

Web技術

神奈川工科大学
情報学部 情報工学科
田中哲雄

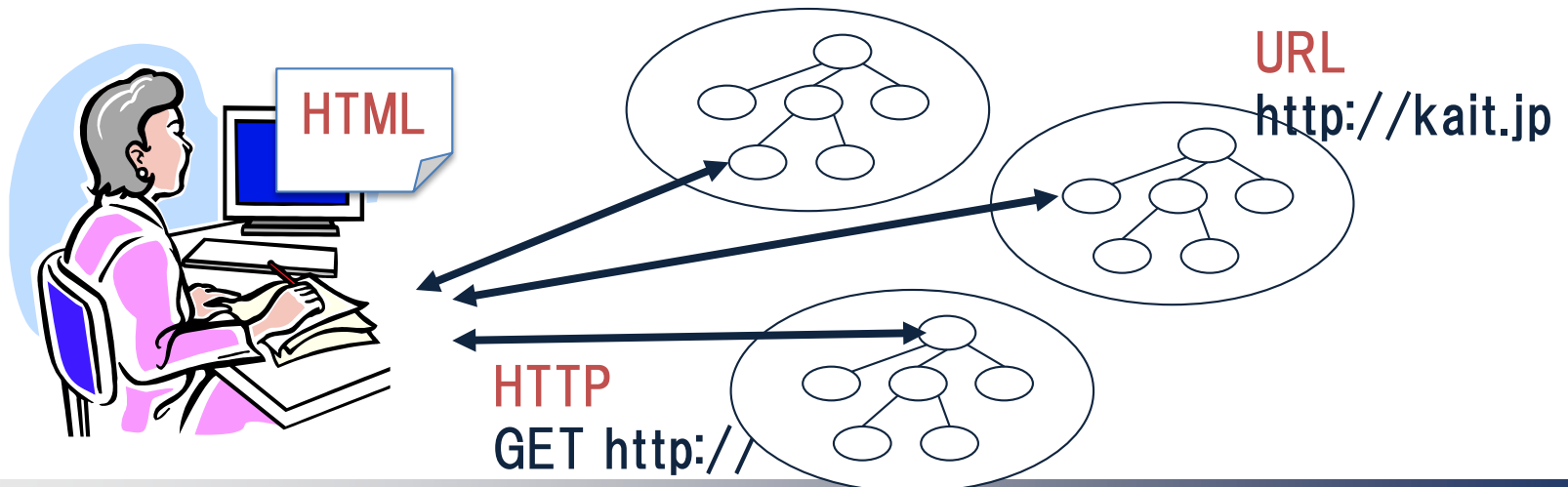
目次

1. Web技術の位置づけ
2. Webの基本技術
3. Webアプリケーション
4. Webアプリケーションの設計手順
5. Web API開発
6. Web技術のトピックス
7. クラウドコンピューティング
8. Ruby on Rails
9. ハンズオン Ruby on RailsでWebアプリ開発

- 
- 2.1 Webの3つの要素技術
 - 2.2 URL
 - 2.3 HTTP
 - 2.4 HTML
 - 2.5 REST

2.1 Webの3つの要素技術

- **URL** (Uniform Resource Locator)
 - 情報がどこにあるのか
- **HTTP** (Hyper Text Transfer Protocol)
 - 情報をどうやって手に入れるのか
- **HTML** (Hyper Text Markup Language)
 - 情報をどうやって表現するのか



2.2 URL ～ 情報がどこにあるのか ～

■ Uniform Resource Locator

- リソース(Web上の情報)を統一的に識別するID
- ブラウザなどのプログラムは, URLによりWeb上にある無数のリソースから, 一つのリソースを指定する

■ 例

- 横浜の天気
<http://weather.yahoo.co.jp/weather/jp/14/4610.html>
- 神奈川工科大学 情報工学科 ブログ記事の一つ
<http://blog.cs.kanagawa-it.ac.jp/2017/09/in-1.html>
- その記事の写真の一つ
http://4.bp.blogspot.com/-HiJ3xuV8Kg4/Wch95o-bNgl/AAAAAAAAAY3s/PauDDy07yXM-DSr5Br_74KQ8HnQwUV37QCK4BGAYYCw/s1600/8.jpg

