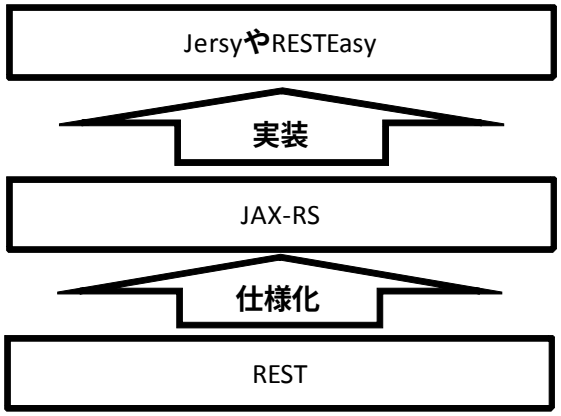
JAX-RS

1.JAX-RS とは

JAX-RS とは Java API for RESTful Web Service の略です。JAX-RS は、REST というアーキテクチャスタイルで Web サービスを作成するための高レベルな API を規定します。JAX-RS は JCP 内で JSR-311 として標準化されました。

JAX-RS の参照実装は https://jersey.dev.java.net/ で開発されている Jersey です。その他にも JAX-RS に完全準拠しているプロダクトに RESTEasy があります。

図 1：REST、JAX-RS、Jersey、RESTEasy の関係



また、JAX-RS は Java EE 6 から完全な Java EE に含まれる予定になっています。

なお、JAX-RS はアノテーションを使用することが前提の仕様であるため、サポートする Java のバージョンは J2SE 5.0 以降です。

2.RESTful Web サービスとは

RESTful Web サービスとは、REST というアーキテクチャスタイルに則って作られている Web サービスを指しています。では、REST とは何でしょうか。REST とは Representational State Transfer の略です。REST は Roy Thomas Fielding 氏が博士論文で発表したアーキテクチャスタイルです。RESTful Web サービスは Amazon S3 や Google App Engine を RESTful 対応にする App Engine RESTful Server などすでにこのアーキテクチャスタイルを採用したサービスが存在します。

RESTful Web サービスは、SOAP を用いた Web サービスの複雑さを回避し、HTTP プロトコルの原則に忠実な Web サービスを目指すものです。

そして、RESTful Web サービスは以下のような特徴を持ちます。

(ア) リソース中心

リソースとは Web サービスで取り扱う名前全般を指しています。リソースは「りんご」や「田中太郎」といった直接そのものを指すこともできますが、「最新の定例会議議事録」や「今週のベストセラートップ 10」のように状況の変化によって直接指し示すものが変わるような名称もリソースとなります。

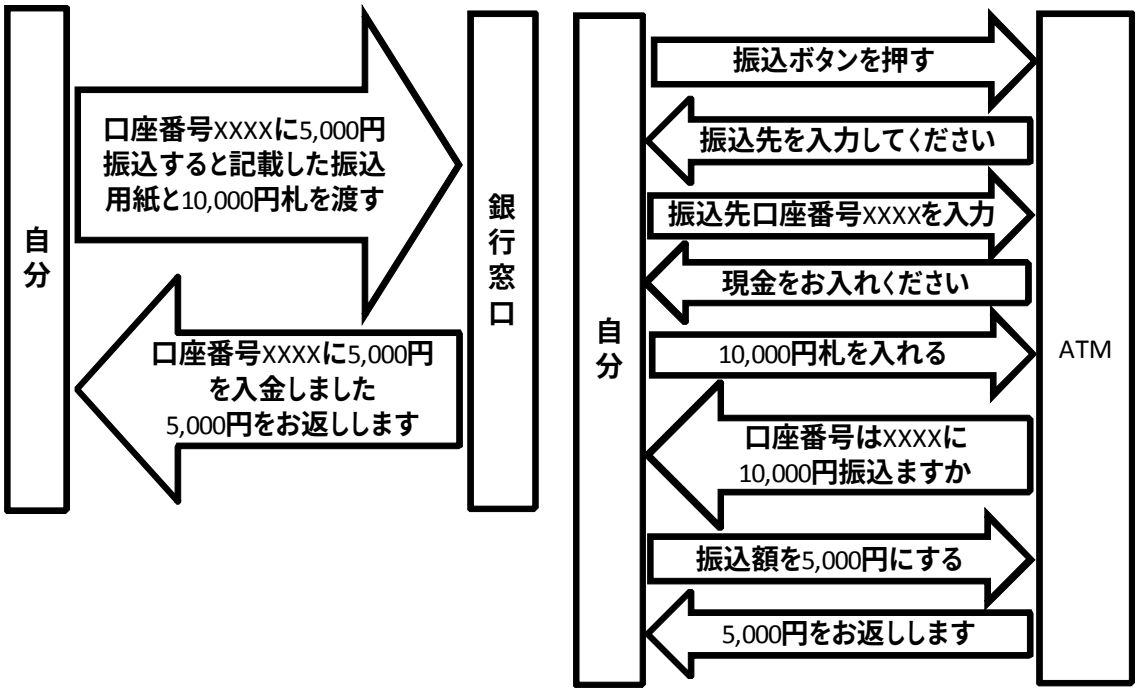
そして、RESTful Web サービスは URI でリソースを表現しており、アクセスすることでリソースを操作します。それに対し、既存の Web サービスは URI で操作を表現しており、アクセスすることで Web サービスを操作しています。

