NetOps/DevOps必携の

JSONパーザ

豊沢 聡◉著

CUTT
カットシステム

■**サンプルファイルのダウンロードについて**

本書掲載のサンプルファイルは、下記 URL からダウンロードできます。

https://---------- （出版社 Web サイト）

https://github.com/---------- （著者の GitHub）

・本書の内容についてのご意見、ご質問は、お名前、ご連絡先を明記のうえ、小社出版部宛文書（郵送または E-mail）でお送りください。
・電話によるお問い合わせはお受けできません。
・本書の解説範囲を越える内容のご質問や、本書の内容と無関係なご質問にはお答えできません。
・匿名のフリーメールアドレスからのお問い合わせには返信しかねます。

はじめに

jq は JSON テキストを解析、抽出、変換するコマンドライン指向のツールです。

JSON（JavaScript Object Notation）は構造的なデータを表現する一般的な形式で、ウェブサービスで広く利用されています。とくにウェブ技術を基盤にした REST（REpresentational State Transfer）では、リソースはたいてい JSON で記述されます。REST はリモートシステムの管理運用でよく用いられ、JSON テキストを送信することでシステムを設定したり、GET リクエストから JSON で書かれたシステム情報を取得します。たとえば、協調ソフトウェア開発でよく用いられる GitHub、AWS などのクラウド環境、あるいは F5 BIG-IP のようなネットワーク機器の REST インタフェースは JSON ベースです。次の例は、GitHub からリポジトリ情報を取得したときのものです。

```
$ curl -sku :$TOKEN https://api.github.com/user/repos | jq '.'
[
  {
    "id": 1234567879,
    "node_id": "MDEabcdefghijkRvcnkzNDc3NzQyMzM=",
    "name": "Sample",
    "full_name": "Sample codes",
    "private": false,
    "owner": {
      "login": "foobar",
      "id": 1234567,
      ...                                                  # 以下略
```

JSON テキストは可読文字で記述されるので、テキストエディタで読み書きできます。しかし、データの階層構造が深いと、経験を積んだ目でもかなり読みづらいです。数百個の構成情報が改行なしで 1 行にまとめられることもあり、そんなときは Unix でおなじみの行指向ツールである grep や sed では簡単には解析できません。そこで、やや込み入った処理ではプログラミングのお世話になります。幸いなことに、Python や JavaScript といったポピュラーな言語には JSON 処理ライブラリが用意されているので、スクリプティングはそれほど難しくありません。

しかし、ちょっとした操作のためにわざわざプログラムを書くのは面倒です。Unix ツールのように、必要なときに必要なものだけをいろいろ組みあわせてさくっと利用できたほうが便利です。

jq はそんな用法を念頭に設計されています。つまり、JSON 専用の sed あるいは awk のようなコマンドです。jq は部分データの抽出、不要な要素の削除、値の変更や演算、簡単な集計などを（たいていは）1 行でできます。なれてくれば、より複雑な操作もできます。

jq が略語だとしたら、もとの単語がなにかは気になるところです。JSON Query という説もありますが、設計者の Stephen Dolan によれば、短くて打ちやすいから選んだだけで、もともと意味はないそうです（Stack Overflow より）。

jq のオフィシャルマニュアルは A4 換算で 50 ページほどなので、使いたい機能を見つけるのはそう大変ではありません。ただ、そっけない記述も多く、ときには試行錯誤も必要です。チュートリアルもありますが、あまり拡充されていません。そこで、シンプルなサンプルとその実行例から各種の機能を説明したのが本書です。オフィシャルマニュアルのほとんどをカバーしていますが、一部、利用頻度の少なそうな機能は省いています。

REST を日常的に利用する NetOps や DevOps 諸氏のお役に立てれば幸いです。

2021 年 6 月
豊沢 聡

注意事項

以下、本書の表記や使用するサンプルファイルなどで注意すべき点を説明します。

表記

コマンドとその出力を示す実行例は、すべて Unix でのものです。$ で始まる行がコマンド、それに続くのが出力です。例によっては、# からコメントを加えたものもあります。

JSON の仕様にコメントはありません。あくまで、本書を読むための補助にすぎません。

```
$ jq -c '.' startMeUp01.json                    # コマンド
["烧餐包","烧卖皇","虾饺","韭菜饺"]                    # 出力
```