2.2 URL (1)構成

http://blog.example.jp:8080/entries/show?id=123#more

(1) (2) (3) (4) (5) (6)

- (1) **スキーム** : URLが利用するプロトコル http, https, ftp, fileなど
- (2) **ホスト名** : ネットワーク上でホスト(サーバ)を一意に指し示す
ドメイン名, または, IPアドレス
- (3) **ポート番号** : 省略可, 省略するとプロトコルのデフォルト値
HTTPのデフォルト値は80
- (4) **パス** : ホスト中でリソースを一意に指し示す
ホスト名との組み合わせにより**リソース**を一意に識別
- (5) **クエリ** : 省略可, クライアントで動的にURLを生成する際に利用
検索サービスに検索キーワードを渡すなど
- (6) **フラグメント** : 省略可, リソース内の部分を指し示す
この例では, id属性の値が"more"である要素

- 次のURLについて, 以下の問いに答えよ
- <https://www.amazon.co.jp/Node-js%E8%B6%85%E5%85%A5%E9%96%80%E6%8E%8C%E7%94%B0%E6%B4%A5%E8%80%B6%E4%B9%83/dp/479805092X/?ie=UTF8&qid=1507178882&sr=8-1&keywords=Node#1>
- スキームは何か **https**
- ホスト名は何か **www.amazon.co.jp**
- クエリの最初の3文字は何か **ie=**
クエリの最後の3文字は何か **ode**

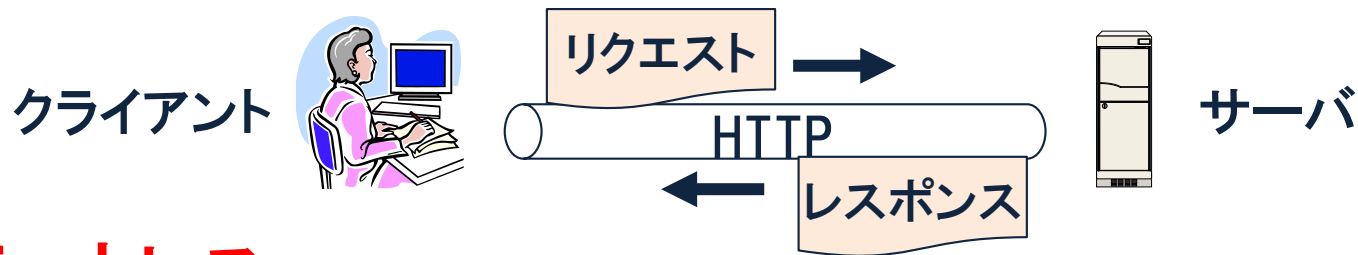
2.3 HTTP ～ 情報をどうやって手に入れるのか ～

■ Hypertext Transfer Protocol

- ブラウザとWebサーバ間でHTMLや画像などの情報をやりとりするためのプロトコル(通信手順/通信規約)

■ クライアント/サーバ

- クライアント(Webブラウザなど)がWebサーバに接続し、**リクエスト**(要求)を出し、**レスポンス**(返答)を受け取る



■ ステートレス

- 1つのリクエストと1つのレスポンスで完結
- サーバ側でクライアント側の状態(経路の情報)を持たない

2.3 HTTP (1) **リクエスト**メッセージ

http://www.example.jp/index.html に対するリクエスト

```
GET /index.html HTTP/1.1
```

```
Host: www.example.jp
```

```
User-Agent: Mozilla/5.0 (Windows NT 6.1)...
```

← リクエストライン

← ヘッダ

← 空行（この例ではなし）

← ボディ(この例ではなし)

