

Web講習会2021 ワールドワイドウェブ発展

第4回: REST API



Arthur

この回の目標

- 簡単なREST APIを設計できるようになる
- HTTPの幂等性と安全性を理解し、APIの設計において適切なメソッドを選ぶようになる
- ドキュメントを読んで、REST APIを利用できるようになる

Web API

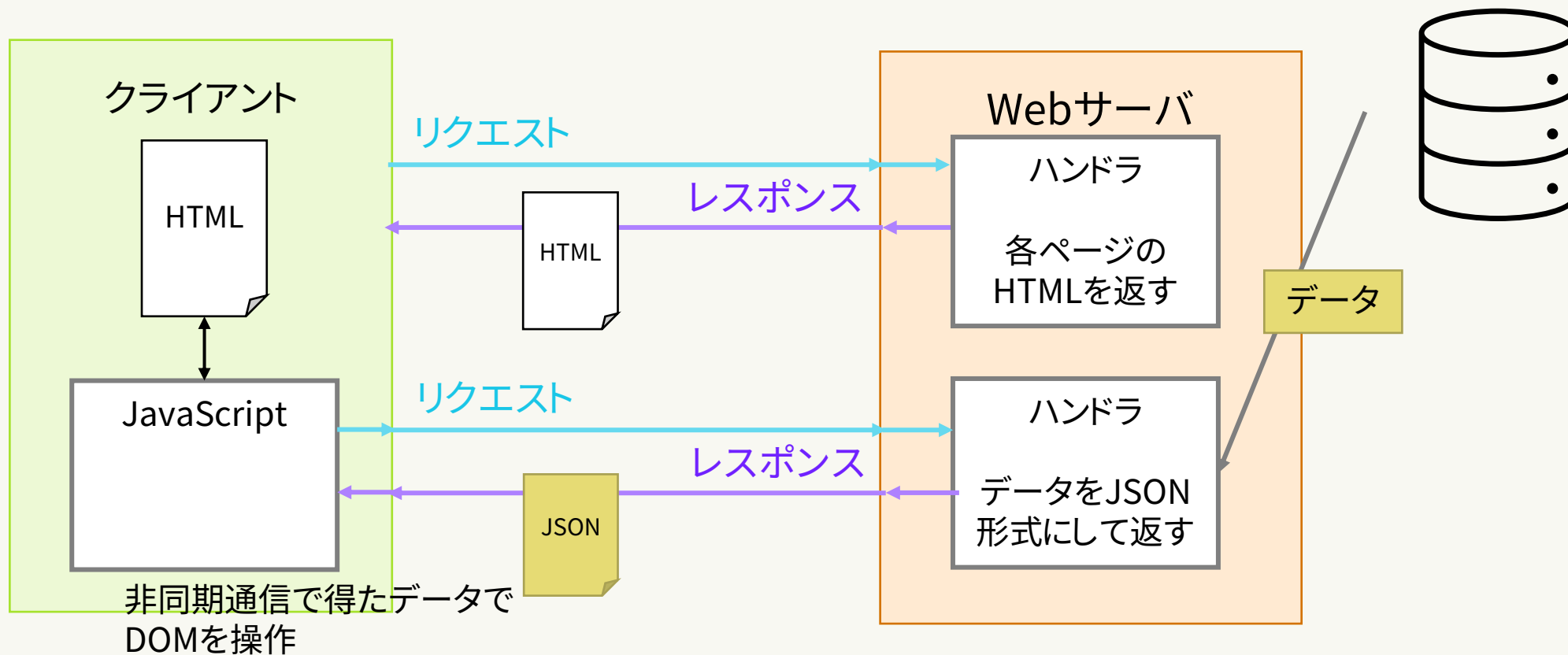


[再掲] AJAX

非同期通信: ブラウザを通常通り動作させたまま裏で通信

ページ遷移なく表示する情報を更新できる

データベース

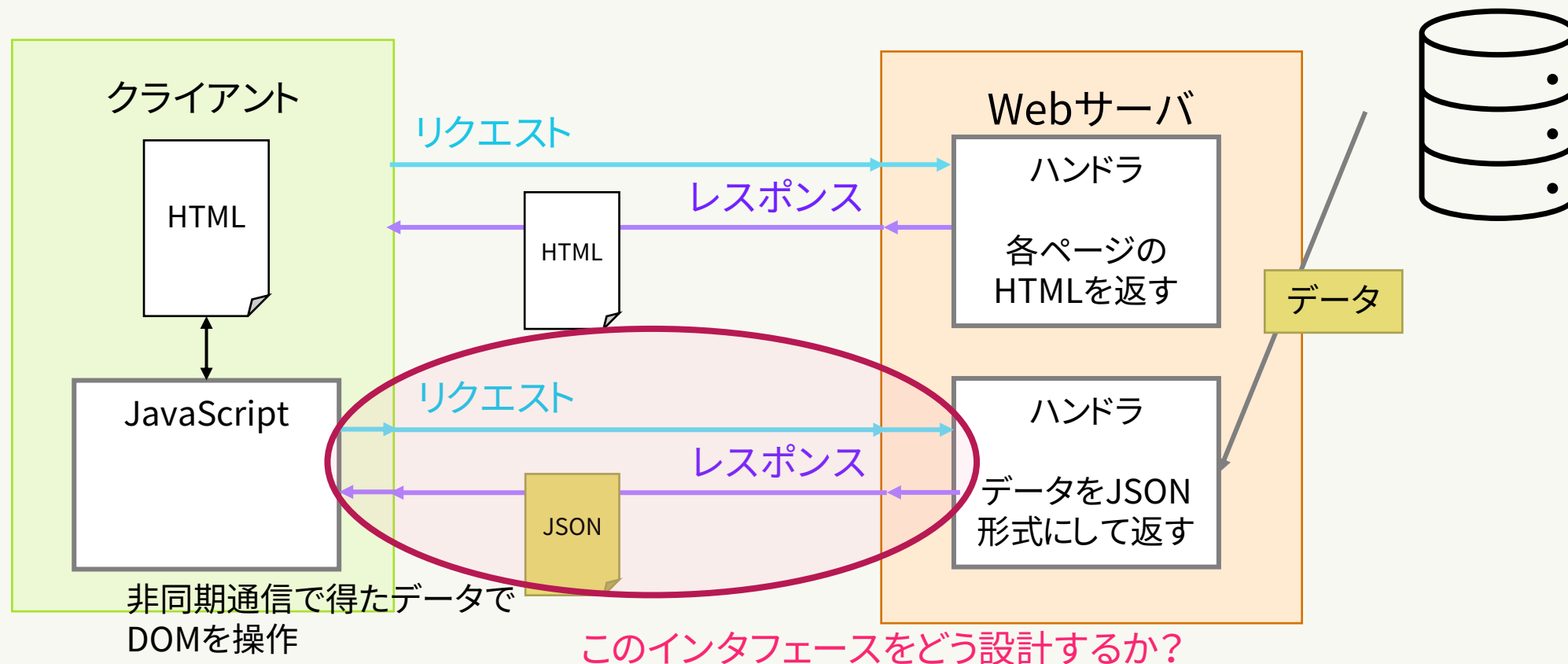


[再掲] AJAX

非同期通信: ブラウザを通常通り動作させたまま裏で通信

ページ遷移なく表示する情報を更新できる

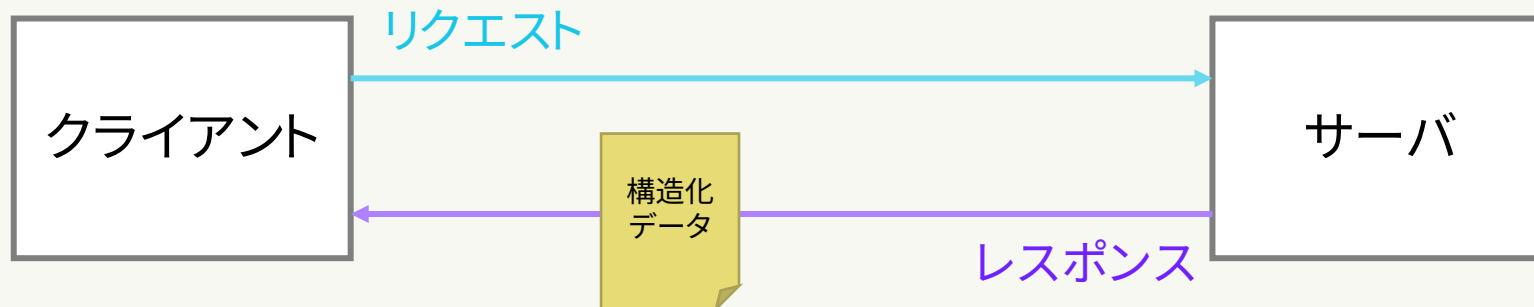
データベース



Web API

本スライドでの (Web) APIの定義

HTTPプロトコルによってやりとりする通信のインタフェース
および、それにより呼び出される側のシステム



Web APIの例

様々なWebサービスがAPIを公開している

- Twitter API
- LINE API
- Graph API (Facebook)
- 住所検索API
- 鉄道駅検索API
- ...

一般には公開されていないが、CSR(⇒第1回)やスマホアプリのためのAPIが用意されていることも

REST API



REST API

REST REpresentational State Transfer

Web APIの設計思想の1つ

RESTfulなAPI … REST API

リソースを、明確で、言語に依存せず標準化されたクライアント＝サーバ間の相互作用により転送する

RESTの4原則