コマンドが 1 行に収まらないときは、シェルに解釈される部分では行末にバックスラッシュ（\）を加えて改行しています。jq に解釈されるフィルタ（シングルクォートで囲まれた jq の第 1 引数）の部分ではそのまま（記号なしで）改行しています。どちらの記法でも、（$ やコメントを除けば）そのままコピー & ペーストで実行できます。

```
$ cat filter01.json | \                         # シェルでの折り返し
  jq -c '.'

$ cat control01.json | jq -r '.aLaCarte[] |     # フィルタでの折り返し
  if endswith("焼き") then
    .
  else
    . + "は焼きものではない"
  end'
```

日本語環境の Windows コマンドプロンプトでは、Windows Subsystem for Linux（WSL）も含めてバックスラッシュ（\）は ¥ で表示されます。

縦に長い出力は、全体を示す必要がなければ省略しています。

```
[
  "焼餐包",
  "焼売皇",
  ...                                                    # 省略部分
```

横に長い出力は書籍の幅にもとづいて、そのまま折り返しています。つまり、書籍の幅のコンソールに出力されたのとおなじ見栄えです。

```
{"赤皿":{"品目":["こはだ","とびっこ軍艦","とろたく軍艦","納豆巻"],"価格":130},"青皿":{"
品目":["びんちょう","〆さば","あじ","いわし"],"価格":210},"金皿":{"品目":["うなぎ","あわ
び","中トロ","いくら軍艦"],"価格":430}}
```

ただし、エラー出力は可読性を考慮して、手で改行を入れています。実際の出力と異なる点、ご了承ください。

```
# 本書での表示
jq: error: syntax error, unexpected INVALID_CHARACTER,
  expecting FORMAT or QQSTRING_START (Unix shell quoting issues?)
  at <top-level>, line 1:

# もとの表示（自然な折り返し）
jq: error: syntax error, unexpected INVALID_CHARACTER, expecting FORMAT or QQSTRING_STAR
T (Unix shell quoting issues?) at <top-level>, line 1:
```

エラーメッセージは jq のバージョンで異なることもあります。

配列要素番号

配列要素の番号は、0 からカウントしています。たとえば、配列 [1, 2, 3] の要素 1 は 0 番目の要素です。

サンプルファイル

本書のほとんどの実行例は、サンプルの JSON テキストから示します。いずれもたかだか 20 行程度の短いものなので各章の冒頭に提示してありますが、出版社のダウンロードサービスあるいは次に示す筆者の GitHub からダウンロードもできます。文字エンコーディングは UTF-8 です。

```
https://github.com/stoyosawa/jqDoc-Public
```

もっとも、ファイルの中身そのものに意味があるわけではないので、似たような構造なら好みのファイルでテストしても問題はありません。

jq のバージョン

本書で使用し、動作を確認した jq はバージョン 1.6（2018 年 11 月 1 日リリース）です。OS にバンドルされているもの、あるいは apt-get や yum などのパッケージ管理ツールからインストールしたものは、これよりも古いバージョンのことがあります。エラーが頻繁に発生するようなら、付録 A.1 から実行形式をインストールしてください。

（jq のオフィシャルサイトのバイナリではない）バンドル版やパッケージ版にはまた、正規表記ライブラリ抜きでコンパイルされたものも散見されます（たとえば F5 社の BIG-IP にバンドルされたもの）。そのタイプのバイナリでは、6.2 節で取り上げる正規表現関数は利用できません。オフィシャル版のインストールをお勧めします。

Windows での利用

jq には Windows 用バイナリもありますが、Windows コマンドプロンプトからの直接利用は勧められません。特殊文字や非 ASCII 文字の扱いに癖があるため、込み入った jq フィルタの記述が難しいからです。Windows ユーザには Windows Subsystem for Linux（WSL）を有効化し、そのコンソールから Linux 版バイナリを使用することを勧めます。WSL の導入と簡単な利用方法は付録 A.2 に示しました。

あえて Windows コマンドプロンプトから利用するにあたっては、引用符にはとくに注意してください。本書の用例では、jq フィルタの特殊記号をシェルに解釈させないようにシングルクォート（'）でくくっていますが、コマンドプロンプトではシングルクォートは使いません。Windows で本書の用例をそのまま用いるとエラーになります。

```
$ jq '.contact.phone' startMeUp02.json                # Unix。フィルタを''でくくる
"09-123-4567"

C:\temp>jq '.contact.phone' startMeUp02.json          # Windows。''でくくるとエラー
jq: error: syntax error, unexpected INVALID_CHARACTER,
  expecting $end (Windows cmd shell quoting issues?) at <top-level>, line 1:
'.contact.phone'
jq: 1 compile error

C:\temp>jq .contact.phone startMeUp02.json            # Windows。''がなければ動作
"09-123-4567"
```

