FHIRのREST API in nutshell

日本医療情報学会 課題研究会 FHIR研究会 セミナー

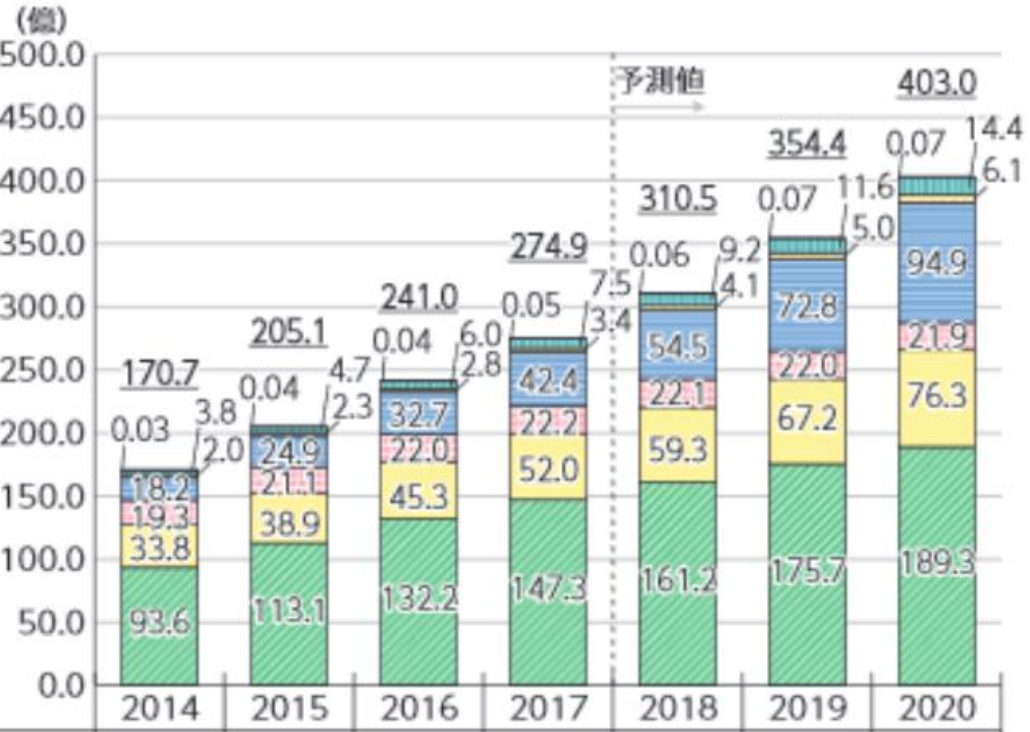
群馬大学医学部附属病院 システム統合センター　鳥飼　幸太
東京大学医学部附属病院 企画情報運営部　土井　俊祐

本日の内容

- **HL7 FHIRとRESTful API**
  - FHIRの基本的アーキテクチャ
  - 薦められる構築方法
  - FHIRにおけるREST API（データの操作法）について
  - CRUD操作についての解説
- **FHIRのREST APIを利用したコーディング例**
- **まとめ**

IoTデバイスの広く浸透する医療への変化

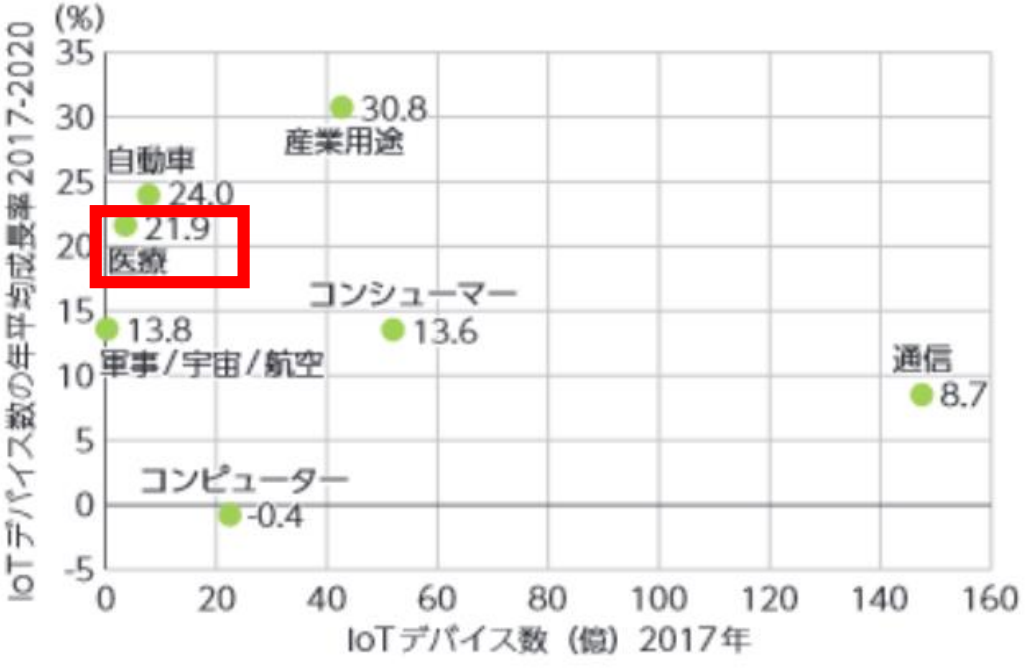
図表1-1-2-1 世界のIoTデバイス*2数の推移及び予測



	2014	2015	2016	2017	2018	2019	2020
合計	170.7	205.1	241.0	274.9	310.5	354.4	403.0
軍事・宇宙・航空	0.03	0.04	0.04	0.05	0.06	0.07	0.07
自動車	3.8	4.7	6.0	7.5	9.2	11.6	14.4
医療	2.0	2.3	2.8	3.4	4.1	5.0	6.1
産業用途	18.2	24.9	32.7	42.4	54.5	72.8	94.9
コンピューター	19.3	21.1	22.0	22.2	22.1	22.0	21.9
コンシューマー	33.8	38.9	45.3	52.0	59.3	67.2	76.3
通信	93.6	113.1	132.2	147.3	161.2	175.7	189.3

(出典) IHS Technology

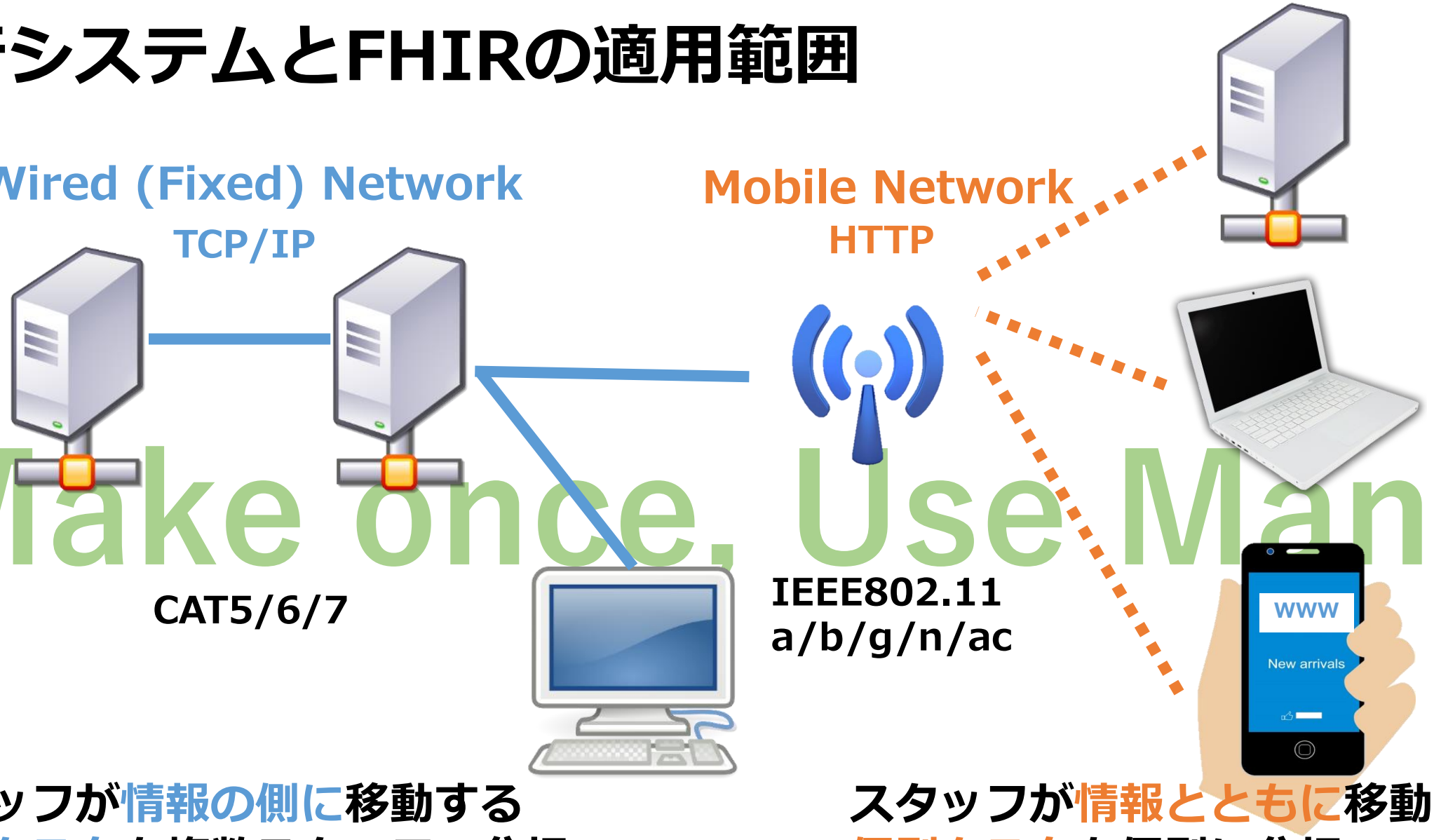
図表1-1-2-2 分野・産業別のIoTデバイス数及び成長率予測



(出典) IHS Technology

総務省 平成30年版 通信情報白書

現行システムとFHIRの適用範囲



“Make once, Use Many”

スタッフが情報の側に移動する
共通タスクを複数スタッフで分担
部門化の流れとともに浸透
共通の端末

スタッフが情報とともに移動する
個別タスクを個別に分担
チーム医療・情報の電子化、即時化
目的ごとのデバイス

FHIRはこれまでの規格と何が違うのか

- FHIRが優れている理由
  - 実装性に強力にフォーカス：速く、簡単に実装できる
    （複数の開発者がたった1日で簡単なインターフェイスを構築できた）
  - クイックスタートを可能とする多くの実装ライブラリが準備されている
  - 革新的な相互運用性：ベースとなるリソースは用意されているが、既に利用されているプロファイルやエクステンション、用語集などを採用できる
  - 既存のHL7 v.2、CDAとは相互互換があり、両方活用できる
  - JSON, XML, HTTPといったWeb標準の強力な基盤を使用できる
  - RESTfulアーキテクチャ、メッセージとドキュメントを使用したシームレスな情報交換、サービスベースのアーキテクチャをサポート

誰でも参加しやすい、参入しやすい

既存の資産も活用できる

Web標準＝医療特化の脱却

HL7 FHIR ポータルサイト「2.17.1 Why FHIR is better」を一部抜粋し翻訳（https://www.hl7.org/fhir/summary.html）

RESTとは

- **REST(REpresentational State Transfer)**
  - Web開発において最も一般的な通信技術の1つ
  - **リソース**と呼ばれる同じ性質を持つ情報の断片（例：Patient, Observation）と、**リソースを一意に識別するURI**で表されるアドレスを持つ

RESTで使用するURLの例

"http://hapi.fhir.org/baseDstu3/Patient/1646554/_history/1?_format=json"