近年のRESTなWebサービスの基本原則

1. 明示的にHTTPメソッドを使う
2. ディレクトリ構造に似たURIで公開する
3. ステートレスにする
4. XML、JSONのいずれか、またはその両方でデータを表現する

RESTの4原則

近年のRESTなWebサービスの基本原則

1. 明示的にHTTPメソッドを使う
2. ディレクトリ構造に似たURIで公開する
3. ステートレスにする
4. XML、JSONのいずれか、またはその両方でデータを表現する

リクエストの内容

1行目…リクエストライン

メソッド名、パス、バージョン指定

2行目以降…ヘッダ

[ヘッダ名]: [値]の形式で1行ずつ

Hostヘッダは(HTTP1.1では)必須
空行でヘッダの終わり

空行以降…メッセージボディ

リソースを取得したいだけ(GET)
なら空でOK

```
GET / HTTP/1.1
Host: buratsuki.page
Accept-Language: ja
```

HTTPリクエストメソッド


表1 HTTP リクエストメソッドの一覧

| メソッド名 | 意味 | 使用状況 |
|---------|-----------------------------------|---|
| GET | 指定したリソースの表現をリクエスト | データの取り込み |
| HEAD | GET と同じだがレスポンス本文はない | 帯域幅を節約するために大きなリソースをダウンロードするかどうかを、事前に決定したいとき |
| POST | サーバーにデータを送信 | フォームの入力内容を送信 |
| PUT | 対象リソースの現在の表現の全体を、リクエストのペイロードで置き換え | 幂等である (複数回成功しても副作用がない) ことが求められるとき |
| DELETE | 指定したリソースを削除 | |
| CONNECT | 対象リソースで識別されるサーバーとの間にトンネルを確立 | |