Windows コマンドプロンプトのデフォルト文字エンコーディングは Shift JIS（SJIS）です。UTF-8 で記述されたファイルは、次の例に示すように文字化けします。

```
C:\temp>chcp                                          # デフォルトはSJIS
現在のコード ページ: 932

C:\temp>type predicate01.json                         # UTF-8は文字化け
{
  "null": null,
  "boolean": true,
  "number": 810,
  "string": "縺輔・縺ｮ蜻ｳ蝎檎・蝗夐」・,
  "array" : ["縺秘」ｯ", "蜻ｳ蝎梧ｱ・, "蜑ｲ繧雁ｹｲ縺怜､ｧ譬ｹ"],
  "object": {"drink": "譯懈干蜷滄・驟・}
}
```

しかし、ファイルを SJIS にすると、コマンドプロンプトで表示はできても UTF-8 マシンの jq では次に示すように処理できません。

```
C:\temp>type predicate01-sjis.json                          # SJIS版は表示できる
{
  "null": null,
  "boolean": true,
  "number": 810,
  "string": "さばの味噌煮定食",
  "array" : ["ご飯", "味噌汁", "割り干し大根"],
  "object": {"drink": "桜花吟醸酒"}
}

C:\temp>jq '.' predicate01-sjis.json                        # 処理失敗
jq: error: syntax error, unexpected INVALID_CHARACTER,
  expecting $end (Windows cmd shell quoting issues?) at <top-level>, line 1:
'.'
jq: 1 compile error
```

目次

第1章

はじめよう

まずは使ってみましょう。

jq が未導入なら、付録 A からインストールしてください。また、JSON テキストの構造や記述方法に不明な点があれば、付録 B に示した JSON の仕様概略を参照してください。

本章のテスト用 JSON ファイルは、次に示す startMeUp01.json と startMeUp02.json です。

```
$ cat startMeUp01.json
["烧餐包", "烧卖皇", "虾饺", "韭菜饺"]
```

4 つの文字列要素を収容した配列ひとつからなるシンプルなものです。

```
$ cat startMeUp02.json
{"name":"別不同","isOpen":false,"評価":4.5,"hours":["11:00","15:00"],"contact":{"phone":
"09-123-4567","メール": "booking@starcafe.com"}
```

5 つのプロパティからなるオブジェクトです。プロパティの値は順に文字列、真偽値、数値、配列、オブジェクトです。すべての要素が改行やインデントなしで 1 行に収められているので、かなり読みにくいです（紙面ではコンソール出力時のように自然に折り返されています）。このようにレスポンスボディを整形せずに返す REST サーバも多くあり、このサンプルはそのような状態をシミュレートしています。

1.1 jq の実行

jq は JSON テキストを整形して表示します。テストファイルからためします。

```
$ jq . startMeUp01.json
[
  "焼餐包",
  "焼卖皇",
  "虾饺",
  "韭菜饺"
]

$ jq . startMeUp02.json
{
  "name": "別不同",
  "isOpen": false,
  "評価": 4.5,
  "hours": [
    "11:00",
    "15:00"
  ],
  "contact": {
    "phone": "09-123-4567",
    "メール": "booking@starcafe.com"
  }
}
```

出力は、JSON テキストの構造が把握しやすくなるよう整形されます。配列は要素単位で、オブジェクトはプロパティ単位で改行されます。加えて、値が配列など入れ子で構造化されている startMeUp02.json の hours や contact プロパティでは、それらがさらにインデントされます。

1.1.1 コマンドの書式

jq コマンドの書式は次のとおりです。

```
$ jq [options] filter file.json
```

- options: 任意指定（optional）なコマンドオプション。jq のデフォルトの挙動を変更するときに用いるもので、前記の実行例では用いていません。1.3 節で説明します。
- filter: フィルタ。入力された JSON テキストをどのように処理するかの指示。前記の実行例ではドット（.）ひとつです。本書はこのフィルタの書き方を 1 冊かけて説明していきます。コマンドオプションを介して、ファイルに記述したフィルタを読み込むこともできます（2.2 節）。
- file.json: 入力 JSON テキストファイル。JSON の仕様にのっとったテキストファイルでなければなりません。仕様は付録 B を参照してください。ファイルからだけでなく、標準入力あるいはフィルタ内に記述した直値からでも入力できます。入力方法は 2.1 節で説明します。