リソースの場所（WebサイトのURL）　　**引数（取得するデータの条件指定）**

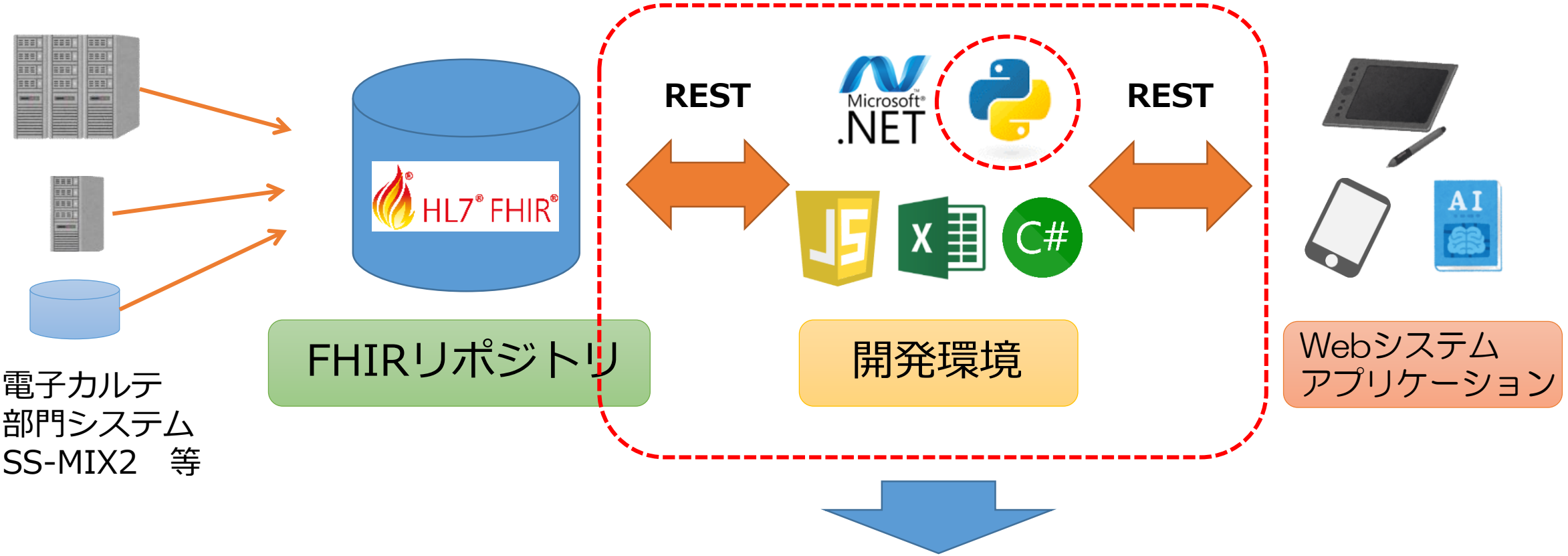
RESTサービスの設計原則

1. **アドレス指定可能なURIで公開されていること**
2. **インターフェース(HTTPメソッドの利用)の統一がされていること**
3. **ステートレスであること**
4. **処理結果がHTTPステータスコードで通知されること**

処理	HTTPメソッド	CRUD操作
登録	POST	CREATE
取得	GET	READ
更新	PUT	UPDATE
削除	DELETE	DELETE

REST通信を利用することで…

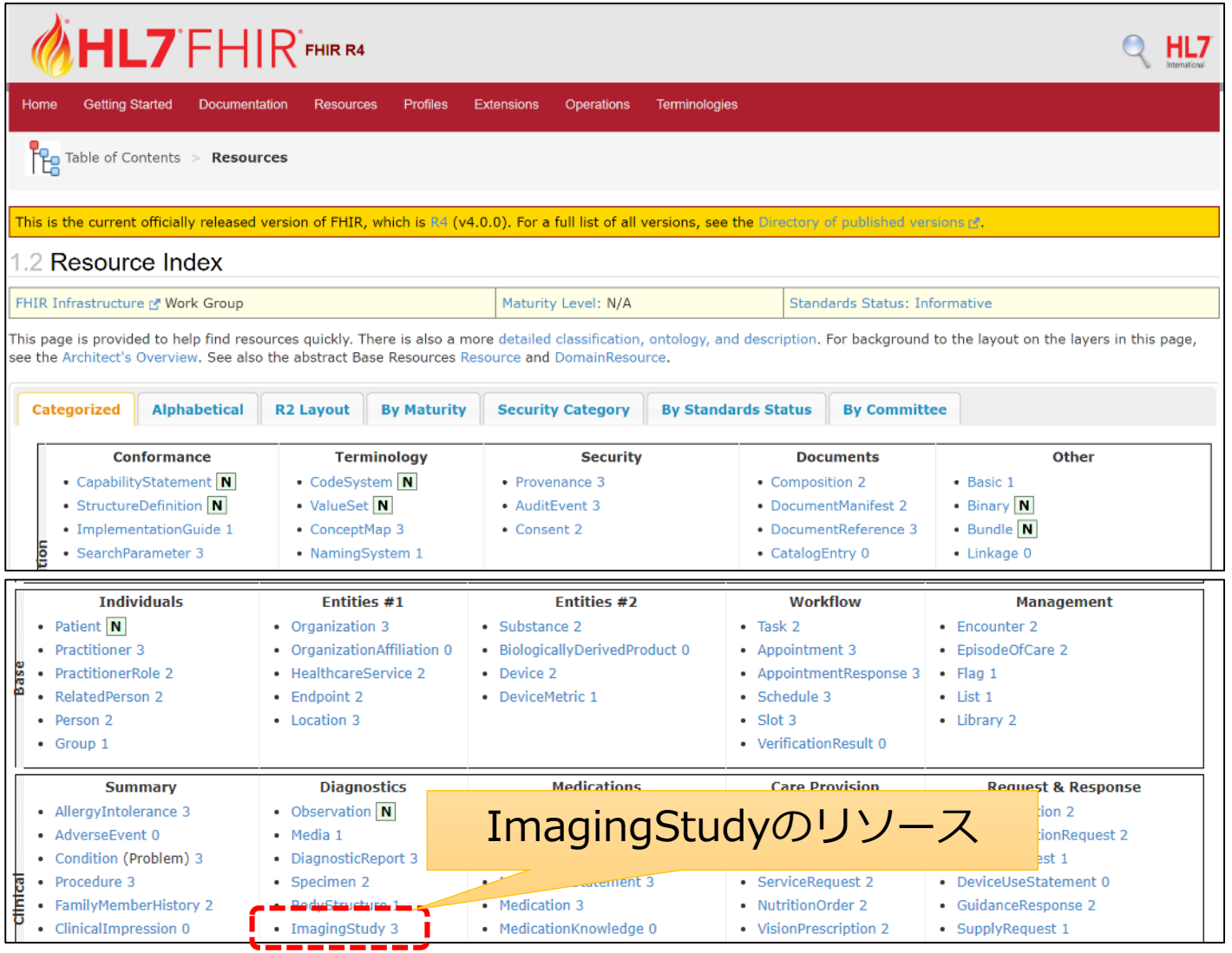
- FHIRを現場に導入するまでの道のり



REST通信はWeb標準
開発の敷居の低さを体感してもらおう！

Resource:リソースとは

- FHIRリポジトリに保存される「構造をもった」データの「種別」
- Patient, Observation, ImagingStudy,…



10.4.3 Resource Content

Structure | UML | XML | JSON | Turtle | R3 Diff | All

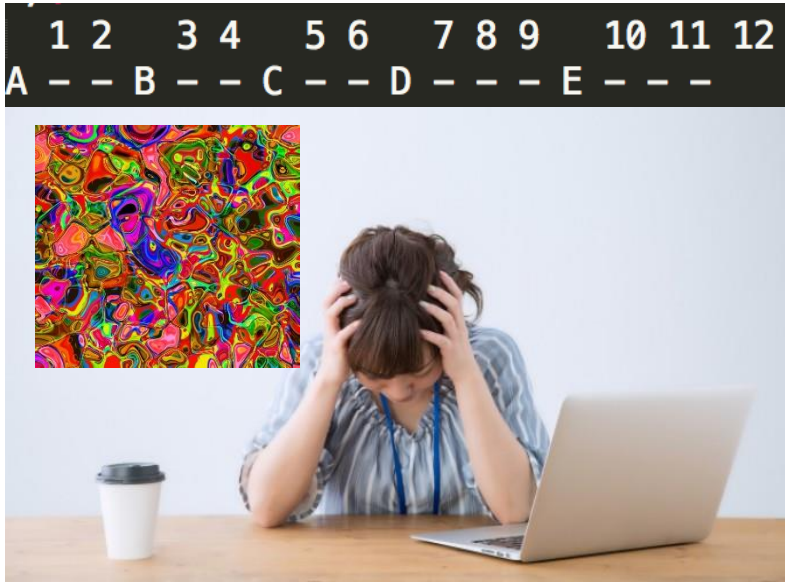
Structure

Name	Flags	Card.	Type	Description & Constraints
ImagingStudy	TU		DomainResource	A set of images produced in single study (one or more series of references images) Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtensic
identifier	Σ	0..*	Identifier	Identifiers for the whole study
status	?! Σ	1..1	code	registered | available | cancelled | entered-in-error | unknown ImagingStudyStatus (Required)
modality	Σ	0..*	Coding	All series modality if actual acquisition modalities AcquisitionModality (Extensible)
subject	Σ	1..1	Reference(Patient | Device | Group)	Who or what is the subject of the study
encounter	Σ	0..1	Reference(Encounter)	Encounter with which this imaging study is associated
started	Σ	0..1	dateTime	When the study was started
basedOn	Σ	0..*	Reference(CarePlan | ServiceRequest | Appointment | AppointmentResponse | Task)	Request fulfilled
referrer	Σ	0..1	Reference(Practitioner | PractitionerRole)	Referring physician
interpreter	Σ	0..*	Reference(Practitioner | PractitionerRole)	Who interpreted images
endpoint	Σ	0..*	Reference(Endpoint)	Study access endpoint
numberOfSeries	Σ	0..1	unsignedInt	Number of Study Related Series
numberOfInstances	Σ	0..1	unsignedInt	Number of Study Related Instances
procedureReference	Σ	0..1	Reference(Procedure)	The performed Procedure reference
procedureCode	Σ	0..*	CodeableConcept	The performed procedure code ImagingProcedureCode (Extensible)
location	Σ	0..1	Reference(Location)	Where ImagingStudy occurred
reasonCode	Σ	0..*	CodeableConcept	Why the study was requested

ImagingStudyリソースの構造（階層構造）
HL7協会がリソースの定義を行ったもの

それぞれのリソースへのアクセスは、
RESTのURIで特定することができる
→**URIさえわかればHTTPでデータを取得したり、登録・修正したりできる**

ハイライト：FHIRのREST APIはURI/CRUDにどのような文法を求めるのか



日常的なURIの世界では、サーバー管理者は
「自らのルールで」呼び出し構造を定義する
放置すれば再び「つながらない混沌」を生じる原因に

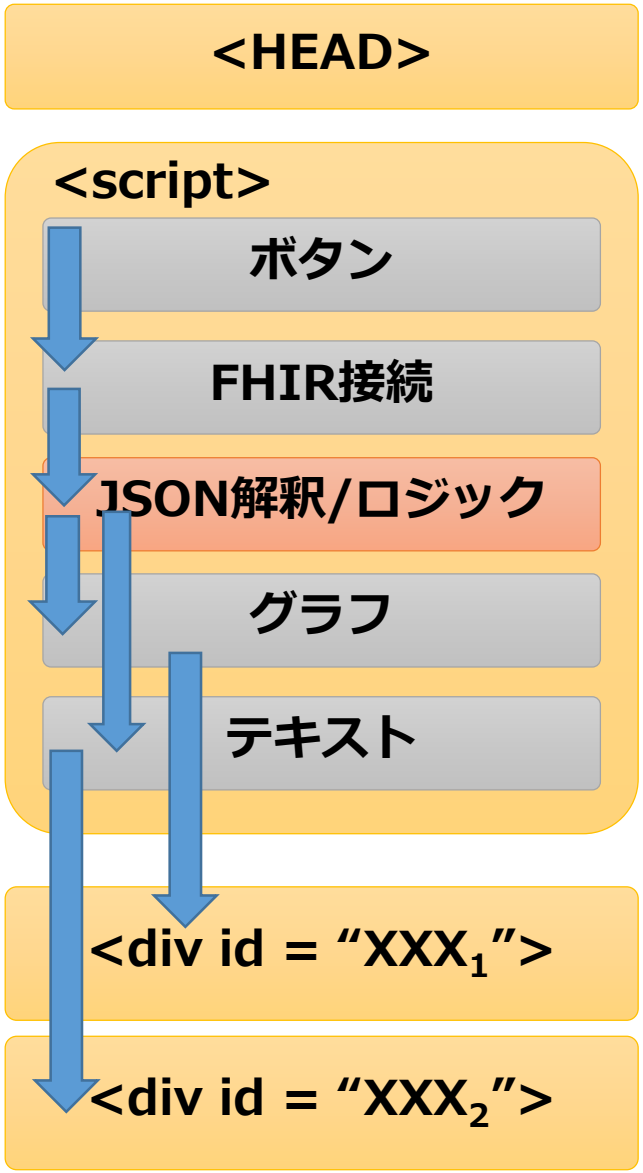
→FHIRの基本コンセプトとしてREST APIを用いる
→WebシステムのHTTPでの呼び出しを基本とする、ということ