| OPTIONS | 対象リソースの通信オプションを示す | |
| TRACE | 対象リソースへのパスに沿ってメッセージのループバックテストを実行 | |
| PATCH | リソースを部分的に変更 | |

REST APIとHTTPメソッド

REST APIでは、データについて行いたい操作に対応するHTTPメソッドを選んで使う

CRUD: 永続性(データベースetc)の4つの機能

| CRUD | | HTTPメソッド |
|--------|--|----------|
| Create | | POST |
| Read |  | GET |
| Update | | PUT |
| Delete | | DELETE |

RESTなURI設計

REST APIでは着目するリソースとURIを対応付ける

[例]

URIは基本リソースの名称(複数形)だけ
NG例: /get_books

| 操作 | URI |
|-----------|-----------------|
| 本の一覧を取得 | GET /books |
| id=1の本を取得 | GET /books/1 |
| 新しい本を追加 | POST /books |
| id=2の本を編集 | PUT /books/2 |
| id=5の本を削除 | DELETE /books/5 |

ディレクトリ構造に似たURI

リソースが階層関係にあるなら、URIでディレクトリツリーを表現

[例] 1つの本に複数の貸し出し履歴が紐づいている

| 操作 | URI |
|-------------------------|----------------------------|
| id=1の本の貸し出し履歴を取得 | GET /books/1/histories |
| id=1の本のid=520の貸し出し履歴を取得 | GET /books/1/histories/520 |