■ 構成要素

- リクエストライン（1行目）

- ◆ **HTTPメソッド**, パス, プロトコルバージョン

- **ヘッダ**（2行目以降）

- ◆ メッセージのメタデータ「名前: 値」を複数記述可

- ボディ

- ◆ POST, PUTリクエストでは, リソースの表現そのものが入る

2.3 HTTP (2)レスポンスメッセージ

http://example.jp/test へのリクエストが成功した場合

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

<html>
... (略)
```

← ステータスライン
← ヘッダ
← 空行
← ボディ

■ 構成要素

- ステータスライン (1行目)
 - ◆ プロトコルバージョン, ステータスコード, テキストフレーズ
- ヘッダ (2行目以降)
 - ◆ メッセージのメタデータ「名前: 値」を複数記述可
- ボディ
 - ◆ GETリクエストでは, リソースの表現そのものが入る

2.3 HTTP (3)メソッド

メソッド	機能	CRUD
GET	リソースの 取得	Read
POST	子リソースの 作成 (URLを指定せず作成)	Create
PUT	リソースの 作成 (URLを指定して作成) リソースの 更新	Create Update
DELETE	リソースの 削除	Delete
HEAD	リソースのメタデータを取得	
OPTION	リソースがサポートするHTTPメソッドを取得	

GET, POST, PUT, DELETEは
データの基本4操作
CRUD(Create, Read, Update,
Delete)に対応する

2.3 HTTP (4)ステータスコード

skip

■ 分類

1XX	処理中 : クライアントはそのまま処理を継続するか, サーバの指示に従ってプロトコルをアップグレード.
2XX	成功
3XX	リダイレクト : クライアントは, レスポンスメッセージの Locationヘッダを見て新たなリソースへ接続する.
4XX	クライアント側 のエラー: エラーの原因はクライアントにある. エラーを解消しない限り正常な結果は得られない.
5XX	サーバ側 のエラー: エラーの原因はサーバ側にある. サーバ側のエラーが解決すれば, 同じリクエストを再送信して正常な結果が得られる可能性がある.

2.3 HTTP (4)ステータスコード 承前

skip

■ 代表的なステータスコード

	テキスト フレーズ	意味
200	OK	リクエスト成功
201	Created	リソースの作成成功. POSTリクエストへの応答. Locationヘッダーに新たなリソースのURLを含む.
301	Moved Permanently	リソースの恒久的な移動 (Locationヘッダに新たなURL)
303	See Other	別URLの参照 (Location ヘッダに新たなURL) POSTで操作した結果をGETで取得することを可能にする

2.3 HTTP (4)ステータスコード 承前

skip

	テキスト フレーズ	意味
400	Bad Request	リクエストの構文やパラメータに誤り
401	Unauthorized	認証情報が不適切（WWW-Authenticateヘッダで認証方式を伝える）
404	Not Found	URLがリソースと一致しない
409	Conflict	リソースに対する操作が、他のリソースと競合する（名前をすでに他で使われているものに変更するなど）
415	Unsupported Media Type	クライアントが指定したメディアタイプをサーバがサポートしない
500	Internal Server Error	サーバ側に問題がある
503	Service Unavailable	サービス停止（メンテナンスなどで一時的にアクセスできない）

2.3 HTTP (4)ステータスコード 承前

skip


- なぜ正しいステータスコードを返す必要があるのか
 - クライアントが状況を正しく把握できずに、システムの挙動に支障をきたす。
- 例: 404 Not Foundで返すべきときに200 OKで返すと,
 - WebAPIの場合
 - ◆ レスポンスメッセージのボディに「リソースがない」というエラーメッセージを入れて、200 OKで返すと、クライアントは、そのエラーメッセージを正常な結果として解釈しようとするかもしれない
 - Webサイトの場合
 - ◆ 検索エンジンのロボットが正式なリソースであると解釈し、インデックス処理を行ってしまうかもしれない

2.3 HTTP (5)ヘッダ

skip

- メッセージのメタデータを表現する
- クライアントやサーバはヘッダを見てメッセージに対する挙動を決定する
- 例1: 認証

保護されたリソースと知らずにアクセス



```
GET /test HTTP/1.1
Host: example.jp
```

サーバが認証情報付のリクエストを要求

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="MyData"
```