1.1.2 フィルタの保護

上記のフィルタは . だけで構成されていますが、複雑なものになるとスペースや特殊記号も含まれます。フィルタの特殊記号がシェルに先行して解釈（展開）されないようにするため、フィルタはシングルクォート（'）でくくります。

特殊記号が含まれたフィルタをシングルクォートなしで記述すると、エラーが発生したり、思わぬ挙動を示すことがあります。次の例で、フィルタ ., .（ドット、カンマ、スペース、ドット）を用いたときの動作を示します（カンマの用法は 3.2 節）。jq の文法上適正なフィルタなのでシングルクォートでくくれば正常に動作しますが、くくられないとエラーが発生します。

```
$ jq '., .' startMeUp01.json                              # シングルクォートされている
[
  "焼餐包",
  "焼卖皇",
  ...                                                     # 以下略

$ jq ., . startMeUp01.json                                # ないとエラー
jq: error: syntax error, unexpected $end (Unix shell quoting issues?)
  at <top-level>, line 1:
```

```
.,
jq: 1 compile error
```

シェルはコマンド行をスペースで分解するので、指定のフィルタは .,（ドットカンマ）、.（ドット）、startmeUp01.json の 3 つの文字列として jq に引き渡されます。jq は最初の ., をフィルタと解釈しますが、これは文法上不正です（中途半端に終わっている）。そのため、「1 行目の ., に文法上のエラーがあります」（意訳）というエラーを発して異常終了します。また、「Unix シェルのクォートはちゃんとしてる ?」とのヒントも示されています。

単一の . のように特殊記号を含まない 1 単語のフィルタならシングルクォートは必要ありませんが、いつもくくるように習慣づけるとよいでしょう。

Windows ではフィルタをシングルクォートでくくると逆にエラーになります。

1.1.3 不正な JSON テキスト

JSON の仕様に反したテキストはエラーになります。たとえば、startMeUp01.json ファイル末尾の閉じ角括弧（]）を削除するとエラーが報告されます。

```
$ cat startMeUp01.json                                    # 末尾の]がない
["焼餐包", "焼卖皇", "虾饺", "韭菜饺"

$ jq '.' startMeUp01.json                                 # エラー
parse error: Unfinished JSON term at EOF at line 2, column 0
```

1.2 フィルタ

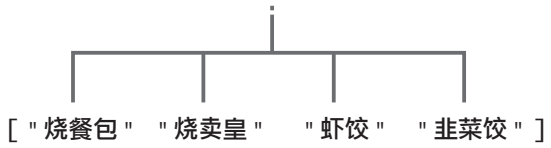
入力 JSON テキストの処理方法を記述したフィルタはどこまでも複雑になりえますが、本節では、入力から所定の要素を取りだすベーシックな記法を紹介します。

1.2.1 アイデンティティ

もっともシンプルなフィルタは、. だけで構成されたものです。ドットは「その JSON テキスト」のトップレベルを指す記号で、マニュアルはこれをアイデンティティ（identity）と呼んでいます。

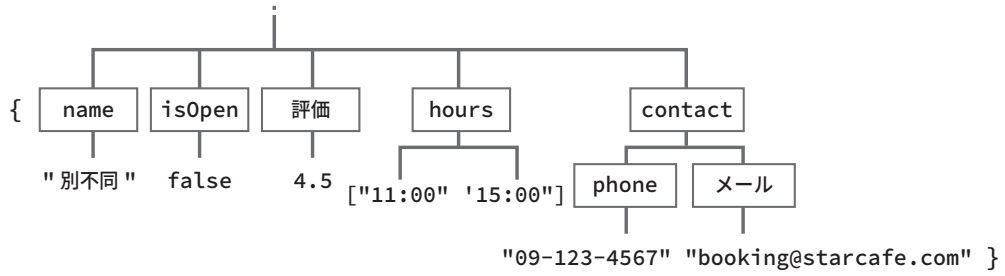
配列がひとつだけ収容された startMeUp01.json なら、前後をくくる [] も含めてドット直下のレベルにある 4 つの要素すべてを指します。

