



How to Design a REST API

B4 小林紹子

November 5, 2025

目次

- 1 概要
- 2 Step1. Object Modeling
- 3 Step2. Create Model URIs
- 4 Step3. Determine Resource Representations
 - 3.1. Collection Resource
 - 3.2. Single Resource
- 5 Step4. Assigning HTTP Methods
 - データの参照 (GET)
 - デバイスまたは設定の更新 (PUT)
 - デバイスまたは構成の削除 (DELETE)
 - 設定の適用/削除 (PUT/DELETE)

概要

- REST を断片的に学ぶことと, 実際にアプリケーション開発に適用することは全く別の課題.
- 今回はネットワークベースのアプリケーション向けの REST API の設計方法を紹介.
- アプリケーションの設計プロセスで REST の原則を適用させる方法に注目.

REST API の 4 つの設定ステップ

- **Object Modeling :**
リソースとして提示するオブジェクトの特定.
- **Create Resource URIs :**
API のエンドポイントとなるリソース URI の決定.
- **Resource Representations :**
各 URI のリソース表現の設定.
- **Assign HTTP Methods :**
可能なすべての操作を決定. それらを HTTP メソッドを介してリソース URI にマッピング.

Steps to Design a REST API

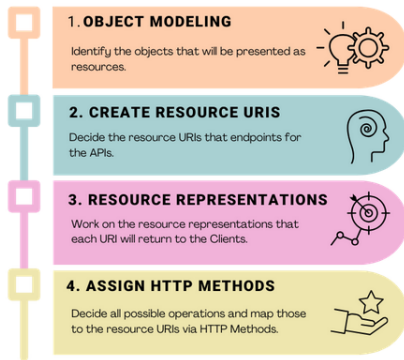


Figure 1: Step to Design a REST API

Step1 : Object Modeling (リソース特定)

- 目的:
アプリケーションで扱うオブジェクト (例: データやデバイス) を特定し, リソースとして定義.
- 主要リソース例:
 - **Devices** (デバイス)
 - **Configurations** (設定)
- リソース同士の関係性:
 - Configuration は Device のサブリソースとして機能することもある.
 - 例: あるデバイスに関する設定は, そのデバイスなしには存在しない.
- 識別子: 全てのリソースは一意の識別子 (例: id) を持つ.

Step2 : Create Model URIs (リソース URI の作成) I

- 目的 : リソースの関係に基づき,URI の構造を決定.
- 設計原則: URI はリソースを識別する名詞であるべき.
- 禁止事項: URI に操作 (動詞) を含めない.
(`/getDevice` や `/createConfiguration` など是不適切).

Step2 : Create Model URIs (リソース URI の作成) II

URI 構造の決定

1 トップレベル・コレクション: 複数形の名詞を使用.

- /devices
- /configurations

2 単一リソース: コレクションに ID を付加.

- /devices/{id}
- /configurations/{id}

3 サブリソース: 親リソースのパスの下に配置.

- /devices/{id}/configurations
- /devices/{id}/configurations/{configId}

※今回は, デバイスをトップレベルリソースとし, 設定がデバイスのサブリソースとする.

Step3. Determine Resource Representations (リソース表現の決定)

- Step2 まででリソース URI が決定された.
- 今回はそのリソース表現（データ形式）に取り組む.
- ほとんどは **XML,JSON** で定義される.
- 今回は XML の例を見ていく.

3.1. コレクションリソースの場合（例: /devices）

- 内容: 最も重要な情報のみを含め, 完全な詳細は省略.
- 目的: レスポンスのペイロードサイズを小さく保ち, パフォーマンスを向上.
- サブコレクションの場合は, プライマリ・コレクションのサブセットであるため, プライマリ・コレクションと異なるデータフィールドを作成しない. 同じ表現フィールドを使用.

3.1. コレクションリソースの場合（例: /devices）II

```
1 <devices size="2">
2
3   <link rel="self" href="/devices"/>
4
5   <device id="12345">
6     <link rel="self" href="/devices/12345"/>
7     <deviceFamily>apple-es</deviceFamily>
8     <OSVersion>10.3R2.11</OSVersion>
9     <platform>SRX100B</platform>
10    <serialNumber>32423457</serialNumber>
11    <connectionStatus>up</connectionStatus>
12    <ipAddr>192.168.21.9</ipAddr>
13    <name>apple-srx_200</name>
14    <status>active</status>
15  </device>
16
```

3.1. コレクションリソースの場合（例: /devices）III

```
17 <device id="556677">
18   <link rel="self" href="/devices/556677"/>
19   <deviceFamily>apple-es</deviceFamily>
20   <OSVersion>10.3R2.11</OSVersion>
21   <platform>SRX100B</platform>
22   <serialNumber>6453534</serialNumber>
23   <connectionStatus>up</connectionStatus>
24   <ipAddr>192.168.20.23</ipAddr>
25   <name>apple-srx_200</name>
26   <status>active</status>
27 </device>
28
29 </devices>
```

3.2. 単一リソースの場合（例: /devices/{id}）

- 内容: 完全な情報を全て含める.
- サブリソースや関連する操作へのリンクリストも含める.
- これにより HATEOAS にすることが可能.