＋

→APIにおける「呼び出し名」「パラメータ」「記載順序」について規約を制定すること

文法構造について見ていきましょう

（2）JavaScriptでREST

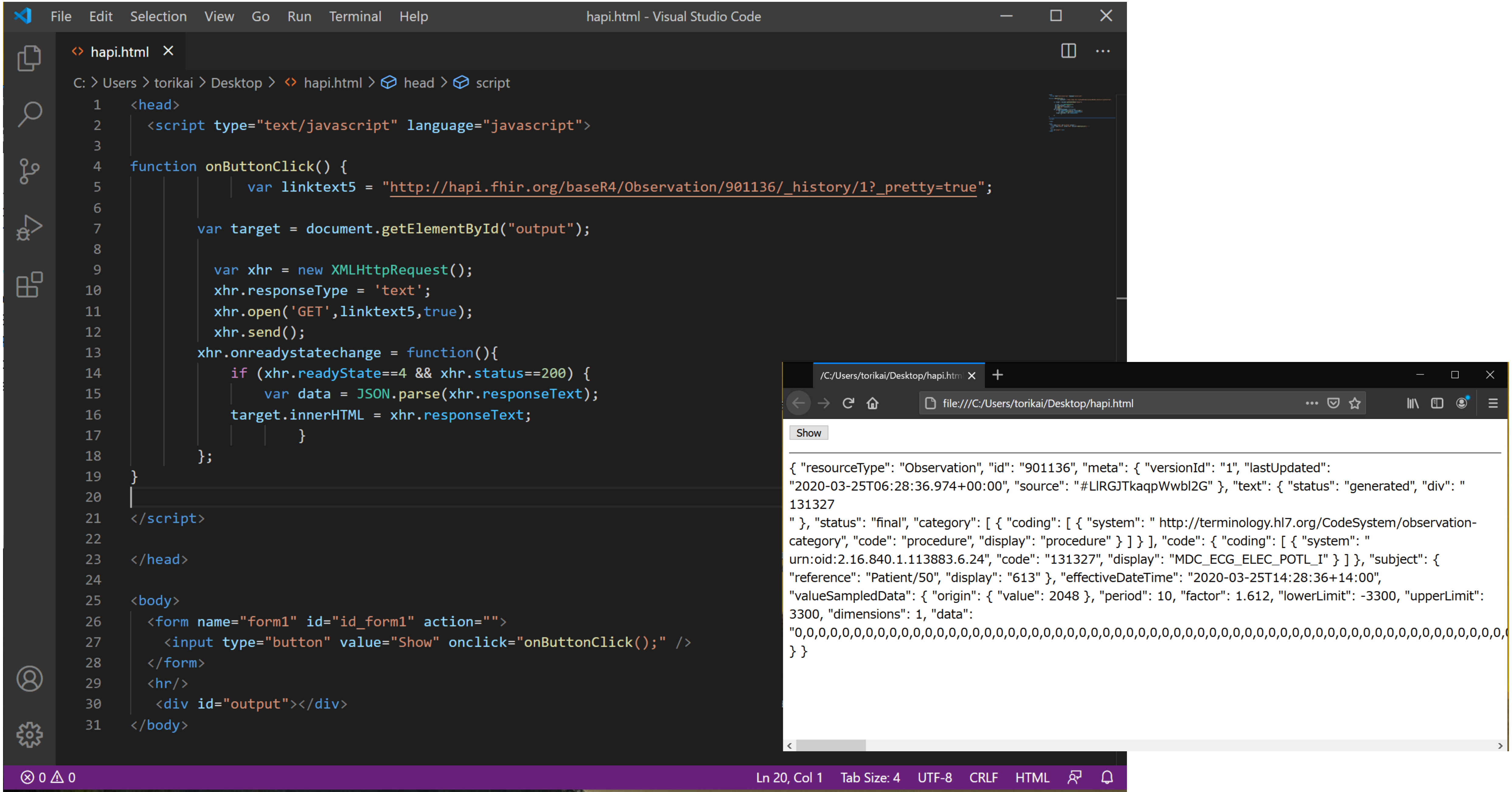


HTML内で動くプログラムはJavaScriptで、これにより
①外部サーバ(HapiFHIR)と接続し、
②JSONデータを受け取り、
③アルゴリズムでデータを抽出し、
④表示のライブラリが指定するフォーマットで
　データ(HTML書式)を渡す

というHTMLコードを出力している

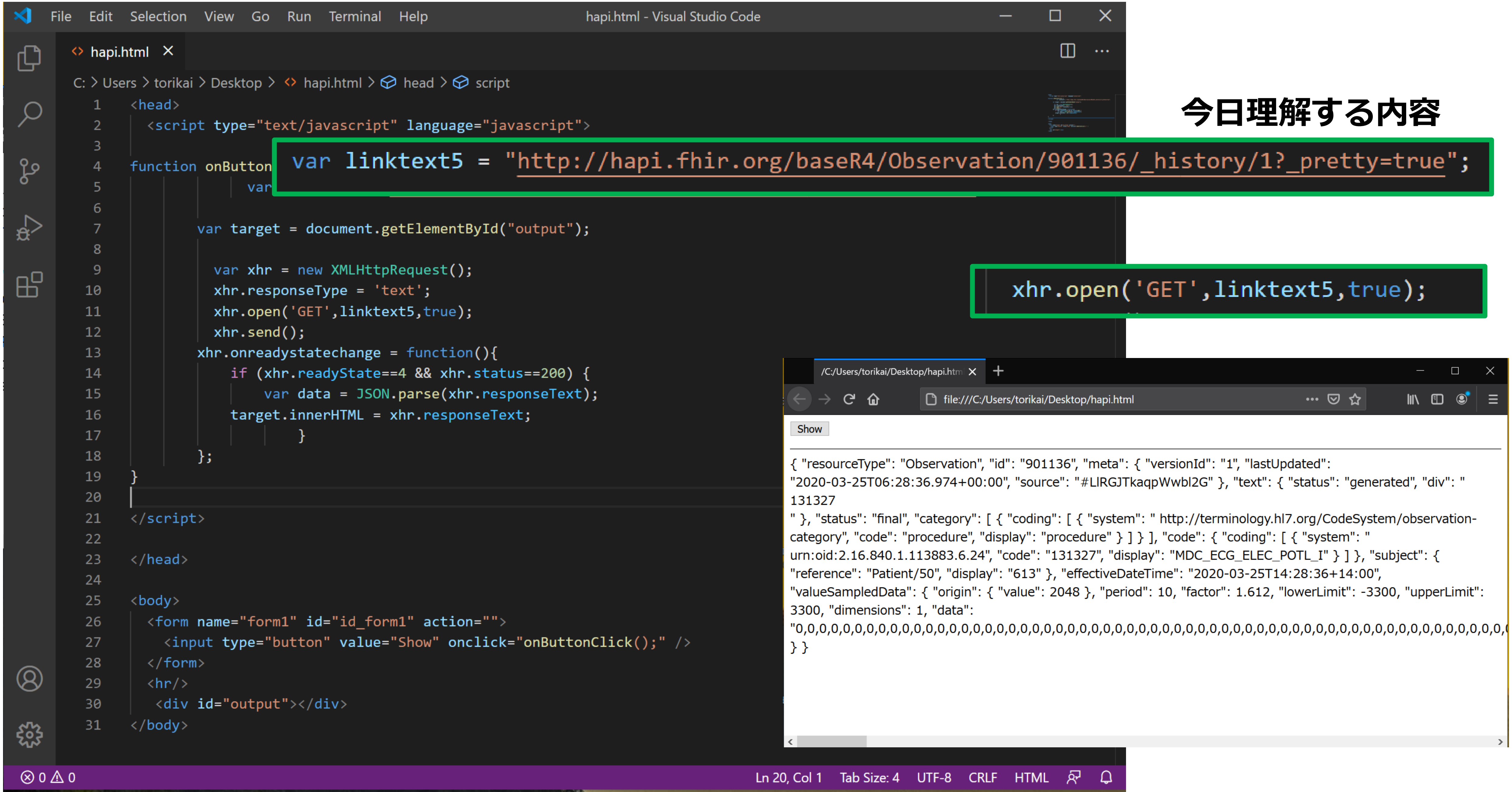
Chart.jsのライブラリ呼び出し

実践例：HTML/JavaScript（この画面例のみで動きます）

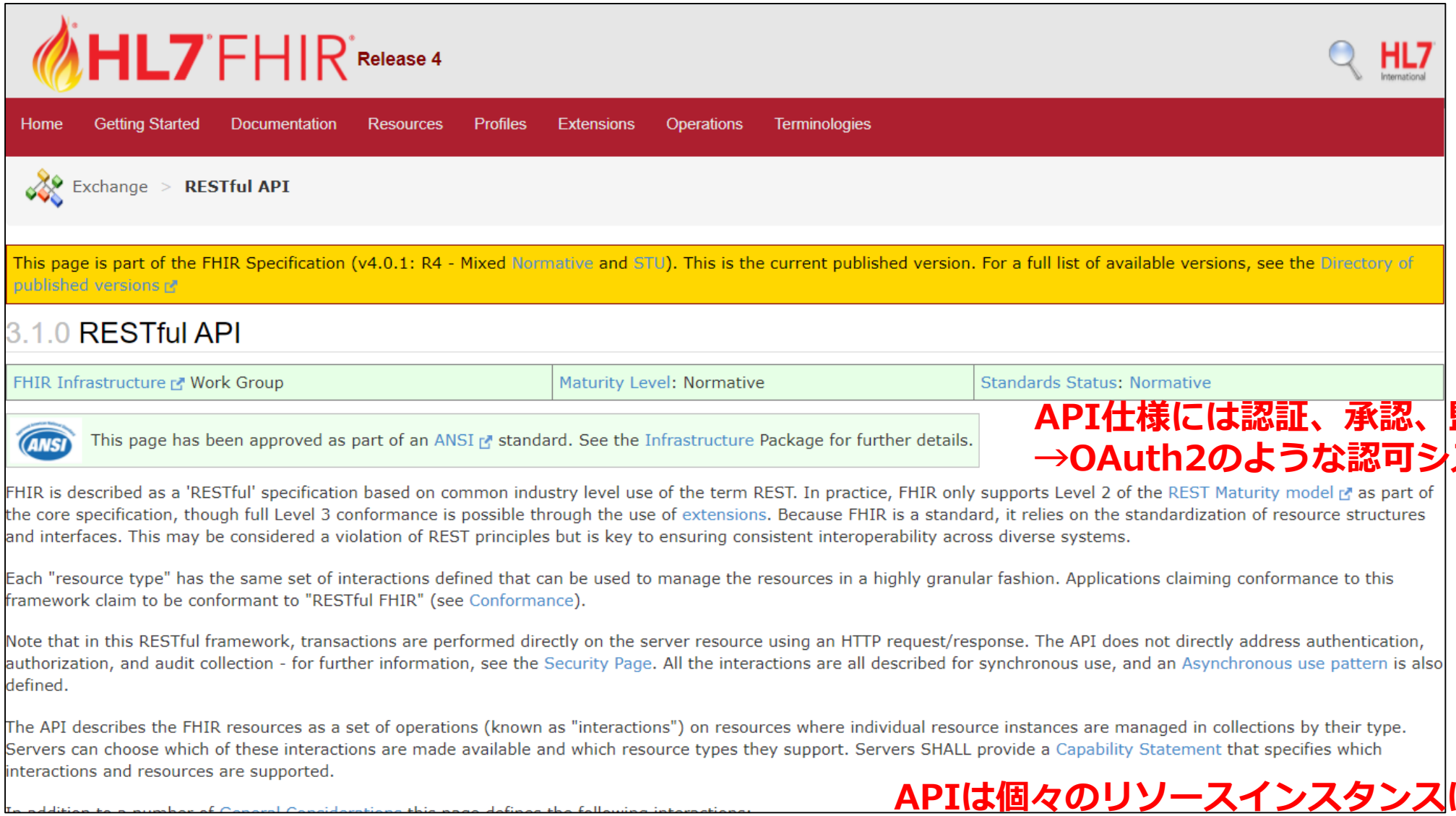


実践例：HTML/JavaScript

今日理解する内容



FHIRのRESTful APIの仕様とは何か



HL7 FHIR Release 4

Home | Getting Started | Documentation | Resources | Profiles | Extensions | Operations | Terminologies

Exchange > **RESTful API**

This page is part of the FHIR Specification (v4.0.1: R4 - Mixed Normative and STU). This is the current published version. For a full list of available versions, see the Directory of published versions

3.1.0 RESTful API

FHIR Infrastructure Work Group	Maturity Level: Normative	Standards Status: Normative

This page has been approved as part of an ANSI standard. See the Infrastructure Package for further details.

FHIR is described as a 'RESTful' specification based on common industry level use of the term REST. In practice, FHIR only supports Level 2 of the REST Maturity model as part of the core specification, though full Level 3 conformance is possible through the use of extensions. Because FHIR is a standard, it relies on the standardization of resource structures and interfaces. This may be considered a violation of REST principles but is key to ensuring consistent interoperability across diverse systems.

Each "resource type" has the same set of interactions defined that can be used to manage the resources in a highly granular fashion. Applications claiming conformance to this framework claim to be conformant to "RESTful FHIR" (see Conformance).

Note that in this RESTful framework, transactions are performed directly on the server resource using an HTTP request/response. The API does not directly address authentication, authorization, and audit collection - for further information, see the Security Page. All the interactions are all described for synchronous use, and an Asynchronous use pattern is also defined.

The API describes the FHIR resources as a set of operations (known as "interactions") on resources where individual resource instances are managed in collections by their type. Servers can choose which of these interactions are made available and which resource types they support. Servers SHALL provide a Capability Statement that specifies which interactions and resources are supported.