図 1.1 ● startMeUp01.json の構造



startMeUp02.json なら、ドットを起点とした階層構造で示される JSON テキスト全体です。これには全体をくくる {} も含まれます。また、hours や contact のように値がさらに構造化されているときはその中身も再帰的に表示されます。

図 1.2 ● startMeUp02.json の構造



1.2.2 オブジェクトへのパス

JSON テキストにある . 配下の要素は、. を起点としたパス構造で参照できます。たとえば、startMeUp02.json のトップレベルにある name プロパティには .name から、isOpen プロパティには .isOpen からアクセスできます。このように、ドットを起点にした要素の参照をパス（path）といいます。ファイルシステム内のカレントディレクトリ（.）からその配下のディレクトリのファイルを指定するのとおなじ要領です。

```
$ jq '.name' startMeUp02.json                    # nameの値だけを抽出
"別不同"

$ jq '.isOpen' startMeUp02.json                  # isOpenの値だけを抽出
false
```

出力はそのプロパティの値です。プロパティ名（キー）は出力されません（キーとともに出力する方法は 8.2 節）。name プロパティの値は文字列なので、出力もダブルクォートでくくられています。isOpen プロパティの値は真偽値なのでくくられません。

プロパティ hours および contact の値は、それぞれ配列とオブジェクトです。そのため、.contact をパスに指定すると、. 同様、その位置から下の要素をすべて示します。

```
$ jq '.contact' startMeUp02.json                 # contactの値を抽出
{                                                # 結果はオブジェクト
  "phone": "09-123-4567",
  "メール": "booking@starcafe.com"
}
```

contact プロパティの値であるオブジェクトの内部の phone プロパティを参照するときは、.contact.phone のようにプロパティ名をドットでチェーンします。ファイルシステムのアナロジーでいえば、カレントディレクトリ配下の contact ディレクトリにあるファイル name を参照しているイメージです。

```
$ jq '.contact.phone' startMeUp02.json           # contactのphoneの値
"09-123-4567"
```

1.2.3 特殊なプロパティ名

startMeUp02.json には、日本語（Unicode 文字）で記述したプロパティ名があります。トップレベルの " 評価 " と contact プロパティの " メール " です。英数文字以外は特殊文字なため、プロパティ名をダブルクォート（"）でくくらなければなりません。つまり、." 評価 "、.contact." メール " です。クォートなしで参照するとエラーになります。

JSON のプロパティ名に使ってはならない文字はありません。ただし、疑問符（?）、バックスラッシュ（\）、制御文字（ASCII コードで 0x00 から 0x1f）はバックスラッシュでエスケープしなければなりません。

```
$ jq '.評価' startMeUp02.json                              # くくられていないのでエラー
jq: error: syntax error, unexpected INVALID_CHARACTER (Unix shell quoting issues?)
  at <top-level>, line 1:
.評価
jq: error: try .["field"] instead of .field for unusually named fields
  at <top-level>, line 1:
.評価
jq: 2 compile errors

$ jq '."評価"' startMeUp02.json                            # ."評価"はOK
4.5

$ jq '.contact.メール' startMeUp02.json                    # 裸のメールなのでエラー
jq: error: syntax error, unexpected INVALID_CHARACTER,
  expecting FORMAT or QQSTRING_START (Unix shell quoting issues?)
  at <top-level>, line 1:
.contact.メール

jq: 1 compile error

$ jq '.contact."メール"' startMeUp02.json                  # "メール"をくくればOK
"booking@starcafe.com"
```

エラーの２行目は「普通ではないプロパティ名なら .["field"] を使え」と、"" でくくったプロパティ名をさらに [] でくくるように勧めています。たとえば、.[" 評価 "] です。これも適正な用法ですが、本書では使いません。

ダブルクォートでくくるときは、プロパティ名だけくくるよう注意します。アイデンティティ . とプロパティ名の組みあわせだからです。". 評価 " や ."contact. メール " のように全体を一気にくくると、思わぬ結果が得られます。

たとえば、ドットはクォートの外にでているものの、."contact. メール " はプロパティが存在しないという意味で null を出力します。contact の配下の " メール " ではなく、トップレベルの contact. メールという 1 単語のプロパティ名を参照しているからです。

```
$ jq '."contact.メール"' startMeUp02.json                    # そのようなプロパティはない
null
```

