Jest用 nuxt-content のモック

jest-mock-nuxt-content

背景

nuxt-content の `fetch` 実行時にチェーンシーケンスをテストしてみると意外と面倒だった。

今後も nuxt-content を利用したいと思っているので、jest-mock-axios を参考にモックを作ってみた。

どこが面倒だったか

`asyncData` 終了後に各メソッド(`sortBy` など)のモックをまとめて確認すると「実行された(チェーンに追加された)順番など」が把握しにくい。

モックの作りにもよるが「チェーンの深い位置にあるメソッド」を確認する場合、目的のモックメソッドに狙いを定めることが少し面倒だった。

```
// 例: skip(p) と limit(n) を確認したい。
$content('blog').sortBy('createdAt').limit(100).sortBy('id')
    .only(['title']).skip(p).limit(n).fetch()
```

対応

jest-mock-axios を参考にする

jest-mock-axios では「待機状態のリクエストに対してモックデータを渡す」ようになっている。

この方式の良いところは「リクエスト別にモックデータと検証処理を順次記述(実行)しやすい」ことにある(と勝手に思っている)。

jest-mock-nuxt-content では

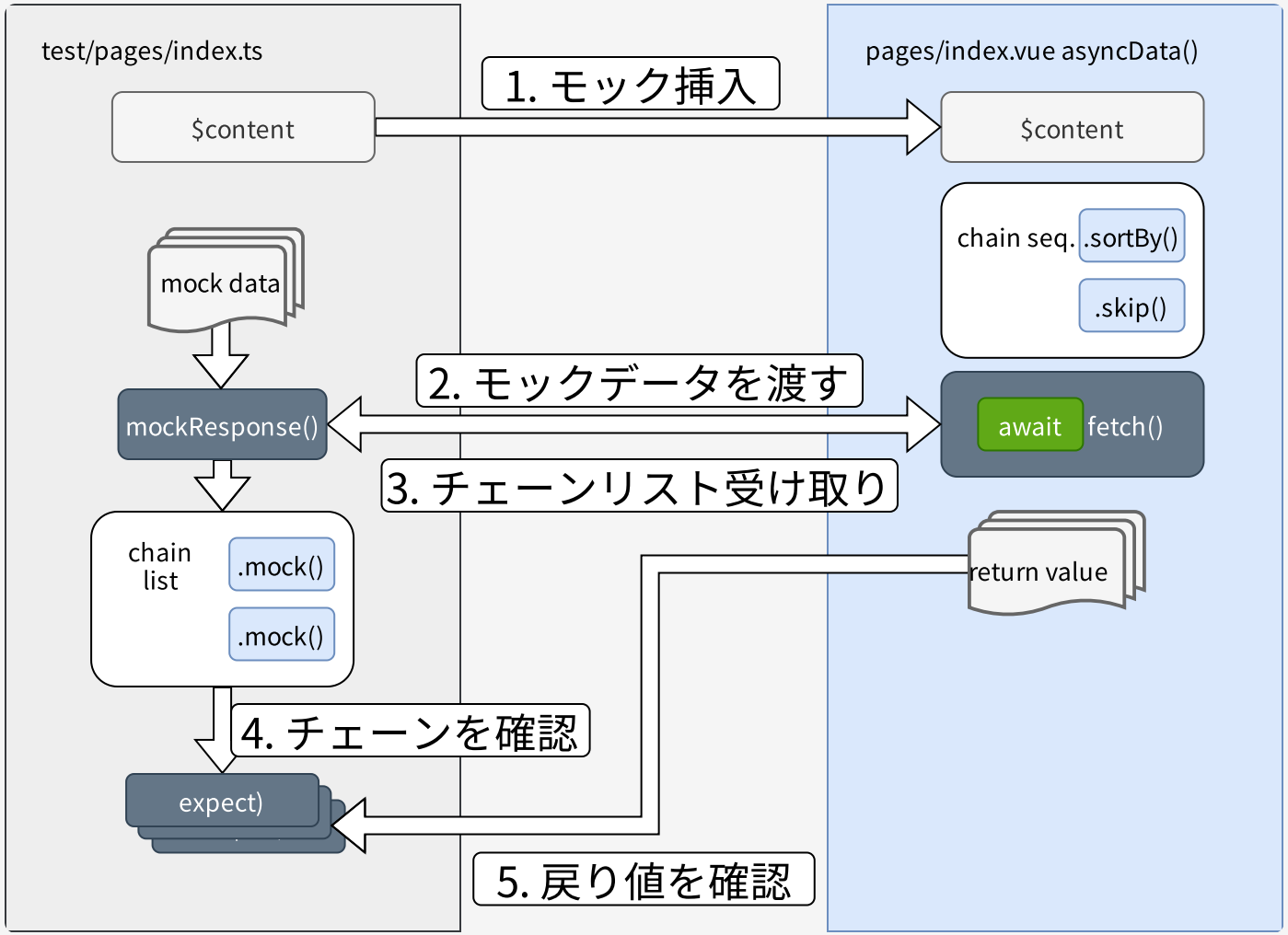
同じように待機状態の `fetch` へモックデータを渡す API を作成し、その API から「`fetch` が実行されたときの各メソッドの実行履歴」を返すようにした。