In addition to a number of General Considerations this page defines the following interactions:

API仕様には認証、承認、監査を含まない
→OAuth2のような認可システムで対応する

APIは個々のリソースインスタンスに対して、そのタイプごとに操作（interaction）を行う

FHIRのRESTful APIの仕様

- RESTful APIでは下記の操作が定義されている

In addition to a number of General Considerations this page defines the following interactions:

Instance Level Interactions	
read	Read the current state of the resource
vread	Read the state of a specific version of the resource
update	Update an existing resource by its id (or create it if it is new)
patch	Update an existing resource by posting a set of changes to it
delete	Delete a resource
history	Retrieve the change history for a particular resource
Type Level Interactions	
create	Create a new resource with a server assigned id
search	Search the resource type based on some filter criteria
history	Retrieve the change history for a particular resource type
Whole System Interactions	
capabilities	Get a capability statement for the system
batch/transaction	Update, create or delete a set of resources in a single interaction
history	Retrieve the change history for all resources
search	Search across all resource types based on some filter criteria

CRUD操作のREADと同じ

特定のバージョンのリソースをREAD

idを指定して更新、CRUD操作のUPDATE

まとめて更新

CRUD操作のDELATE

変更履歴を取得

CRUD操作のCREATE

リソースタイプの検索

特定のリソースタイプの変更履歴

FHIRにおけるCRUD形式の説明

FHIR REST APIでのCRUD操作(read)

最も多用される要素

```
GET [base]/[type]/[id] {?_format=[mime-type]}
```

3.1.1.5.9 Elements

```
GET [base]/Patient?_elements=identifier,active,link
```

3.1.0.2 read

The `read` interaction accesses the current contents of a resource. The interaction is performed by an HTTP `GET` command as shown:

```
GET [base]/[type]/[id] {?_format=[mime-type]}
```

This returns a single instance with the content specified for the resource type. This url may be accessed by a browser. The possible values for the Logical Id ("id") itself are described in the id type. The returned resource SHALL have an `id` element with a value that is the `[id]`. Servers SHOULD return an `ETag` header with the versionId of the resource (if versioning is supported) and a `Last-Modified` header.

Note: Unknown resources and deleted resources are treated differently on a read: a `GET` for a deleted resource returns a `410` status code, whereas a `GET` for an unknown resource returns `404`. Systems that do not track deleted records will treat deleted records as an unknown resource. Since deleted resources may be brought back to life, servers MAY include an ETag on the error response when reading a deleted record to allow version contention management when a resource is brought back to life.

In addition, the search parameter `_summary` can be used when reading a resource:

```
GET [base]/[type]/[id] {?_summary=text}
```

This requests that only a subset of the resource content be returned, as specified in the `_summary` parameter, which can have the values `true`, `false`, `text`, `count` and `data`. Note that a resource that only contains a subset of the data is not suitable for use as a base to update the resource, and might not be suitable for other uses. The same applies to the _elements parameter - both that it should be supported, and the subset implications. Servers SHOULD define a `Resource.meta.tag` with the SUBSETTED as a Simple Tag to explicitly mark such resources.

A HEAD request can also be used - see below.

FHIR REST APIでのCRUD操作(vread)

```
GET [base]/[type]/[id] {?_format=[mime-type]}
```

通信の変更に伴うバージョン変更に対応するもの

logical Idのフォーマットに従う

バージョンが存在しない場合コード410が戻る

データが存在しない場合コード404が戻る

3.1.0.3 vread

The `vread` interaction performs a version specific read of the resource. The interaction is performed by an HTTP `GET` command as shown:

```
GET [base]/[type]/[id]/_history/[vid] {?_format=[mime-type]}
```

This returns a single instance with the content specified for the resource type for that version of the resource. The returned resource SHALL have an `id` element with a value that is the `[id]`, and a `meta.versionId` element with a value of `[vid]`. Servers SHOULD return an `ETag` header with the versionId (if versioning is supported) and a `Last-Modified` header.

The Version Id ("vid") is an opaque identifier that conforms to the same format requirements as a Logical Id. The id may have been found by performing a history interaction (see below), by recording the version id from a content location returned from a `read` or from a version specific reference in a content model. If the version referred to is actually one where the resource was deleted, the server should return a `410` status code.

Servers are encouraged to support a version specific retrieval of the current version of the resource even if they do not provide access to previous versions. If a request is made for a previous version of a resource, and the server does not support accessing previous versions (either generally, or for this particular resource), it should return a `404` Not Found error, with an operation outcome explaining that history is not supported for the underlying resource type or instance.

A HEAD request can also be used - see below.

FHIR REST APIでのCRUD操作(update)

```
PUT [base]/[type]/[id] {?_format=[mime-type]}
```

既に登録されたIdに対して適切

Idが不適切である場合コード400が戻る

適切である場合、サーバは読まれたデータと同じものを返す

データを返す動作が適切でない場合もある

Idが適切である場合コード200が戻る

XMLコンテンツ、使用法、JSONでの空白についてサーバーの裁量

（データ改変になるため、デジタル署名の意味に影響を与える）

3.1.0.4 update

The `update` interaction creates a new current version for an existing resource or creates an initial version if no resource already exists for the given id. The `update` interaction is performed by an HTTP `PUT` command as shown:

```
PUT [base]/[type]/[id] {?_format=[mime-type]}
```

The request body SHALL be a Resource with an `id` element that has an identical value to the `[id]` in the URL. If no `id` element is provided, or the id disagrees with the id in the URL, the server SHALL respond with an HTTP `400` error code, and SHOULD provide an OperationOutcome identifying the issue. If the request body includes a `meta`, the server SHALL ignore the provided `versionId` and `lastUpdated` values. If the server supports versions, it SHALL populate the `meta.versionId` and `meta.lastUpdated` with the new correct values. Servers are allowed to review and alter the other metadata values, but SHOULD refrain from doing so (see metadata description for further information). Note that there is no support for updating past versions - see notes on the history interaction.

A server SHOULD accept the resource as submitted when it accepts the update, and return the same content when it is subsequently read. However systems might not be able to do this; see the note on transactional integrity for discussion. Also, see Variations between Submitted data and Retrieved data for additional discussion around update behavior. Note that `update` generally updates the whole content of the resource. For partial updates, see `patch` below.

If the interaction is successful, the server SHALL return either a `200` OK HTTP status code if the resource was updated, or a `201` Created status code if the resource was created (or brought back to life/re-created), with a `Last-Modified` header, and an `ETag` header which contains the new `versionId` of the resource. If the resource was created (i.e. the interaction resulted in a `201` Created), the server SHOULD return a `Location` header (this is for HTTP conformance; it's not otherwise needed). The body of response is as described in Managing Return Content.

Note: Servers MAY choose to preserve XML comments, instructions, and formatting or JSON whitespace when accepting updates, but are not required to do so. The impact of this on digital signatures may need to be considered.

FHIR REST APIでのCRUD操作(delete)

```
DELETE [base]/[type]/[id]
```

request.body部は空で

deleteされると、searchで引っかからなくなる

200は消去すべき内容がない場合

204はそもそも存在しない場合

202は在否を明らかにしたくない場合

3.1.0.7 delete

The `delete` interaction removes an existing resource. The interaction is performed by an HTTP `DELETE` command as shown:

```
DELETE [base]/[type]/[id]
```

The request body SHALL be empty.

A delete interaction means that subsequent non-version specific reads of a resource return a `410` HTTP status code and that the resource is no longer found through search interactions. Upon successful deletion, or if the resource does not exist at all, the server should return either a `200 OK` if the response contains a payload, or a `204 No Content` with no response payload, or a `202 Accepted` if the server wishes to be non-commital about the outcome of the delete.

Whether to support delete at all, or for a particular resource type or a particular instance is at the discretion of the server based on the policy and business rules that apply in its context. If the server refuses to delete resources of that type as a blanket policy, then it should return the `405` Method not allowed status code. If the server refuses to delete a resource because of reasons specific to that resource, such as referential integrity, it should return the `409` Conflict status code. Note that the servers MAY choose to enforce business rules regarding deletion of resources that are being referenced by other resources, but they also MAY NOT do so. Performing this interaction on a resource that is already deleted has no effect, and the server should return either a `200 OK` if the response contains a payload, or a `204 No Content` with no response payload. Resources that have been deleted may be "brought back to life" by a subsequent `update` interaction using an HTTP `PUT`.

FHIR REST APIでのCRUD操作(patch)

```
PATCH [base]/[type]/[id] {?_format=[mime-type]}
```

既に存在するリソースに「追加」のみを行う処理

例：ある診察記事に対して、補足説明を加える場合

送信側の負担が少なくて済む

patch操作はバージョン依存性が大きい（インターフェースプログラムの実装仕様によるため）

3.1.0.6 patch

As an alternative to updating an entire resource, clients can perform a patch operation. This can be useful when a client is seeking to minimize its bandwidth utilization, or in scenarios where a client has only partial access or support for a resource. The `patch` interaction is performed by an HTTP `PATCH` command as shown:

```
PATCH [base]/[type]/[id] {?_format=[mime-type]}
```

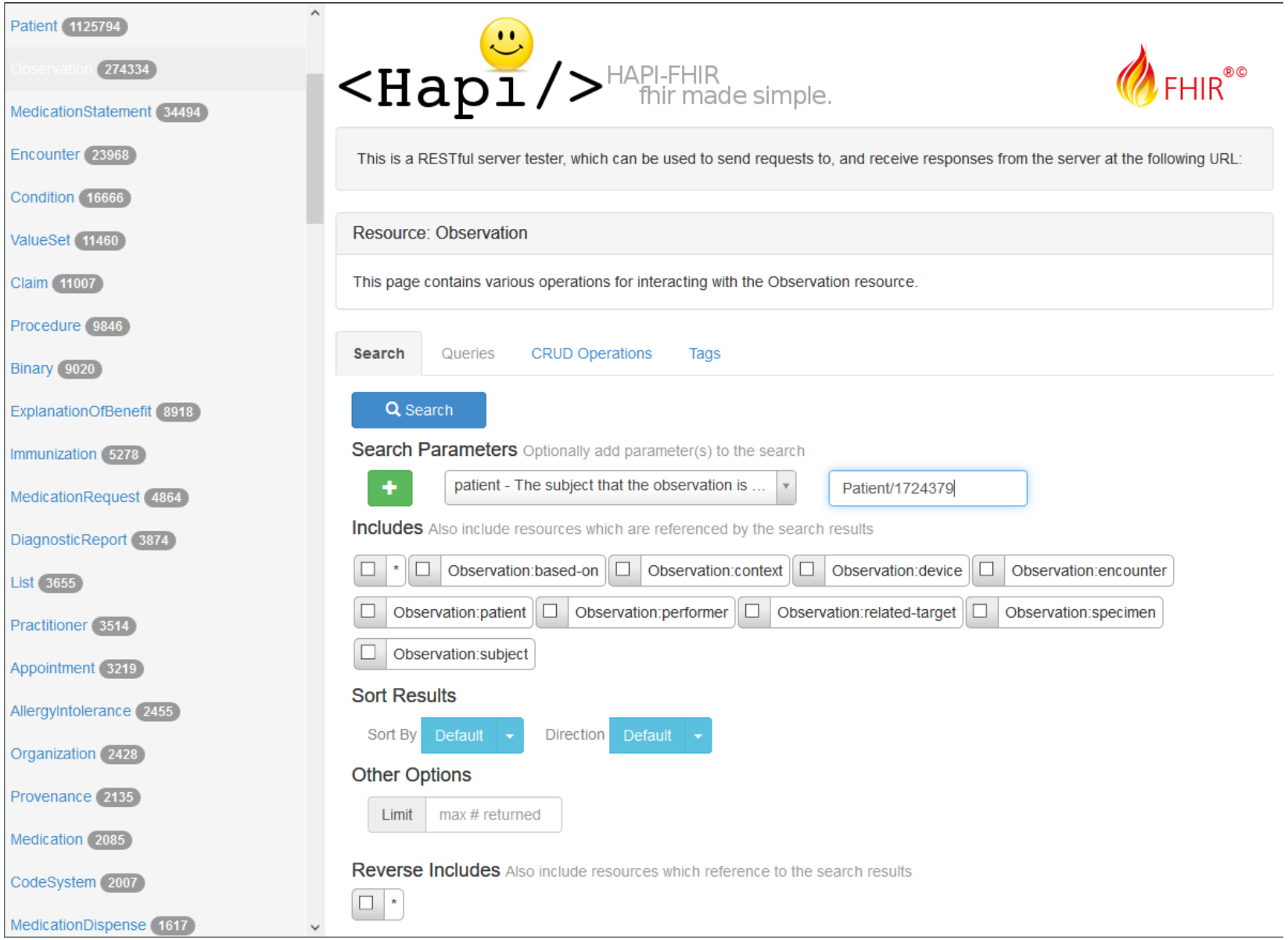
The body of a PATCH operation SHALL be either:

- a JSON Patch document with a content type of `application/json-patch+json`
- an XML Patch document with a content type of `application/xml-patch+xml`
- a FHIRPath Patch parameters resource with FHIR Content Type

