);
    } else if (error.request) {
      // Запрос был сделан, но ответ не получен
      // `error.request`- это экземпляр XMLHttpRequest в браузере и экземпляр
      // http.ClientRequest в node.js
      console.log(error.request);
    } else {
      // Произошло что-то при настройке запроса, вызвавшее ошибку
      console.log('Error', error.message);
    }
    console.log(error.config);
  });
```

Обработка ошибок

Используя параметр конфигурации validateStatus, вы можете определить HTTP-коды, которые должны вызывать ошибку.

```
axios.get('/user/12345', {
  validateStatus: function (status) {
    return status < 500; // Разрешить, если код состояния меньше 500
  }
})
```

Отмена запросов

Отмененный запрос в Axios — это процесс остановки выполнения HTTP-запроса, если он больше не нужен или если происходят условия, при которых его результат будет неактуален. Это полезно в ситуациях, когда нужно предотвратить выполнение запросов, которые могут занимать ресурсы, но результат не будет использован.

Когда применять отмену запросов:

- Когда пользователь меняет контекст (например, переходит между страницами).
- Для предотвращения перегрузки сервера при множественных запросах.
- Для улучшения производительности и UX, когда результат предыдущего запроса больше не нужен.

AbortController

Начиная с версии v0.22.0 Axios поддерживает AbortController для отмены запросов через fetch API:

```
const controller = new AbortController();

axios.get('/foo/bar', {
   signal: controller.signal
}).then(function(response) {
   //...
});
// отмена запроса
controller.abort()
```

CancelToken устарел

Вы также можете отменить запрос, используя CancelToken.

API токена отмены axios основан на отозванном cancelable promises proposal.

```
const CancelToken = axios.CancelToken;
const source = CancelToken.source();

axios.get('/user/12345', {
  cancelToken: source.token
}).catch(function (thrown) {
  if (axios.isCancel(thrown)) {
    console.log('Request canceled', thrown.message);
  } else {
    // обработка ошибки
  }
});

axios.post('/user/12345', {
  name: 'new name'
}, {
  cancelToken: source.token
})

// отмена запроса (указывать сообщение необязательно)
source.cancel('Operation canceled by the user.');
```

URL кодирующие параметры

По умолчанию axios сериализует объекты JavaScript в JSON. Чтобы отправить данные в формате application/x-www-form-urlencoded, вы можете использовать один из следующих вариантов.

Кроме того, вы можете кодировать данные с помощью библиотеки qs:

```
const params = new URLSearchParams();
params.append('param1', 'value1');
params.append('param2', 'value2');
axios.post('/foo', params);
```

```
import qs from 'qs';
const data = { 'bar': 123 };
const options = {
  method: 'POST',
  headers: { 'content-type': 'application/x-www-form-urlencoded' },
  data: qs.stringify(data),
  url,
};
axios(options);
```

JWT токен (авторизация)

JWT (JSON Web Token) — это компактный, URL-безопасный стандарт для передачи данных между сторонами в виде JSON-объекта. Каждый токен состоит из трех частей, разделенных точками: Header, Payload и Signature. Эти части кодируются в формате Base64Url.

Структура JWT:

- **Header (заголовок)** — содержит тип токена (в данном случае "JWT") и алгоритм шифрования, например, HS256 (HMAC + SHA256).
- **Payload (полезная нагрузка)** — содержит данные, которые нужно передать, например, идентификатор пользователя (ID), роли, время действия токена и прочее.
- **Signature (подпись)** — генерируется с использованием секретного ключа. Подпись защищает токен от изменения данных.

Применение JWT на Frontend:

Аутентификация: JWT часто используется для подтверждения подлинности пользователя. После успешного входа в систему сервер генерирует JWT и отправляет его клиенту. Клиент сохраняет этот токен, например, в localStorage или sessionStorage.

Отправка JWT с запросами: Для доступа к защищенным ресурсам клиент добавляет JWT в заголовок HTTP-запросов, обычно в поле Authorization:

Authorization: Bearer <JWT-токен>

Таким образом, каждый запрос на сервер аутентифицируется без необходимости повторной авторизации.

Валидация токена: На сервере проверяется подпись JWT, чтобы удостовериться, что токен не был подделан. Также проверяются такие параметры, как срок действия токена.

Обновление токена: Для увеличения безопасности можно реализовать механизм "refresh token", при котором токен обновляется после его истечения.

Пример использования JWT на Frontend:

JWT позволяет безопасно передавать данные между Frontend и Backend, обеспечивает простую аутентификацию и масштабируемость приложения.

```
import axios from 'axios';

axios.post('https://api.example.com/login', {
  username: 'yourUsername',
  password: 'yourPassword'
})
  .then(response => {
    const token = response.data.token; // Получаем токен от сервера
    localStorage.setItem('token', token); // Сохраняем токен в localStorage
  })
  .catch(error => {
    console.error('Ошибка авторизации:', error);
  });
```

```
const token = localStorage.getItem('token'); // Достаём токен из localStorage

axios.get('https://api.example.com/protected', {
  headers: {
    Authorization: `Bearer ${token}`
  }
})
  .then(response => {
    console.log('Защищённые данные:', response.data);
  })
  .catch(error => {
    console.error('Ошибка доступа:', error);
  });
```

Ресурсы

Код с урока (git репозиторий) - ТЫК

Документация AXIOS - ТЫК

JWT io - ТЫК