RESTの4原則

近年のRESTなWebサービスの基本原則

1. 明示的にHTTPメソッドを使う
2. ディレクトリ構造に似たURIで公開する
3. ステートレスにする
4. XML、JSONのいずれか、またはその両方でデータを表現する

HTTPのステートレス性

純粋なHTTPはステートレス

同じリクエストで、リソースの状態が変わっていなければ必ず同じレスポンス

対話形式のフォームの通信のやりとりのイメージ

客： 「ハンバーガーセットをお願いします」

店員： 「サイドメニューは何になさいますか？」

客： 「ハンバーガーセットをポテトをお願いします」

店員： 「ドリンクは何になさいますか？」

客： 「ハンバーガーセットをポテトとジンジャーエールをお願いします」

⋮

実際のWebはステートレス？

ステートレスだとログインが必要なページなどで面倒

セッションを用いてサーバに状態を保持する

セッションIDをCookieとしてクライアントに保存させ、ヘッダに乗せて送信

REST APIでは、このように状態に依存する実装を許容しない

レスポンスを定めるのに必要な情報は全てリクエストに乗せる

ex) 認証情報など

Web APIの利用



サンプルAPIの起動

Web APIのサンプルを用意しました
code4.zipを解凍した中のserverディレクトリを開く

以下の手順で実行

1. `yarn` で依存関係をインストール
2. `yarn compile` でTSをコンパイル
3. `yarn start` でサーバ起動

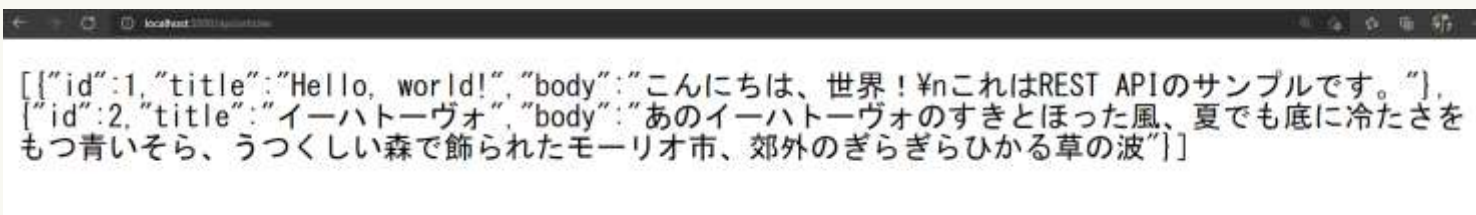
```
~/repositories/tutorial-web-2021-advanced/code4/server >>> yarn
yarn install v1.22.10
warning package.json: No license field
warning No license field
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

Done in 0.46s.
~/repositories/tutorial-web-2021-advanced/code4/server >>> yarn compile
yarn run v1.22.10
warning package.json: No license field
$ tsc
Done in 1.35s.
~/repositories/tutorial-web-2021-advanced/code4/server >>> yarn start
yarn run v1.22.10
warning package.json: No license field
$ node dist/index.js
SAMPLE REST API is listening on port 3000.
```

サンプルAPIの起動確認

サーバが起動した状態のまま

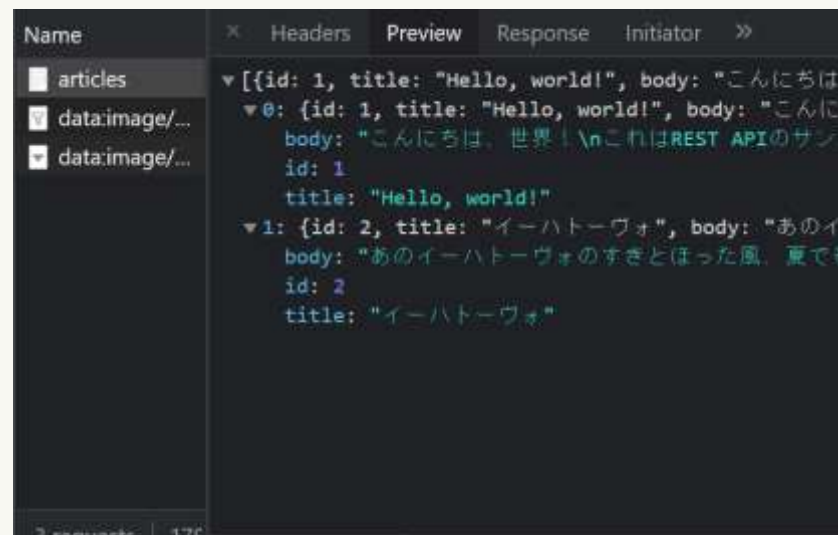
`http://localhost:3000/api/articles`
にアクセス



```
[{"id":1,"title":"Hello, world!","body":"こんにちは、世界！\nこれはREST APIのサンプルです。"}, {"id":2,"title":"イーハトーヴォ","body":"あのイーハトーヴォのすきとほった風、夏でも底に冷たさをもつ青いそら、うつくしい森で飾られたモーリオ市、郊外のぎらぎらひかる草の波"}]
```