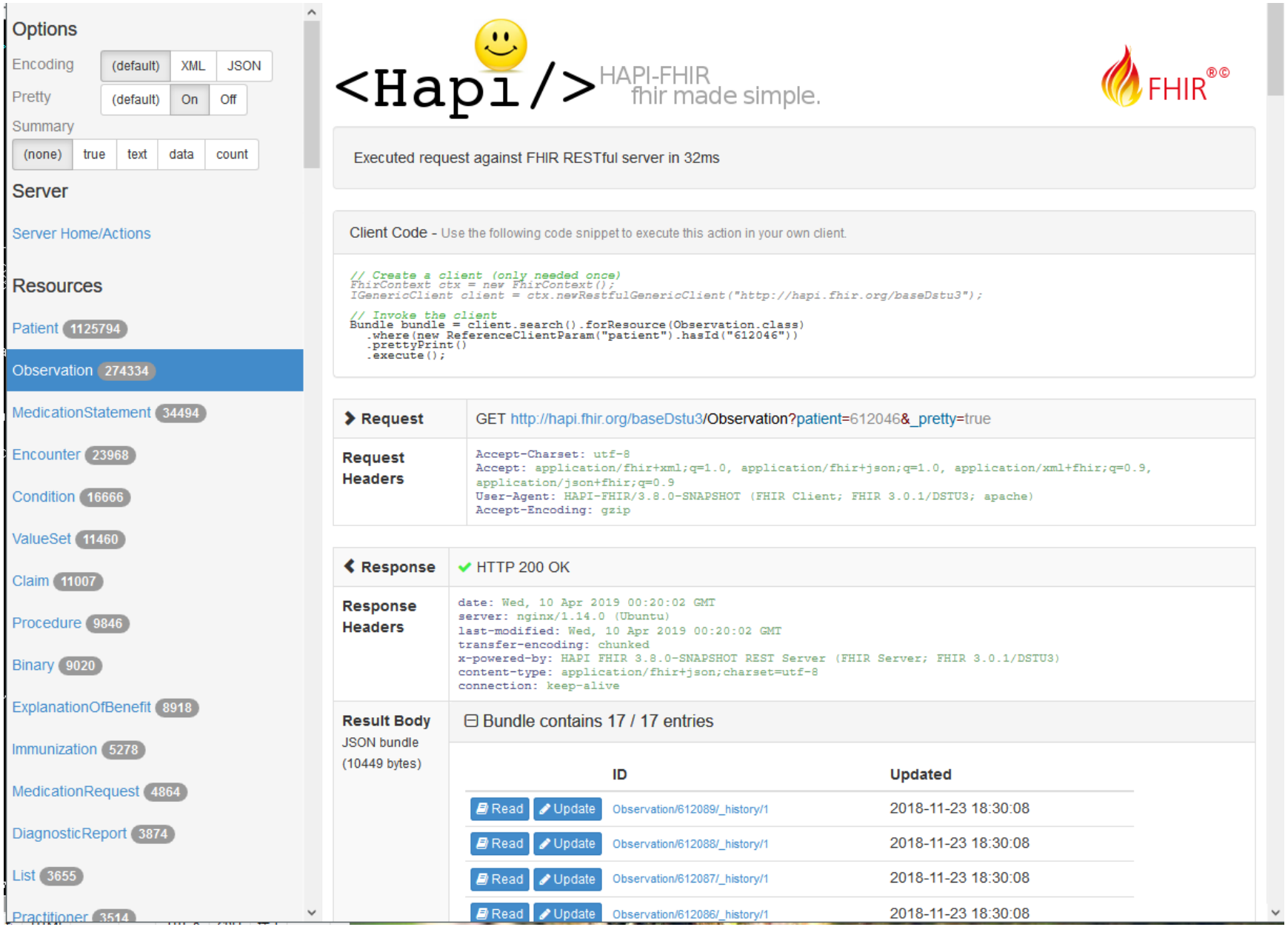
In either case, the server SHALL process its own copy of the resource in the format indicated, applying the operations specified in the document, following the relevant PATCH specification. When the operations have all been processed, the server processes the resulting document as an `Update` operation; all the version and error handling etc. apply as specified, as does the Prefer Header.

Processing PATCH operations may be very version sensitive. For this reason, servers SHALL support conditional PATCH, which works exactly the same as specified for update in Concurrency Management. Clients SHOULD always consider using version specific PATCH operations so that inappropriate actions are not executed.

FHIR REST APIでのCRUD操作



FHIR REST APIでのCRUD操作

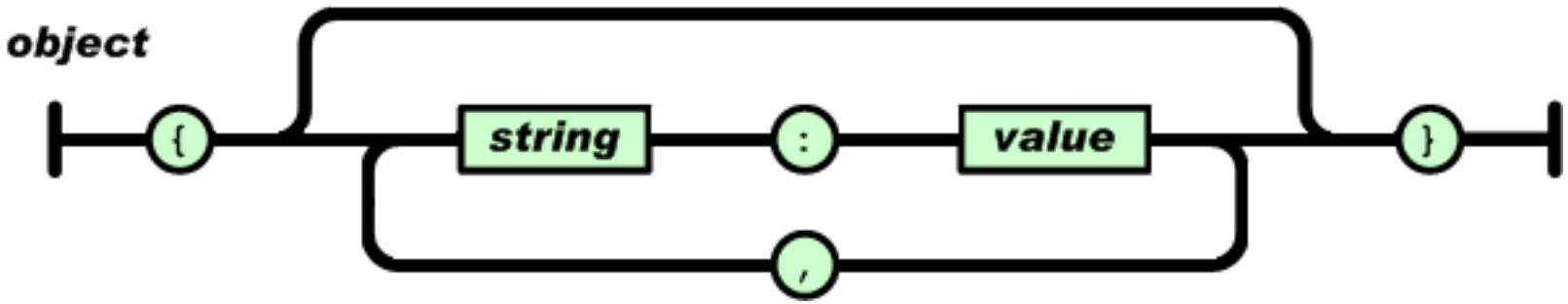


JSON形式、他プログラミング言語での実装例紹介

JSON（Web標準のデータ定義言語）

- JSON=JavaScript Object Notation
- JavaScriptの文法に沿っています
- 内容１：名前/値
- 内容２：配列

https://www.json.org/json-ja.html



例
{“食事”：“うな重”}

```
{ "name"    : "John Smith",
  "sku"     : "20223",
  "price"   : 23.95,
  "shipTo" : { "name" : "Jane Smith",
               "address" : "123 Maple Street",
               "city" : "Pretendville",
               "state" : "NY",
               "zip"    : "12345" },
  "billTo" : { "name" : "John Smith",
               "address" : "123 Maple Street",
               "city" : "Pretendville",
               "state" : "NY",
               "zip"    : "12345" }
}
```

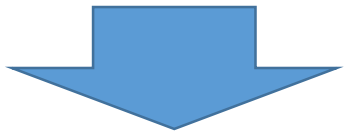
https://www.ibm.com/support/knowledgecenter/ja/SS9H2Y_7.5.0/com.ibm.dp.doc/json_jsonexamples.html

ざっくり概略

サーバに
RESTでアクセス

JSON形式の
FHIRデータを読み取る

サーバに
RESTで返す



ここはPythonでもVBAでもJavaScriptでも何でも良い

ソースコード解説

FHIRサーバに接続

**受信したデータの
JSONオブジェクト変換**

**JSONのラベルを読み取って
必要な項目を抜き出し**

**HTML表示指定箇所に
テキストを出力**

```
function onButtonClick() {

  var linktext5 = "http://hapi.fhir.org/baseDstu3/Observation?patient=1723865&_pretty=true"
  target = document.getElementById("output");

  var xhr = new XMLHttpRequest();
  xhr.responseType = 'text';
  xhr.open('GET',linktext5,true);
  xhr.send();

  let insertLabels = '';
  let insertDatas = '';
  let insertLabels2 = '';
  let insertDatas2 = '';

  xhr.onreadystatechange = function(){
        if (xhr.readyState==4 && xhr.status==200) {

  var data = JSON.parse(xhr.responseText);
  //要素の求め方 data[ entry ].length

  var extractdata;
  extractdata += '<br>'

        for (  var i in data.entry  ) {
              if (data.entry[i].resource.code.text == 'Heart Rate'){
                    insertLabels += "'" + data.entry[i].resource.effectiveDateTime + "',";
                    insertDatas += "'" +  data.entry[i].resource.valueQuantity.value + "',";
                    if(i < (data['entry'].length - 1)){
                          insertLabels += ",";
                          insertDatas += ",";
                    }
              }
              if (data.entry[i].resource.code.text == 'Respiratory Rate'){
                    insertLabels2 += "'" + data.entry[i].resource.effectiveDateTime + "',";
                    insertDatas2 += "'" +  data.entry[i].resource.valueQuantity.value + "',";
                    if(i < (data['entry'].length - 1)){
                          insertLabels2 += ",";
                          insertDatas2 += ",";
                    }
              }
              extractdata +=  data.entry[i].resource.id + '-' + data.entry[i].resource.code.text + "-" + data.entry[i].resource
        }

  var outtext =  '<br> patient_id =' + data.entry[0].resource.subject.reference + '<br> elements.No = ' + data['entry'].length +
  target.innerHTML = outtext;

  plotres(data, '');
  var insertLabel;
  var insertData;
  var insertLabel2;
  var insertData2;
```

ソースコード解説

グラフ用のフォーマット成型

"labels: [" +"'1', '2', '3', '4', '5', '6', '7'"+ "],"

'data: [12, 19, 3, 17, 6, 3, 7],'

グラフ表示のコード部に出力

グラフ表示のコード部

```
        plotres(data, '');
        var insertLabel;
        var insertData;
        var insertLabel2;
        var insertData2;

        //日付
        insertLabel = "labels: [" + insertLabels + "],";
        //内容
        insertData = 'data: [' + insertDatas + '],';

        //日付
        insertLabel2 = "labels: [" + insertLabels2 + "],";
        //内容
        insertData2 = 'data: [' + insertDatas2 + '],';

        console.log(insertLabels);
        console.log(insertDatas);

        onShowChart2(insertLabel, insertData);
        onShowChart3(insertLabel2, insertData2);

            }
    };

    }


    function onShowChart2(insertLabel, insertData){

        var chartheader = document.createElement("script");
        chartheader.innerHTML = "var ctx = document.getElementById('myChart').getContext('2d');" +
                                'var myChart = new Chart(ctx, {' +
                                "type: 'bar'," +
                                'data: {' +
                               //このラベル領域を差し替える
                                insertLabel +
                                "datasets: [{" +
                                "label: 'patient Heart Rate'," +
                               //このデータ領域を差し替える
                                insertData +
                                'backgroundColor: "rgba(153,255,51,0.4)"' +
                                '}]' +
                                '}' +
                                '});';
        target = document.getElementById("outchartscript");
        target.appendChild(chartheader);
    }
```

（1）VBAでREST

- VBAでもREST APIを利用できる関数が標準に

REST API を呼び出す方法

Excel 2013 から新たに WEBSERVICE 関数が標準関数として登場しました。
引数に URL を指定するだけで、XML や JSON といった形式問わず、結果文字列がセルに値として出力されます。

```
=WEBSERVICE(URL)
```

Excel 2010 以前のバージョンについては、MSXML2.XMLHTTP オブジェクトを利用したユーザ定義関数を定義します。
（ユーザ定義関数は、EXCEL マクロで VBA の Function として実装します）

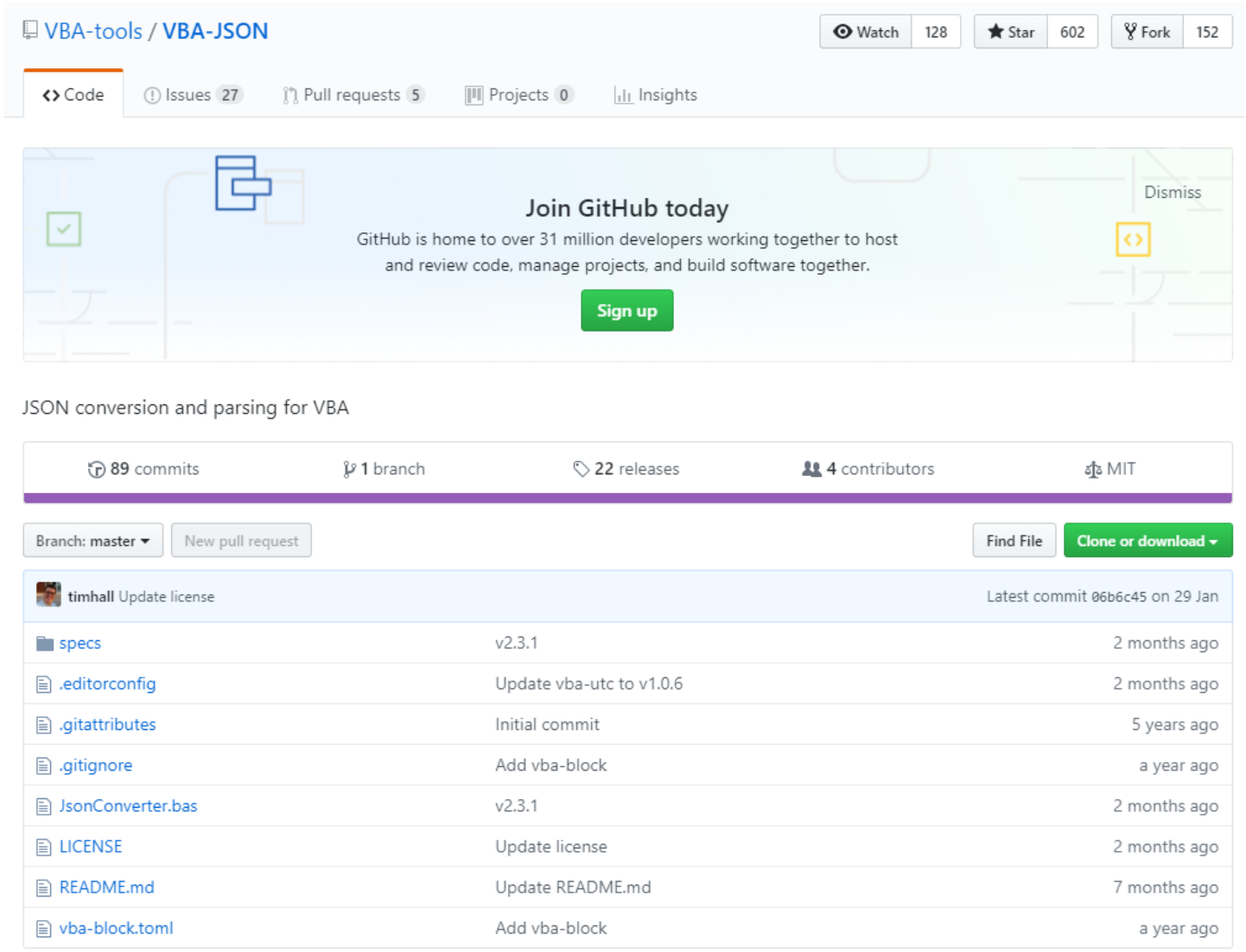
```
Function RESTAPI(URL As String)
   Dim objXMLHttp As Object, zipArr
   Set objXMLHttp = CreateObject("MSXML2.XMLHTTP")
       objXMLHttp.Open "GET", URL, False
       objXMLHttp.Send

   RESTAPI = objXMLHttp.responseText
End Function
```