方法

対象

認証情報付のリクエストを送信

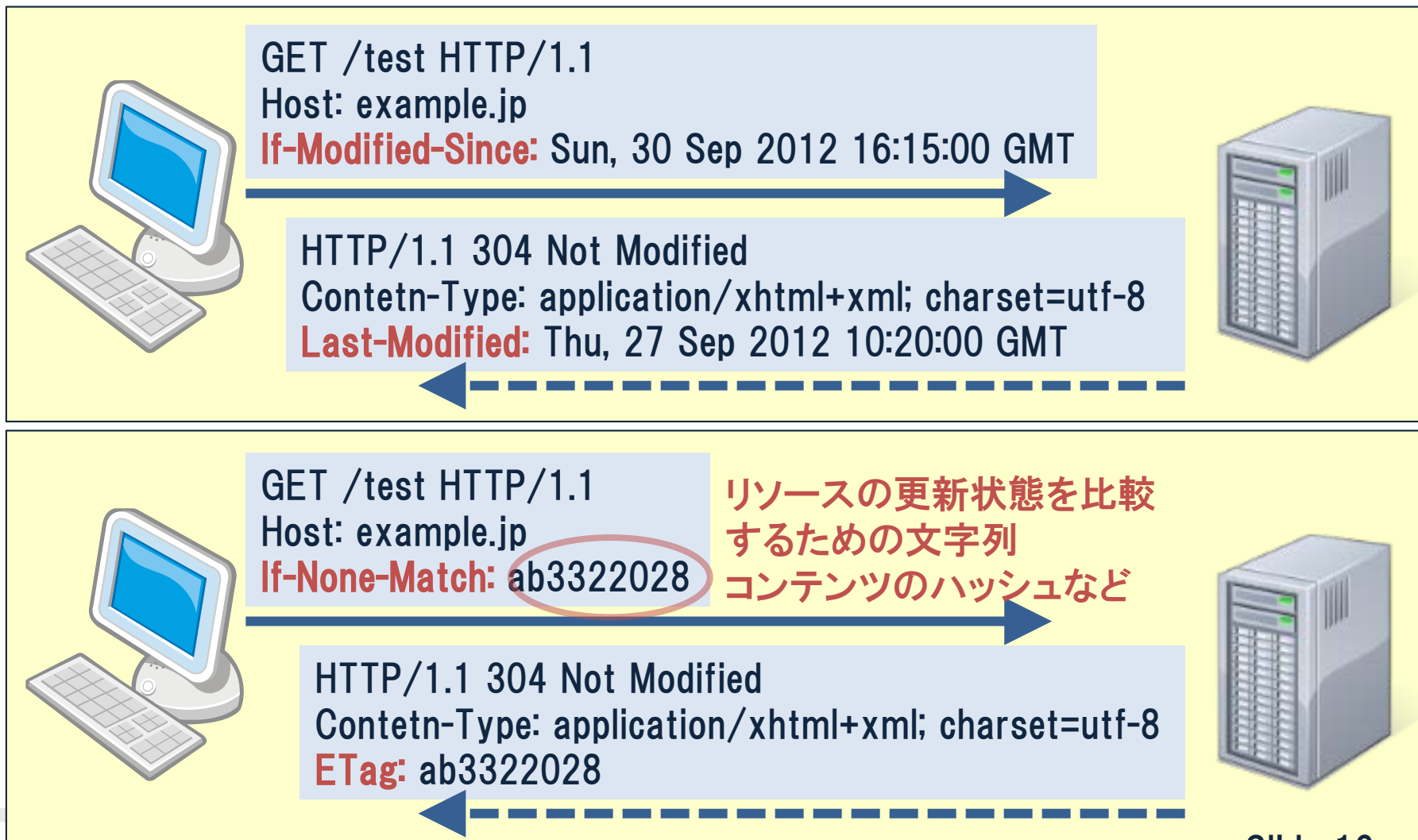
```
GET /test HTTP/1.1
Host: example.jp
Authorization: Basic QWxpYmFVulHN1c2F==
```

IDとパスワードをbase64
エンコーディング

2.3 HTTP (5)ヘッダ 承前

skip

■ 例2: キャッシュを利用するための条件付きGET (2種類の方法)