HATEOAS (Hypermedia As The Engine Of Application State)

- クライアントは,API の次に取り得る操作を, リソース表現に含まれるリンクを通じて動的に発見できるべき.

3.2. 単一リソースの場合（例: /devices/{id}）II

```
1 <device id="12345">
2   <link rel="self" href="/devices/12345"/>
3
4   <id>12345</id>
5   <deviceFamily>apple-es</deviceFamily>
6   <OSVersion>10.0R2.10</OSVersion>
7   <platform>SRX100-LM</platform>
8   <serialNumber>32423457</serialNumber>
9   <name>apple-srx_100_lehar</name>
10  <hostName>apple-srx_100_lehar</hostName>
11  <ipAddr>192.168.21.9</ipAddr>
12  <status>active</status>
13
14  <configurations size="2">
15    <link rel="self" href="/configurations" />
16
```

3.2. 単一リソースの場合（例: /devices/{id}）III

```
17 <configuration id="42342">
18   <link rel="self" href="/configurations/42342" />
19 </configuration>
20
21 <configuration id="675675">
22   <link rel="self" href="/configurations/675675" />
23 </configuration>
24 </configurations>
25
26 <method href="/devices/12345/exec-rpc" rel="rpc"/>
27 <method href="/devices/12345/synch-config"rel="synch device
    configuration"/>
28 </device>
```

3.2. 単一リソースの場合（例: /devices/{id}）IV

- 単一デバイスリソースの例:
 - サブリソース（Configuration）へのリンクを含める.
 - そのリソースに対して実行可能なカスタム操作を<method>タグで示す.

```
1      <configuration id="675675">
2          <link rel="self" href="/configurations/675675" />
3      </configuration>
4  </configurations>
5
6      <method href="/devices/12345/exec-rpc" rel="rpc"/>
7      <method href="/devices/12345/synch-config"rel="synch device
8          configuration"/>
9  </device>
```

データの参照 (GET) I

- リソース URI とその表現が確定したので, アプリケーションで可能なすべての操作を決定し, それらをリソース URI にマッピングする.
- 操作の性質 (参照、作成、更新、削除) に基づき、適切な HTTP メソッド (べき等性を考慮) を選択.

データの参照 (GET)

データの取得には, すべて **GET** メソッドを使用 (安全かつべき等) .

対象	URI の例	備考
全コレクション	/devices	サイズが大きい場合は, ?startIndex=... で範囲選択可能.
特定デバイス	/devices/{id}	完全な情報を返す.
サブコレクション	/devices/{id}/configurations	特定のデバイスに紐づく設定リスト.

- 1 HTTP GET /devices?startIndex=0&size=20
- 2 HTTP GET /configurations?startIndex=0&size=20

デバイスまたは設定の更新 (PUT)

更新操作はべき等な操作であるため,**PUT** メソッドを使用.

```
1 HTTP PUT /devices/{id}
2 HTTP PUT /configurations/{id}
```

PUT メソッドのレスポンス例:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/xml
3 <configuration id="678678">
4     <link rel="self" href="/configurations/678678" />
5     <content><![CDATA[. updated content here .]]></content>
6     <status>active</status>
7     <link rel="raw configuration content" href="/configurations
8         /678678/raw" />
9 </configuration>
```

4.7. デバイスまたは構成の削除

削除操作は常に DELETE を使用.

```
1 HTTP DELETE /devices/{id}
2 HTTP DELETE /configurations/{id}
```

削除が成功した場合のレスポンス :

- 非同期削除 (キュー登録) → 202 Accepted
- 同期削除 (即時削除) → 200 OK または 204 No Content

非同期操作では, 成功/失敗を追跡するために **タスク ID** を返す.

※サブリソース削除時は挙動を検討すべき.

一般的には「ソフトデリート」を行う. (ステータスを INACTIVE に設定.)

これにより他の参照を消す必要がなくなる.

4.8. 設定の適用／削除（PUT/DELETE）

実際のアプリケーションでは、構成をデバイスに適用または削除する操作が必要。この場合も幂等性を保つために **PUT** と **DELETE** を使用。

設定をデバイスに適用

```
1 HTTP PUT /devices/{id}/configurations
```

設定をデバイスから削除

```
1 HTTP DELETE /devices/{id}/configurations/{configId}
```

設定そのものを削除するのではなく、デバイスとの関連付けを解除する操作である。