WEBSERVICE関数を活用

※環境によって動かない場合は CMD or ターミナルを呼び出すことでも可能

VBAでJSON（JsonConverter）

- https://github.com/VBA-tools/VBA-JSON
- パースもできます（！）



Drop-in replacement for Scripting.Dictionary on Mac

45 commits | 1 branch | 7 releases | 2 contributors | MIT

Branch: master | New pull request | Find File | Clone or download

timhall Update README.md — Latest commit c157fcf on 31 Aug 2018

specs	v1.4.1	3 years ago
.gitattributes	Initial commit	5 years ago
.gitignore	Initial vba-block	a year ago
Dictionary.cls	v1.4.1	3 years ago
LICENSE	Initial vba-block	a year ago
README.md	Update README.md	7 months ago
vba-block.toml	Initial vba-block	a year ago

README.md

VBA-Dictionary

VBA-Dictionary

ハンズオンの実例

Excel VBA

restout.txt - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

{ "resourceType": "Patient", "id": "1579762", "meta": { "versionId": "1", "lastUpdated": "2019-03-07T01:55:12.813+00:00" }, "text": { "status": "generated", "div": "<div xmlns=¥"http://www.w3.org/1999/xhtml¥">Mary Moore " }, "name": [{ "use": "official", "text": "Mary Moore", "family": "Moore", "given": ["Mary"] }], "gender": "female", "birthDate": "1953-06-24" }

nection.xlsm - [Module1 (コード)]

デバッグ(D) 実行(R) ツール(T) アドイン(A) ウィンドウ(W) ヘルプ(H)

10 行, 89 桁

CreateJsonObject

```
stConnection()

.Cells(1, 1) = Application.WorksheetFunction.WebService("http://hapi.fhir.org/baseDstu3/Patient

b

eateJsonObject()

Dim str As String
str = Application.WorksheetFunction.WebService("http://hapi.fhir.org/baseDstu3/Patient/1579762/_histo

Open "C:¥data¥restout.txt" For Output As #1
 Print #1, str
Close #1

Dim jsonObj As Object
Set jsonObj = JsonConverter.ParseJson(str)

Sheet1.Cells(1, 2) = jsonObj("id")

End Sub
```

JsonConverter
Module1
クラス モジュール
Dictionary

プロパティ - Module1

Module1 Module

全体 | 項目別

(オブジェクト名) Module1

Excel 出力画面

RESTConnection.xlsm - Excel

ファイル ホーム 挿入 ページ レイアウト 数式 データ 校閲 表示 開発 アドイン アンテナハウス 操作アシ サインイン 共有

A3

	A	B	C	D	E
	{				
	"resourceType": "Bundle",				
	"id": "641ba2a5-1950-4c90-998c-f0a694aca8d2",				
	"meta": {	1579762			
	"lastUpdated": "2019-03-29T02:05:57.048+00:00"				
	},				
1	"type": "searchset"				
2		1953/6/24			
3					
4					
5					

Sheet1

準備完了 100%

実際のFHIRサーバからREST形式でJSONデータを取得し、Id, birthDateを抽出できました

MSXML2.XMLHTTP.6.0を用いた呼び出し

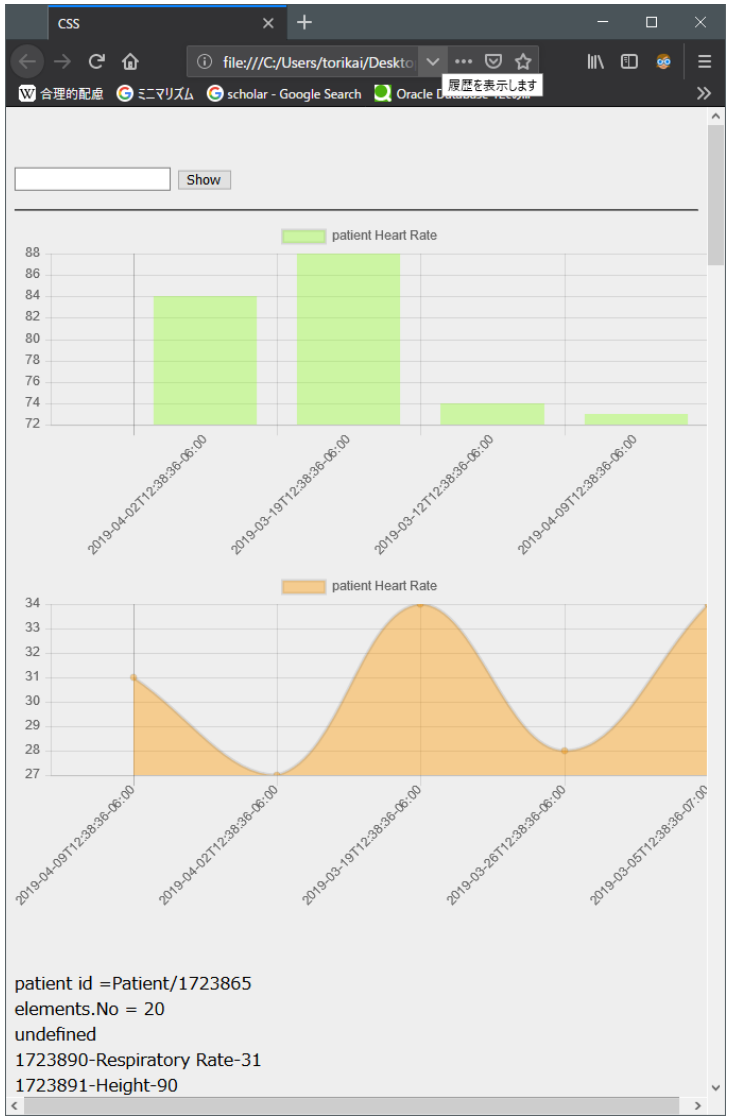
単純なGETは以下のようなコードで簡単に呼び出すことが可能です。

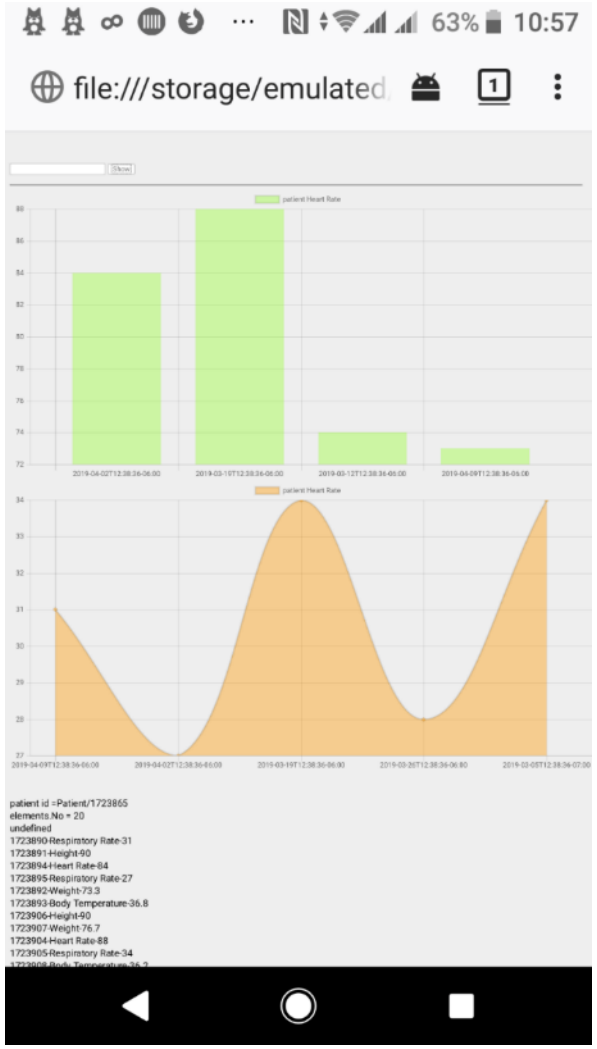
http.vba

```
Dim objHTTP As Object
Set objHTTP = CreateObject("MSXML2.XMLHTTP.6.0")
objHTTP.Open "GET", "http://test.com/index.htm, False
objHTTP.setRequestHeader "Authorization", "Basic EncodedUserNameAndPassword"
objHTTP.setRequestHeader "Content-Type", "text/plain"
objHTTP.send
```