1.2.4 配列要素へのパス

配列要素を要素番号から参照するときのパスは、[n] です。n は 0 からカウントします。startMeUp01.json には 4 つの要素が収容されているので、n の値は 0 から 3 の範囲です。参照するには、トップレベルからスタートしてその配下の配列を参照するという意味で . を先頭に加えなければなりません。たとえば、(0 からカウントして) 1 番目の要素である " 焼売皇 " は .[1] から参照します。

```
$ jq '.[1]' startMeUp01.json                                 # 1番目の要素
"焼売皇"
```

配列範囲外の要素を指定すると、一般的なプログラミング言語ではエラー (Out of Range など) が発生しますが、jq は存在しないプロパティ名同様、null を出力します。

```
$ jq '.[100]' startMeUp01.json                               # 範囲外
null
```

startMeUp02.json の hours プロパティの値は配列です。その要素へのアクセスも、上記とおなじ要領で [n] を用います。ただし、トップレベルからみて hours プロパティの配列要素なので .hours を先頭に加えます。

```
$ cat startMeUp01.json | jq '.hours[0]'                    # hours配列の0番目の要素
"11:00"

$ cat startMeUp01.json | jq '.hours[1]'                    # hours配列の1番目の要素
"15:00"
```

1.2.5 イテレータ

角カッコ [] はなかに数値（配列）あるいは文字列（オブジェクト）が記述されないときはイテレータと解釈されます。イテレータ（iterator）は配列やオブジェクトの要素を順に取りだして処理するメカニズムで、端的には要素数分のループです。イテレータを作用させると、その要素が順に出力されます。次の例では、トップレベルの . にイテレータ [] を作用させることで、4 つの要素を順に取りだしています。

```
$ jq '.[]' startMeUp01.json                                # 4つの要素ぜんぶ
"焼餐包"
"焼卖皇"
"虾饺"
"韭菜饺"
```

出力結果は単体の . と似ていますが、やや異なります。

. は「トップレベルからみたその JSON テキスト全体」なので、出力は配列です。そのことは、出力の前後に置かれた [] から示されています。つまり、フィルタ . はひとつの JSON テキスト（中身は 4 要素の配列）を受け取り、そのままひとつの JSON テキストとして（整形しつつ）出力します。

これに対し、.[] は「トップレベルからみたその JSON テキスト（の配列）を要素単位で抽出し、ループしながらすべて書きだす」という意味なので、4 つの要素をくくる [] はありません。このとき、4 つの要素はそれぞれ独立した JSON テキストです（{} や [] でくくられていない生の文字列ひとつ、数値ひとつだけでも立派な JSON テキストです）。言い換えれば、もとの（配列という）構造はなくなります。

イテレータ [] はオブジェクトにも利用できます。その場合、要素番号ではなくプロパティ名（キー）でループし、その値を順に取りだします。次の例では、startMeUp02.json の contact プロパティの値であるオブジェクトの値を順次抽出しています。

```
$ jq '.contact[]' startMeUp02.json                    # contactの値を抽出
"09-123-4567"
"booking@starcafe.com"
```

startMeUp02.json 全体に適用すれば、トップレベルのプロパティの値をすべて取得できます。

```
$ jq '.[]' startMeUp02.json
"別不同"                                               # nameの値
false                                                 # isOpenの値
4.5                                                   # "評価"の値
[                                                     # hoursの値
  "11:00",
  "15:00"
]
{                                                     # contactの値
  "phone": "09-123-4567",
  "メール": "booking@starcafe.com"
}
```

hours や contact のように値がさらに配列やオブジェクトのときは、構造を保ったまま表示されるところに注意してください。

構造をすべて解消して値だけ取り出すには、再帰のダブルドット（..）と基本型だけを選択的に抽出する関数 scalars をパイプ（|）で組みあわせます。これらは 3.6 節、4.2 節、3.3 節でそれぞれ説明します。

```
$ jq '.. | scalars' startMeUp02.json
"別不同"                                               # nameの値
false                                                 # isOpenの値
4.5                                                   # "評価"の値
"11:00"                                               # hoursの0番目の値
"15:00"                                               # hoursの1番目の値
"09-123-4567"                                         # contactのphoneの値
"booking@starcafe.com"                                # contactの"メール"の値
```