2.3 HTTP (5)ヘッダ 承前 (よくつかわれる)

skip

ヘッダ	タイプ	説明
Host	リクエスト	LRLのドメイン名部分(必須)
Date	リクエスト レスポンス	リクエストが送信された時間 リクエストが完了した時間(必須)
Content-Type	リクエスト レスポンス	クライアントがサーバに送信する表現の種類 サーバがクライアントに送信する表現の種類
User-Agent	リクエスト	リクエストを送信しているソフトウェアの種類
Location	レスポンス	リソースがアクセスすべきURL ステータスコード3XXに付随
Authorization	リクエスト	ユーザ名, パスワードなどの認証情報
WWW-Authenticate	レスポンス	認証情報の送信を要求することを示す ステータスコード401 Unauthorizedに付随
ETag	レスポンス	表現の特定のバージョンを示す文字列
If-None-Match	リクエスト	前回のGET時のETagの値
Last-Modified	レスポンス	表現が最後に変更された日時
If-Modified-Since	リクエスト	前回のGET時のLast-Modifiedの値

2.4 HTML ～ 情報をどうやって表現するのか ～

■ Hypertext Markup Language

- Webページを記述するための**マークアップ言語**
- マークアップ言語: <html><body><h1><p>
などの**タグ**で文書の木構造を表現する言語
- 開始タグ<XXX>と終了タグ</XXX>で囲まれた部分をXXX**要素**という
- 要素には「名前=値」の形式で**属性**を指定できる

■ 要素の例

```
<a href="http://example.jp/">リンク</a>
```



2.4 HTML (1)標準

■ HTMLの現在の標準

- HTML5.1 (2017年10月3日付 **W3C 勧告**)
- HTML5.2 (2017年10月5日現在 **W3C 勧告候補**)
- HTML Living Standard
(**WHATWG**が継続的にアップデートしている動的な仕様)

■ 位置づけ

- W3Cが策定するHTML 5.xはWHATWGのHTML Living Standardを基にしたスナップショットのようなもの
- HTML仕様はほぼ同じでだが、異なる点もある

WHATWG: Web Hypertext Application Technology Working Group
W3C: World Wide Web Consortium

2.4 HTML (2)構成

skip

```
<!DOCTYPE html>
<html>
  <head>
    <title>こんにちは, HTML!</title>
  </head>
  <body>
    <h1>Hello, HTML!</h1>
    <p>これは<a href="http://example.jp/">
      HTML</a>のサンプルです. </p>
  </body>
</html>
```

ドキュメントタイプ宣言

head要素

- タイトルや他の文書との関係など, 文書のメタデータを記述

body要素

- 文書の本文を記述
- 文書の構成を表現する
ブロックレベル要素と
- テキストに意味づけする
インライン要素

要素の例

- h1, h2, h3: 見出しを表すブロックレベル要素
- p: 段落を表すブロックレベル要素
- a: ハイパーリンクを表すインライン要素

2.4 HTML (3)CSS, Javascript

- HTMLはCSS, Javascriptという言語内言語をもつ
- **CSS** (Cascading Style Sheets)
 - HTMLをどのように表示するかを指定するスタイルシート
 - ◆ 各要素の大きさ, 色, レイアウトなどを指定
 - 文書の構造(HTML)と表示方法(CSS)を分離
- **Javascript**
 - ブラウザ内で使えるプログラミング言語
 - 動的なアプリケーションの構築に用いられる

■ インターネットに関するプロトコルや言語に関する記述のうち適切なものはどれか。 **イ**

ア: HTMLは、文書の論理構造を表すタグをユーザが定義できる言語である **XML**

イ: HTTPは、HTML文書などを転送するための通信プロトコルである

ウ: URLは、ネットワーク上のリソースを、「場所」という概念に依存せず、「名前」によって永続的に特定するための識別子である **URN**

エ: CSSは、XMLデータをHTMLデータへ変換するためのスタイルシートを記述する言語である **XSLT**

2.5 REST (Representational State Transfer)

- Webシステムのソフトウェアアーキテクチャスタイル
 - 機能ではなく、リソースを中心にものごとを考える
- Fieldingが2000年に博士論文で提唱
 - HTTPで転送されるのはハイパーテキストだけではない
 - リソースの**状態**(state)の**表現**(representation)を転送(transfer)している
- **リソース**
 - Web上の情報
 - ◆ 参照するにあたいする情報, 名前を付けるに値する情報
 - リソースは状態を持つ
 - リソースの状態は複数の表現を持つ (HTML, CSV, PDF, ...)