SETまで使い、途中のHeaderまで制御しようと思えば、この形式のほうが明示的で扱いやすいかも

Webの利点：PCでもスマートフォンでも動作する

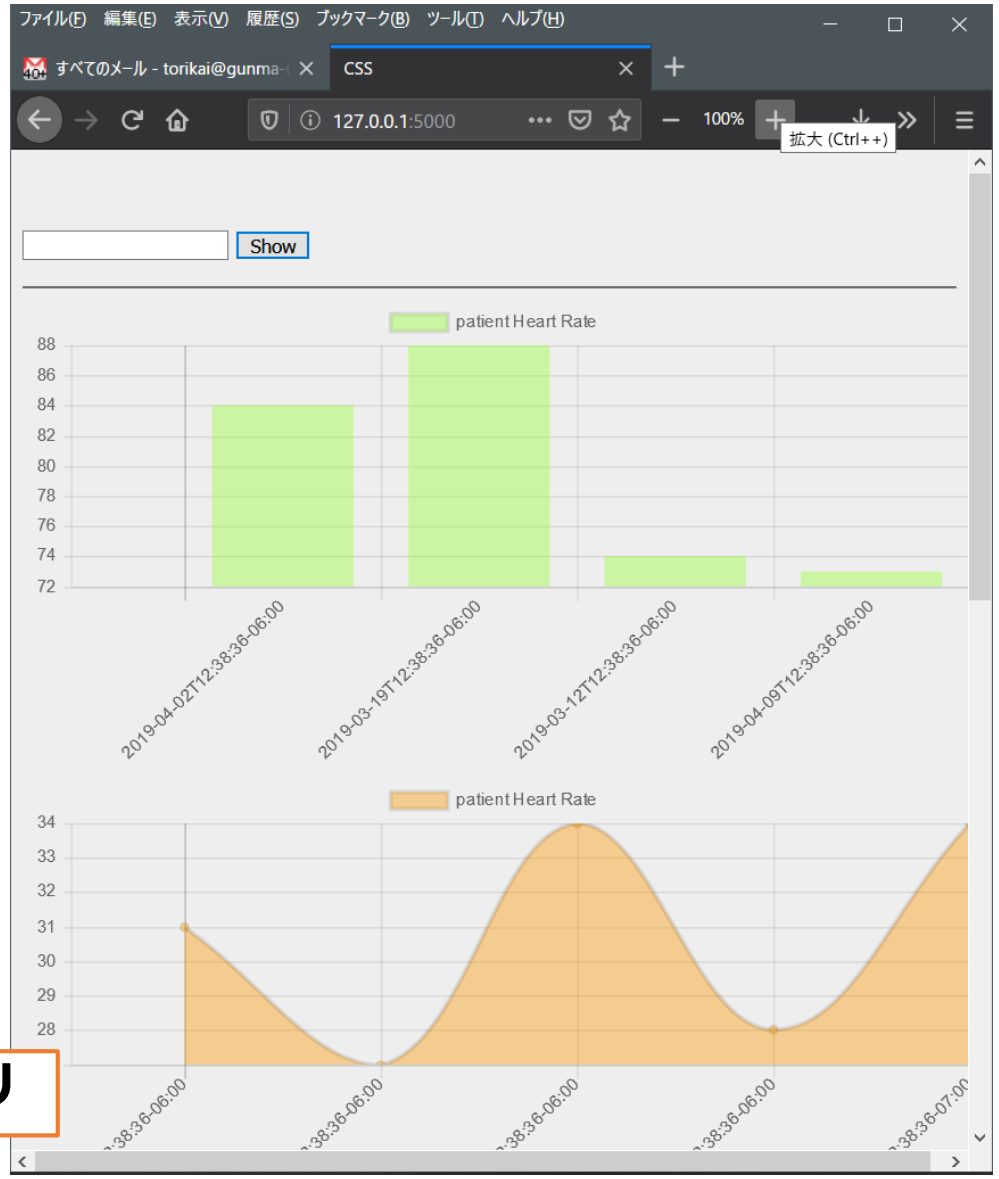




活用例：様々な帳票やダッシュボードの作成

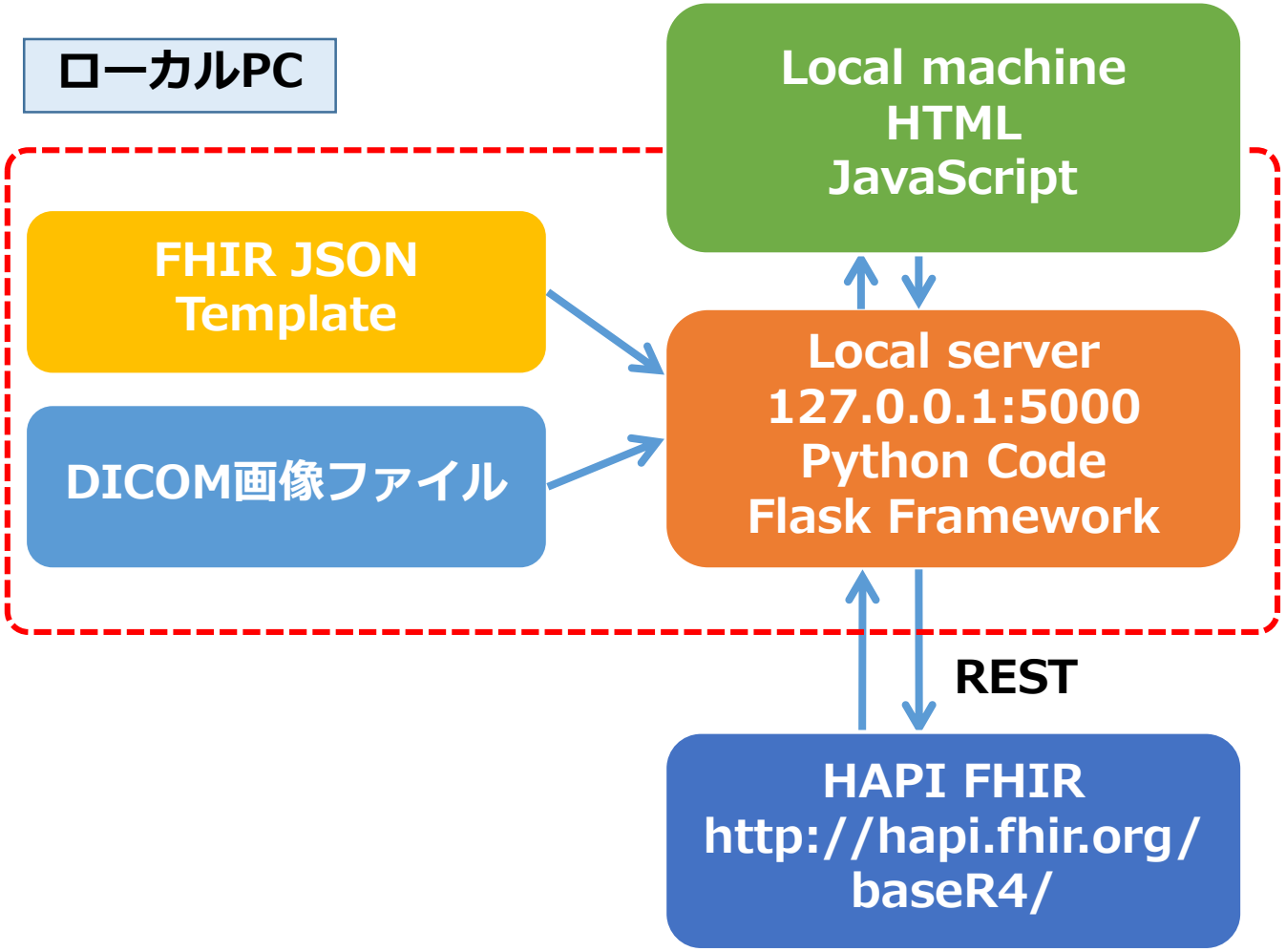
データをFHIRリソースで表現できれば、Web標準技術で可能な様々なアプリや、様式なども自由に作成できる

個別に必要なものをDIYで作成するのに適している



WEBで表示できる簡易検査結果モニタアプリ

（3）PythonでREST：DICOMをFHIR形式に



Python/Flaskが中心となり、

1）FHIRサーバとのREST通信
2）DICOMファイルの読み取り
3）FHIRテンプレートに書込み

ImagingStudyリソースの作成とHTMLへの送信プログラムを作成します。
（表示系は前回チュートリアルのJavaScriptコーディングで）

PythonにおけるREST呼び出しの最も単純な形

サンプルコード

hapifhirtest0.py - TeraPad

ファイル(F) 編集(E) 検索(S) 表示(V) ウィンドウ(W) ツール(T) ヘルプ(H)

```
import pprint
import requests
import matplotlib.pyplot as plt
import math
import numpy as np
import json

def main():

    response = requests.get(

        'http://hapi.fhir.org/baseDstu3/Observation',

        params={'patient': '1723865'})


    jread = response.json()

    jdata = json.loads(json.dumps(jread))

    pprint.pprint(json.dumps(jread))

if __name__== '__main__':

    main()
[EOF]
```

RESTのrequest処理

JSON処理に必須

FHIRリソースのURL

RESTのパラメータ入力方法

JSON文字列に変換

JSON文字列としてコンソール出力

DICOMデータを埋め込みリソースを作成

FHIRテンプレートにDICOMデータを埋め込みImagingStudyリソースを作成

エディタにapp3.pyを表示

```
#Read dicom tag from raw file by PyDICOM
d = pydicom.read_file('CT_LEE_IR6a.dcm')
d2 = d.pixel_array
os.makedirs('static', exist_ok=True)
plt.imshow(d2, cmap=plt.cm.bone)
plt.savefig('static/dicom.jpg')
pil_img = Image.fromarray(d.pixel_array)
pil_img.save('static/figure1.png')
```

```
#File Template Read
path_w = './FHIRImagingStudyTemplate2.json'
with open(path_w, mode='r') as f:
    s = f.read()
    #print(s)
    jimg = json.loads(s)
    #pprint.pprint(jimg)
```

```
#Mapping for R! of ImagingStudy
jimg['text']['status'] = "registered"
jimg['text']['div'] = "" #<--追加
jimg['subject'] = "subject title"
jimg['series'][0]['uid'] = ""
jimg['series'][0]['modality'] = d.Modality
jimg['series'][0]['actor'] = d[0x0008, 0x0090].value
jimg['series'][0]['instance'][0]['uid'] = d.SOPInstanceUID
```

DICOMデータの読み込み

```
#Read dicom tag from raw file by PyDICOM
d = pydicom.read_file('CT_LEE_IR6a.dcm')
```

ライブラリ「pydicom」により、DICOMデータを読み取り、オブジェクト的に扱うことができる

テンプレートの読み込み

```
jimg = json.loads(s)
```

ファイルのデータを読み取りjsonとして取り扱うことができる

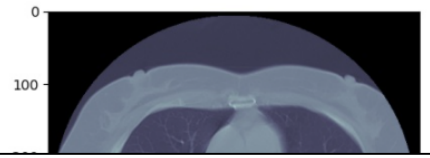
テンプレートのJSONにDICOMデータを埋め込み

```
jimg['series'][0]['modality'] = d.Modality
```

pythonプログラムの中でのJSONデータの取り扱い方の例。詳細は次ページ

DICOMデータを埋め込みリソースを作成

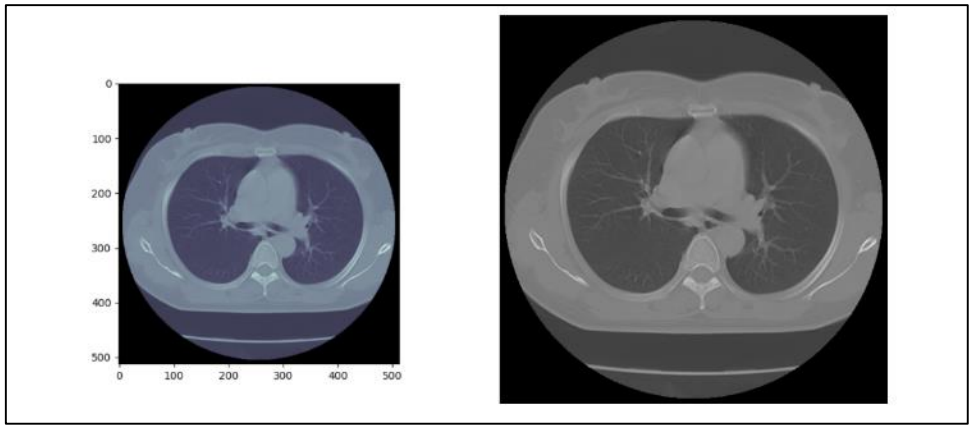

```
{
  "resourceType": "ImagingStudy",
  "id": "example",
  "text": {
    "status": "registered",
    "div": ""
  },
  "identifier": [
    {
      "system": "urn:dicom:uid",
      "value": "urn:oid:2.16.124.113543.6003.1154777499.30246.19789.3503430045"
    }
  ],
  "status": "available",
  "subject": "subject title",
  "started": "",
  "numberOfSeries": 1,
  "numberOfInstances": 1,
  "series": [
    {
      "uid": "",
      "number": 3,
      "modality": "CT",
      "description": "CT Surview 180",
      "numberOfInstances": 1,
      "bodySite": {
        "system": "http://snomed.info/sct",
        "code": "67734004",
        "display": "Upper Trunk Structure"
      },
      "instance": [
        {
          "uid": "1.2.392.200036.8120.100.20101001.1175630.2001002001",
          "sopClass": "1.2.840.10008.5.1.4.1.1.2",
          "number": 1
        }
      ],
      "actor": "SHIROGANE^HIDEO"
    }
  ]
}
```

PythonでのJSONデータの作成方法がわかれば、複数のソースからデータを取り込みFHIRリソースを作成することもできる

例）

- 画像データと…DICOMから
- 患者のアレルギー情報…HISから
- 他病院の検査歴…別のFHIRサーバから



ユーザー/ディベロッパーの立場からみるFHIR

個人的には、接続様式が固定的なバックエンド（サーバ間通信や保存/呼び出し）の利点よりも、活用バリエーションが大きいフロントエンド（オンデマンド、インタラクティブな、スマートフォン運用にも適合する末端ビューの作成）でFHIRを採用する利点が大きい

JSON/RESTの利用とともに、HTML/JavaScriptとの親和性が高く、REST構文での検索パラメータ指定によって、求める情報を収集できる

Web技術はPC、デバイスともに表示サポートを行っているため、デバイスハードウェアの変更によってソフトウェアが動かなくなるリスクを少なくでき、新規デバイスの利用までの時間短縮が期待される

HL7の実装と同様に、FHIRのうち、「データの使い方が決まっているフロントエンド＝末端」から部分的に実装していく（たとえば、処方や検査など）ことで、サーバサイドのシステム環境に大きな影響を与えずに現場での活用シーンを効果的に増加できると期待される