これにより「`fetch` 別にメソッド(モック)の実行履歴が隔離」され、「メソッドの実行順」や「使われた(使われなかった)メソッド」なども把握しやすくなる。

基本的な流れ

1. `asyncData` へモック(`$content`)を挿入する
2. 待機中の `fetch()` へ `mockResponse()` API でモックデータを渡す
3. このとき同時にメソッドのチェーンリスト(造語: モック化したチェーンシーケンス)を受け取る
4. チェーンリスト内のモックでメソッドの実行状況を確認
5. 最終的に戻り値を確認

概要図


test/pages/index.ts
pages/index.vue asyncData()
1. モック挿入
$content
$content
chain seq.
.sortBy()
.skip()
mock data
2. モックデータを渡す
mockResponse()
await
fetch()
3. チェーンリスト受け取り
chain
list
.mock()
.mock()
return value
4. チェーンを確認
expect)
5. 戻り値を確認

テストのサンプル

`asyncData` のテスト

nuxt-content の fetch 処理は `asyncData` の中で使うことが多い(ように感じる)のだが、 `asyncData` の定番なテスト方法が不明。

今回は「Nuxtアプリケーションを Jest でテストする - アクトインディ開発者ブログ」を参考に単体でメソッドを実行する方法で対応。ただし、データを使ってレンダリングしたい場合は `setData` ではなくインスタンスを 2 つ作成することを前提とした。

Nuxt のライフサイクル的には正しくなさそうだが「this を使わず引数と戻り値でやりとりする」ようなので Local Vue の設定などをあわせておけばさほど問題にはならないと予想。

サンプル

以下のようなコードをテストする場合。

```
// pages/index.vue
export default Vue.extend({
  async asyncData({ $content, params }) {
    const article = await $content('pages/home').fetch()
    const images = await $content('gallery').sortBy('position').fetch()
    return { article, images }
  },
  // snip...
})
```

テストコード側から `$content` を引数として `asyncData` を実行する。

```
// test/pages/index.ts
const content = mockContent()
const $content = content.$content

const wrapperAsyncData = shallowMount(indexPage, {
    // snip...
})
if (wrapperAsyncData.vm.$options.asyncData) {
  const data = wrapperAsyncData.vm.$options.asyncData({
    $content,
    params: {},
  } as any)
  snip...
}
```

`asyncData` 側では最初の `fetch` で待機状態となるので、 `$content` を検証し `mockResponse` でモックデータを渡す 。

```
// pages/index.vue
const article = await $content('pages/home').fetch()
```

```
// test/pages/index.ts
expect($content).toHaveBeenLastCalledWith('pages/home')
await content.mockResponse(mockDataArticle)
```

続いて２番目の `fetch` で待機状態となるので、再度 `$content` を検証し `mockResponse` でモックデータを渡す。このとき、チェーンリスト `imagesChain` を受け取っておく。

```
// pages/index.vue
const images = await $content('gallery').sortBy('position').fetch()
```

```
// test/pages/index.ts
expect($content).toHaveBeenLastCalledWith('gallery')
const imagesChain = await content.mockResponse(mockDataImages)
```

2 番目の `fetch` では `sortBy` の実行も確認したいので、チェーンリストで確認する。

```
// test/pages/index.ts
expect(imagesChain.at(0).getMockName()).toEqual('sortBy')
expect(imagesChain.at(0)).toHaveBeenCalledWith('position')
```

最後に `asyncData` からの戻り値を検証する。

```
// pages/index.vue
return { article, images }
```

```
// test/pages/index.ts
expect(await data).toEqual({
  article: mockDataArticle,
  images: mockDataImages,
})
```

おわりに

おわりに

個人的には jest-mock-nuxt-content を作成することでテストの記述がすっきりしたと感じている。

しかしながら、実際に作成中のプロジェクト内でテストを記述してみると「モックデータを渡した(待機を解除させた)後にチェーンを検証する」ことには違和感があった。

今後はこの辺の違和感を解消できればと考えている。