2.5 REST ～ RESTfulなWebアプリケーション ～

■ RESTfulなWebアプリケーション

- RESTの考え方に基づいたWebアプリケーション（下記の特徴をもつWebアプリケーション）をRESTfulなWebアプリケーションという

■ 特性

- アドレス可能性
- 統一インターフェース
- ステートレス
- 接続性

2.5 REST (1)アドレス可能性

- リソースの表現はアドレス可能である
 - 全てのリソースは少なくとも一つのURLをもつ
 - 一つのURLは唯一つのリソースを指す
 - ◆ 複数のリソースを指すことはない
 - ◆ URLには「サーバがどのデータを処理するか」の情報が含まれている
- アドレス可能であることにより
 - リンクできる, ブックマークできる, 資料に掲載できる
 - ◆ **にアクセスして, 条件に○○を入力して, △△ボタンを押したときに表示されるリストの×番目を見てと言わなくてよくなる
 - アプリケーション間で受け渡しできる

2.5 REST (2)統一インターフェース

- リソースは, HTTPメソッド(GET, PUT, DELETE, POST)によって処理される
 - どのリソースもこれらのメソッド(の一部)をサポートする
 - メソッドは, どのリソースに対しても同じように機能する
 - ◆ 新たな従属リソースを**作成**する(**POST**)
 - ◆ リソースの表現を**取得**する(**GET**),
 - ◆ リソースを**更新**する(**PUT**),
 - ◆ リソースを**削除**する(**DELETE**),
- 統一インターフェースによって
 - ユーザははじめてみるリソースでもどのように操作すればよいかが分かる
 - GETの**安全性**, PUTとDELETEの**べき等性**によって, 信頼性が向上する

2.5 REST (2)統一インターフェース 承前

- GETは「安全」でなければならない
 - サーバ(リソース)の状態の変更を要求してはならない
 - GETリクエストを何度送信しても, リクエストを全く送信しない状態と同じでなければならない
- PUTとDELETEは「べき等」でなければならない
 - 特定のURLに複数のPUTまたはDELETEを送信しても, リクエストを1回だけ送信した場合と同じ結果が得られなければならない ※ 安全な操作はべき等である
 - ◆ レスポンスがなければリクエストを再送できる.
 - ◆ 信頼できないネットワークで信頼性を向上させられる
- POSTリクエストは, 従属リソースの生成(または, リソースへのデータの追加)のみに使用しなければならない

2.5 REST (2)統一インターフェース 承前

リソースに対する典型的なアクション(操作)

■ リソースの

- 一覧取得
- 作成
- 取得
- 更新
- 削除

■ 例

- ブログの投稿
- 在庫管理システムの商品
- スケジュール管理の一つの予定
- メモアプリの一つのメモ

投稿の一覧表示画面

**ブログ		
新規作成		
○月○日 大学院入試	表示	編集 削除
○月×日 研究会で発表	表示	編集 削除
...		

投稿表示画面

○月○日 大学院入試 ... (ブログ記事の詳細) ...