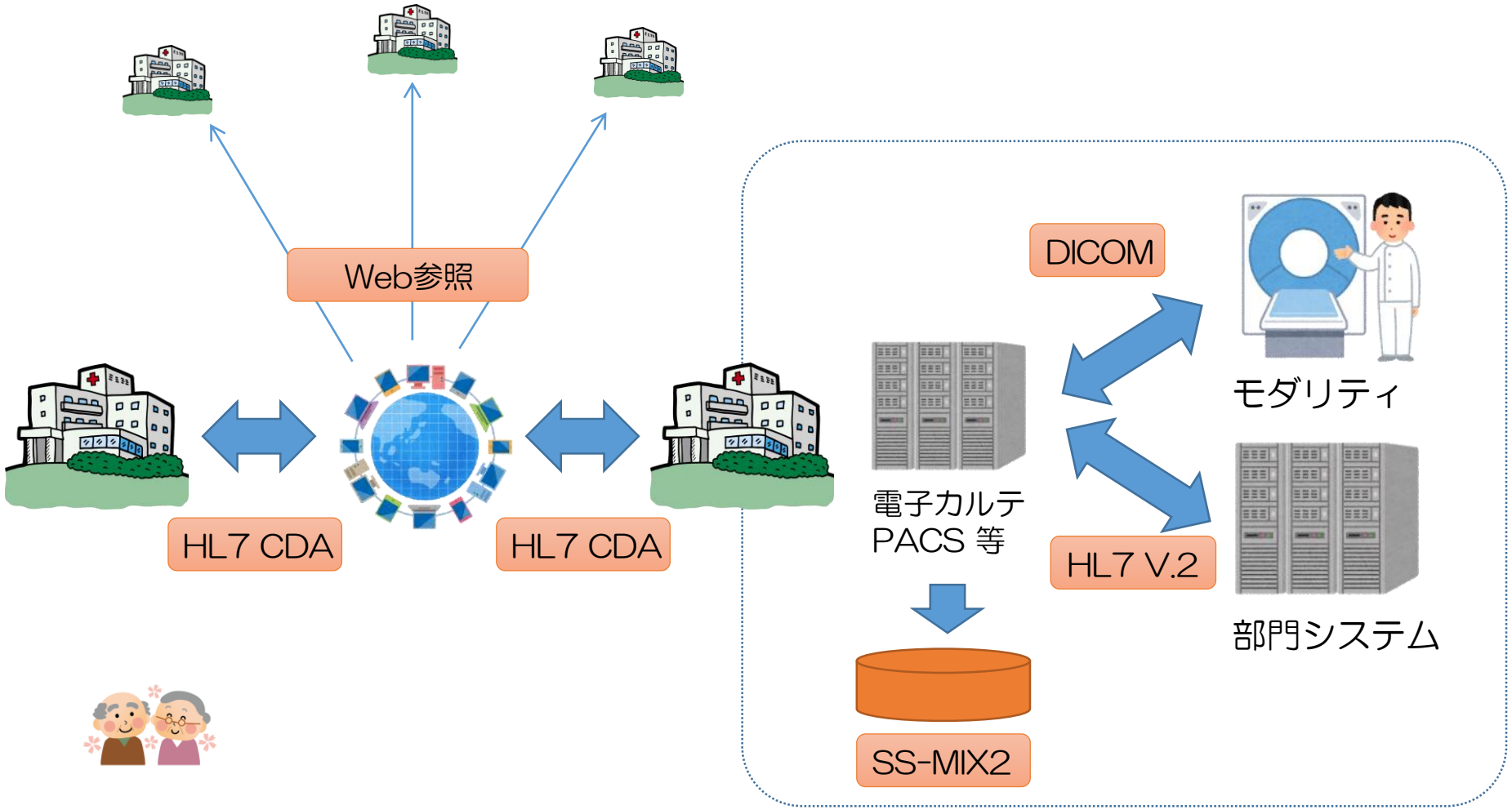
バックエンドではinteroperabilityの確保が死活問題（第２のガラパゴスを防ぐため）であり、現在導入されたHL7での接続を最大限活用し、また共通プロトコルの策定も同時に進める必要がある

今後の医療情報利活用環境に対する期待

私が提案する開発指針

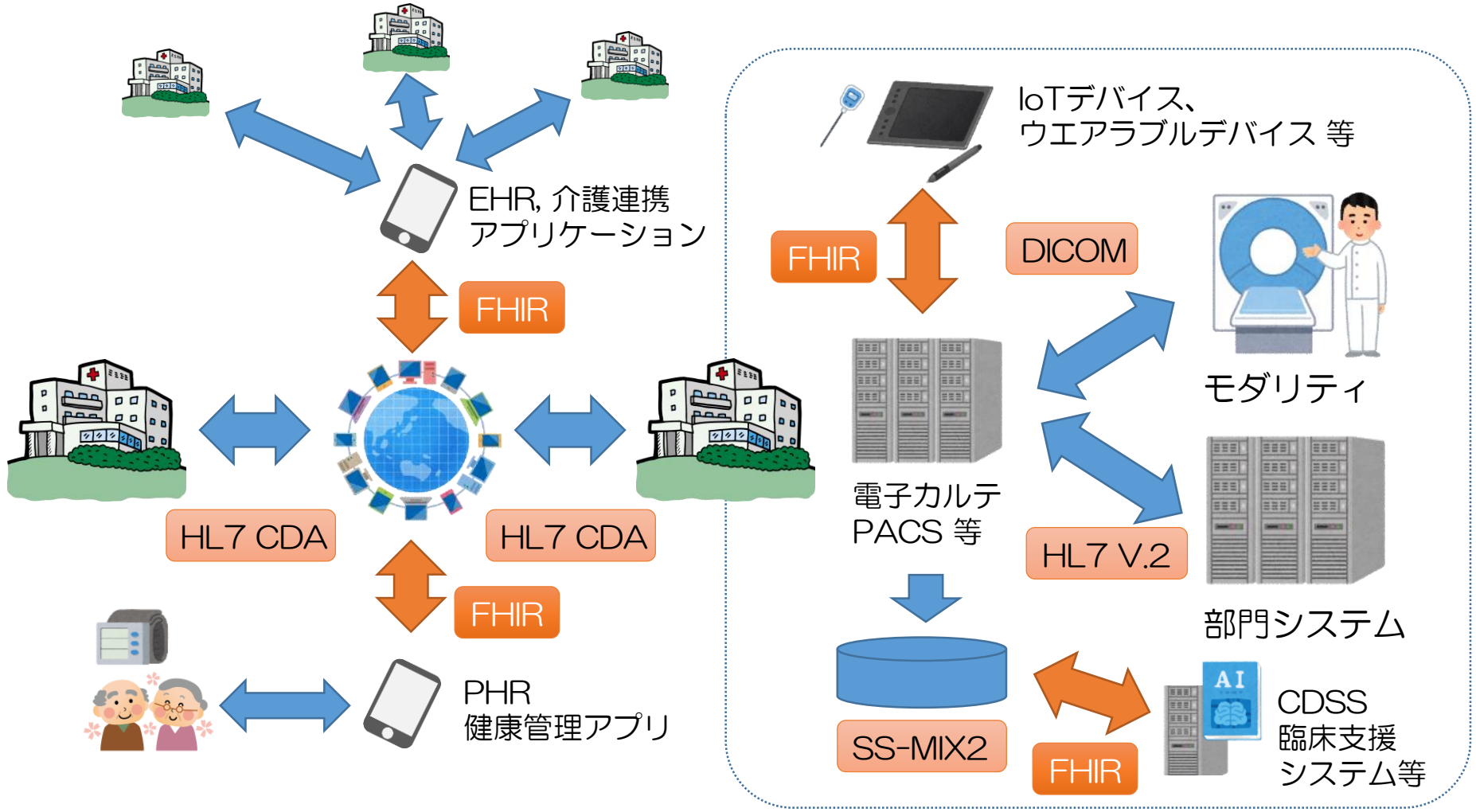
- **“Clean code that works” –Kent Beck, 「テスト駆動開発入門」**
  できるだけ、コメントなしで、短時間で理解できるコードを書こう
- **「あらゆるデータはn項のリレーションで表現される」-Frank Codd, RDBMS、**
  しかしこの言葉は「データ側の変容コスト」まで包括しない
- **KISS-”Keep It Simple, Stupid”できるだけ実体をシンプルに保とうとすることの重要性**
- **実装開始前に、ゴールとなる「なぜ」の答えをできるだけ包括的かつ普遍的な抽象度にまで引き上げる努力をすること**
- **アクションする際に考慮すべき4つの境界条件：**
  どれを買い、どこまでを覚えて使い、どこからを考えて作り、どの時点から人に委ねるか

現在の医療情報システム



- **HL7 v2.5をはじめとした標準規格に対応できる「開発力のある」ベンダや「規模や資金のある」施設が利用**

今後のシステム構築の形（REST/FHIRで広がる通信）



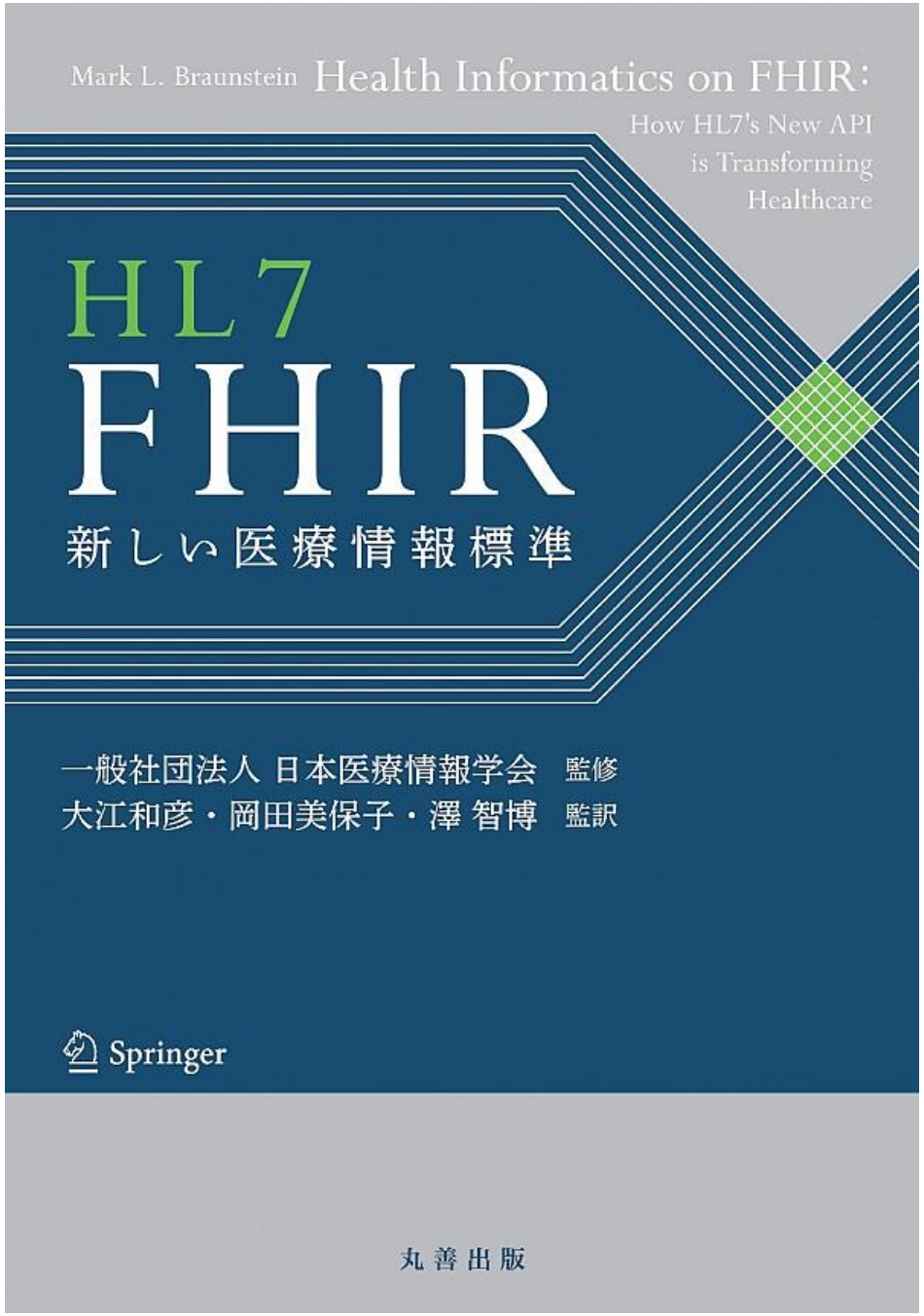
- **現在の標準規格を利用しながら、これまで連携が難しかったデバイスや利用者との接続を補完し、さらにより広いベンダの参入を可能に**

ただし、FHIRは（今のところ）「完全な」規格ではない

- 発展途上であること
  - 現在のR4版からは正式版となっているが、リソースが「Normative」（＝確定版）となったものは**Patient（患者基本情報）とObservation（検査）**のみ
- 「自由度の高さ」は管理されなければ「自由奔放」になりかねない
  - **80%のシステムで実際に使われるであろう要素を収載（「80%ルール」と呼ばれる）**
  - 利用者が自由に構築できる以上、それ以外の要素が自由に拡張されかねない

日本医療情報学会では、昨年度「HL7 FHIR実装検討WG（NeXEHRS研究会）」「FHIR研究会」など複数の課題研究会が立ち上がり、日本におけるリソースのあり方や実装のユースケースなどの検討の場となることが期待されている

- Web標準技術であるが故に**（本日のセミナーの本筋はココ）**
  - **認証の管理やセキュリティ対策**はもちろん開発者に責任が生じる
  - 医療分野以外のベンダからの参入が期待できるとはいえ、医療情報の安全管理に関するガイドラインは最低限遵守されるよう、発注者側も管理しなければならない



一般社団法人　日本医療情報学会(監修)

大江　和彦(監修 | 翻訳)

岡田　美保子(監修 | 翻訳)

澤 智博(監修 | 翻訳)

ご清聴ありがとうございました。

Technology Association Japan

第49回日本Mテクノロジー学会大会

MTA2020 on 世界遺産@富岡製糸場

【会期】　2021年8月28日(金)～29日(土)
【大会長】群馬大学医学部附属病院
　　システム統合センター
　　副センター長・准教授　鳥飼幸太

FHIRのプログラミングに関するハンズオンセミナーを予定しています。（これまでも年3回実施していますのでHPをご覧下さい）