開発者ツールのNetworkタブを利用すると
整形して表示できる

もしくはブラウザ拡張機能 etc



サンプルAPIの仕様書①

GET /api/articles ... 記事の一覧を取得

レスポンス

コード 200: 成功

```
[
  {
    id: 1,
    title: 'Hello, world!',
    body: 'こんにちは、世界！\nこれはREST APIのサンプルです。'
  },
  {
    id: 2,
    title: 'イーハトーヴォ',
    body: 'あのイーハトーヴォのすきとほった風、夏でも底に冷たさをもつ青いそら、うつくしい森で飾られたモーリオ市、郊外のぎらぎらひかる草の波'
  }
]
```

サンプルAPIの仕様書②

GET /api/articles/{id} … 指定したidの記事を取得

リクエストパラメータ

id (path): 記事のID[必須]

レスポンス

コード 200: 成功

```
{  
  id: 1,  
  title: 'Hello, world!',  
  body: 'こんにちは、世界！\nこれはREST APIのサンプルです。'  
}
```

コード 404: 指定したidの記事が存在しない

サンプルAPIの仕様書③

POST /api/articles ... 記事を新規作成

リクエストパラメータ

title (body): 記事のタイトル[必須]

body (body): 記事の本文[必須]

レスポンス

コード 200: 成功

```
{ id: 3, title: 'テスト投稿です', body: 'こんにちは!' }
```

サンプルAPIの仕様書④

PUT /api/articles/{id} … 指定したidの記事を編集

リクエストパラメータ

id (path): 記事のID[必須]

title (body): 記事のタイトル[必須]

body (body): 記事の本文[必須]

レスポンス

コード 200: 成功

```
{ id: 1, title: 'Hello, world!', body: '編集ができていることを確認' }
```

コード 404: 指定したidの記事が存在しない

サンプルAPIの仕様書⑤

DELETE /api/articles/{id} … 指定したidの記事を削除

リクエストパラメータ

id (path): 記事のID[必須]

レスポンス

コード 200: 成功

```
{  
  id: 2,  
  title: 'イーハトーヴォ',  
  body: 'あのイーハトーヴォのすきとほった風、夏でも底に冷たさをもつ青いそら、うつくしい森で飾られたモーリオ市、郊外のきらきらひかる草の波'  
}
```

コード 404: 指定したidの記事が存在しない

リクエストbodyのContent-Type

POSTやPUTなどのメソッドでは、リクエストのbodyにデータを乗せることがある

bodyデータの表現形式: Content-Type

- `application/x-www-form-urlencoded`
キーと値を = でつないだものを & で連結して表現
ex) `title=test&body=HelloWorld!`
- `application/json` ⇐ 今回はこちら
JSON形式で表現
ex) `{"title": "test", "body": "HelloWorld!"}`

APIへリクエストを送るプログラム

APIにリクエストを送り、結果を取得するプログラムを作る
JSON形式でのHTTP通信には、**axios**というパッケージが便利

こちらのサンプルコードも用意してあります

code4.zipの中のclientディレクトリを開く
以下の手順で実行

1. **yarn** で依存関係をインストール
2. **node index.js** で実行

```
~/repositories/tutorial-web-2021-advanced/code4/client >>> yarn
yarn install v1.22.10
warning package.json: No license field
warning No license field
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
Done in 0.10s.
~/repositories/tutorial-web-2021-advanced/code4/client >>> node index.js
200
[
  {
    id: 1,
    title: 'Hello, world!',
    body: 'こんにちは、世界！\nこれはREST APIのサンプルです。'
  },
  {
    id: 2,
    title: 'イーハトーヴォ',
    body: 'あのイーハトーヴォのすきとほった風、夏でも底に冷たさをもつ青いそら、うつくしい森で飾られたモーリオ市、郊外のざらざらひかる草の波'
  }
]
```