投稿編集画面

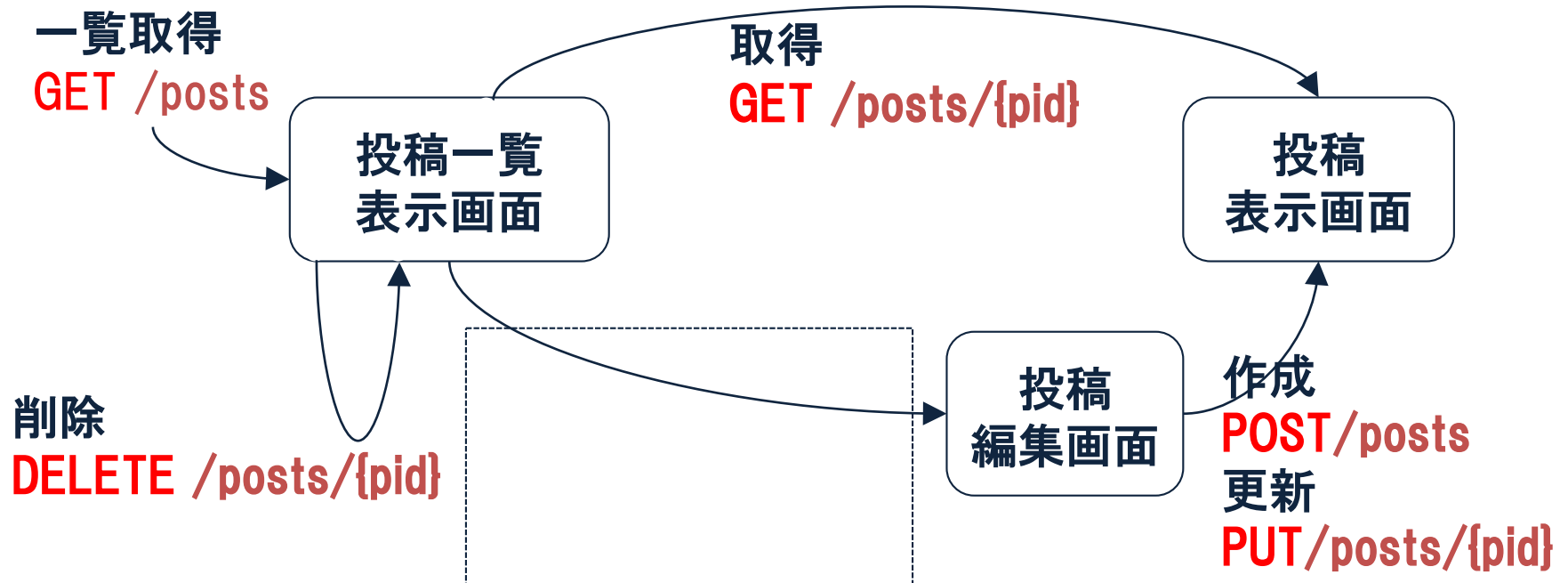
挿入	書式	...
○月○日 大学院入試 ... (ブログ記事の詳細) ...		

2.5 REST (2)統一インターフェース 承前 アクションと統一インターフェース, 画面遷移

■ 例: ブログの統一インターフェースと画面遷移

- 5つのアクション(操作)

- ◆ 一覧取得, 作成, 取得, 更新, 削除



2.5 REST (3)ステートレス

- サーバが**クライアント(アプリケーション)の状態を格納しない**
 - アプリケーション状態を考慮したい場合は、それをリクエストの一部として送信する(認証情報など)
 - リクエストは個別に処理される(以前のリクエストの情報に依存しない)
- ステートレスによって
 - **負荷分散**が容易:どのサーバに処理させても良い
 - **信頼性**が向上:回復不能な状態に陥ることを心配せずにクライアントはリクエストを再送できる
 - アプリケーション状態を保存可能:URLに必要な情報が全て含まれるので、**ブックマーク**できる(検索結果の50ページ目を見るのに、「次」を50回たどる必要はない)

2.5 REST (4) 接続性

- サーバは、リンクとフォームを送信することにより、クライアント(アプリケーション)の状態を遷移させる
 - リンクをたどったり、フォームに入力することにより、アプリケーション内を移動することができる
 - リンクがなくても、URL構築のルールを知っていれば、目的の状態に遷移させることはできるが、...
- 接続性により（リンクがあれば）
 - URL構築のルールを知らなくてもよい
 - URL構築のルールが変更されても、ルールを学習しなおす必要がない
- Webアプリケーションでは当たり前だが、Web APIにおいては当たり前ではない

2.5 REST (5) 演習

演習

次の文はRESTfulアプリケーションの利点を記したものである。□を埋めて文を完成させよ。

- アドレス可能性によって、リソースにリンクをはることができる。また、ブックマークしたり、資料に掲載したりできる。さらに、アプリケーション間で受け渡しできる。
- GETの安全性とPUTとDELETEのべき等性によって信頼性が向上する
- ステートレスによって負荷分散が容易なる。