[再掲] インストールしたパッケージの利用

npmやYarnでインストールしたパッケージは、ビルトインモジュールと同様に利用可能

```
const axios = require('axios')
const url = 'http://s3-ap-northeast-1.amazonaws.com/titechapp-web-data/news.json'
const main = async () => {
  const response = await axios.get(url)
  console.log(response.data)
}
main()
```

```
~/Documents/titechapp/pr4 >>> node axios_test.js
[
  {
    id: 1,
    type: 2,
    textJa: 'TitechApp 2(iOS)をリリースしました！',
    textEn: 'TitechApp 2 for iOS has been released!',
    linkURL: '/products/titechapp/',
    date: '2018.07.12'
  },
  {
    id: 2,
    type: 1,
    textJa: 'Titech App ProjectのWebサイトを公開しました！',
    textEn: 'The website of Titech App Project has been opened!',
    linkURL: '',
    date: '2018.07.13'
  },
  {
    id: 3,
    type: 2,
    textJa: 'TitechApp 2(Android)をリリースしました！',
    textEn: 'TitechApp 2 for Android has been released!',
    linkURL: '/products/titechapp/',
    date: '2019.01.05'
  }
]
```

GETリクエストの送信

```
const axios = require('axios')
const HOST = 'http://localhost:3000'
const main = async () => {
  const response = await axios.get(`${HOST}/api/articles`)
  console.log(response.status)
  console.log(response.data)
}
main()
```

```
~/repositories/tutorial-web-2021-advanced/code4/client >>> node index.js
200
[
  {
    id: 1,
    title: 'Hello, world!',
    body: 'こんにちは、世界！\nこれはREST APIのサンプルです。'
  },
  {
    id: 2,
    title: 'イーハトーヴォ',
    body: 'あのイーハトーヴォのすきとおった風、夏でも底に冷たさをもつ青いそら、うつくしい森で飾られたモーリオ市、郊外のぎらぎらひかる草の波'
  }
]
```

POSTリクエストの送信

リクエストbodyに何かデータを入れたいときは、第3引数を利用
オブジェクトを入れると、axiosはJSON形式にして送信してくれる

```
const main = async () => {  
  const payload = {  
    title: 'テスト投稿です',  
    body: 'こんにちは!',  
  }  
  const response = await axios.post(`${HOST}/api/articles`, payload)  
  console.log(response.status)  
  console.log(response.data)  
}  
main()
```

```
~/repositories/tutorial-web-2021-advanced/code4/client >>> node index.js  
200  
{ id: 3, title: 'テスト投稿です', body: 'こんにちは!' }
```


冪等性と安全性

HTTPメソッドの冪等性

冪等性

サーバの状態が同じなら、あるリクエストを何回受信しても同じ効果が発生

ex) GET /api/articles は状態が同じであれば結果が同じ → 冪等

ex) POST /api/articles は、2回実行すると2個記事ができる → not 冪等

以下のHTTPメソッドは冪等であるべき

- GET
- HEAD
- PUT
- DELETE

2回目以降はレスポンスコードが変わるが、
サーバの状態は変わらない

クイズ

Q.

「最後に追加された記事を削除する」

という機能を持ったエンドポイントを作るとき、
DELETEメソッドを利用して良いか？

クイズ

Q.

「最後に追加された記事を削除する」

という機能を持ったエンドポイントを作るとき、
DELETEメソッドを利用して良いか？

A.

いいえ、DELETEメソッドは幂等であるべき

2回呼ばれると2個記事が消えるので幂等ではない

HTTPメソッドの安全性

安全性

サーバの状態を変更しない

ex) GET /api/articles を実行してもサーバの記事データは不変 → 安全

ex) PUT /api/articles はサーバの記事データを書き換える → not 安全

以下のHTTPメソッドは安全であるべき

- GET
- HEAD

次回予告

Reactによるフロントエンド開発

座学とペアプログラミングの組み合わせで実施予定