また、リソースは想像できる記述であることが推奨されます。URI がある程度想像可能でサービスの利用が簡単であることが RESTful Web サービスのメリットを高めます。たとえば、「りんご」であれば、URI は「http://www.mysystem.jp/foods/apple」となります。このような URI であれば、次に「梨」というリソースを追加したい場合は URI の「apple」の所を「pear」に書き換えたてアクセスすればよい、と想像が容易です。

(イ) ステートレスである

RESTful Web サービスでのリクエストは状態を持ちません。このことを指して RESTful Web サービスはステートレスであると言います。REST では状態も一つのリソースとしてとらえます。ステートレスな状態とステートフルな状態とを銀行に入金する時を例にとって説明します。以下の図をご覧ください。

図 2：ステートレスとステートフル



銀行で振込をする場合、窓口と ATM どちらかでおこないます。窓口ではあらかじめ要望を振込用紙に記入し、窓口に渡すと、要望がすべて伝わります。これのように状態を記憶しないことを指してステートレスと呼びます。それに対して ATM では画面と音声ガイダンスの指示に従いステップバイステップで要望を段階的に伝えています。このように状態を記憶していることを指してステートフルと呼びます。

RESTful Web サービスがステートレスであるということは、1 つのリクエストを受け、レスポンスを返すという 1 対のデータの受け渡しが独立していることを表しています。これは HTTP がリソースの状態のためのプロトコルであり、クライアント側の状態を管理しないためです。これにより、RESTful Web サービスはセッション管理（セッションタイムアウト時の処理や、セッションレプリケーション等）から解放され、システムをシンプルにすることができます。つまり、RESTful Web サービスはサーバに持つリソースの状態

だけを管理する Web サービスです。

(ウ) リソースに対する操作が HTTP のメソッドを通じて行われる

リソースに対するアクセスに REST では HTTP のメソッドを使用します。REST で主に使用する HTTP のメソッドは以下の通りです。

表 1：REST で主に使用する HTTP メソッドの一覧

メソッド	操作内容
GET	リソースを取得
PUT	既存リソースの変更
POST	主に新規リソースの作成
DELETE	リソースの削除
HEAD	メタデータの取得
OPTIONS	アクセス可能な HTTP メソッドを取得

REST では名詞であるリソースに対して動詞である HTTP メソッドを使用し、HTTP メソッドに応じたその時点のリソースの表現を取得します。このように REST は HTTP の標準的なメソッドを使用します。これを指して「統一インターフェース」といいます。

(エ) ハイパーリンクを使用してリソースの参照を行う

HTTP の特徴にハイパーリンクを使ってリソースを関連付ける特徴があります。RESTful Web サービスもハイパーリンクを使い、リソースをからリソースに移動可能にします。例えば「http://www.mysystem.jp/foods/」以下にどのようなリソースがあるか、初めてアクセスしたときにはわかりません。以下にハイパーリンクを使ってリソースを探す流れを図にしました。

図 3：ハイパーリンクを使って目的のリソースを探す流れ

「foods」にアクセスすると、配下に存在するリソースの一覧をハイパーリンクとして表示します。表示されたリンクの中から目的のリソース、図ではパンをクリックし、目的のリソースにたどり着くことができます。

このようにして、リソースの接続性が大事であるといわれています。十分な接続性があるという場合は、サーバ側が用意したリンクに従い、目的のリソースにたどり着けることにあります。

また、この接続性が存在することにより、内部のリソースの URI が変更された場合にもクライアントが影響を受けることを防ぎます。ただしこれは、クライアントが Web サービスのルートからリンクを使い、目的のリソースにたどり着くようにしなければなりません。クライアントがリソースの規則性からダイレクトに特定のリソースを推測するような場合はこの接続性のメリットを失います。

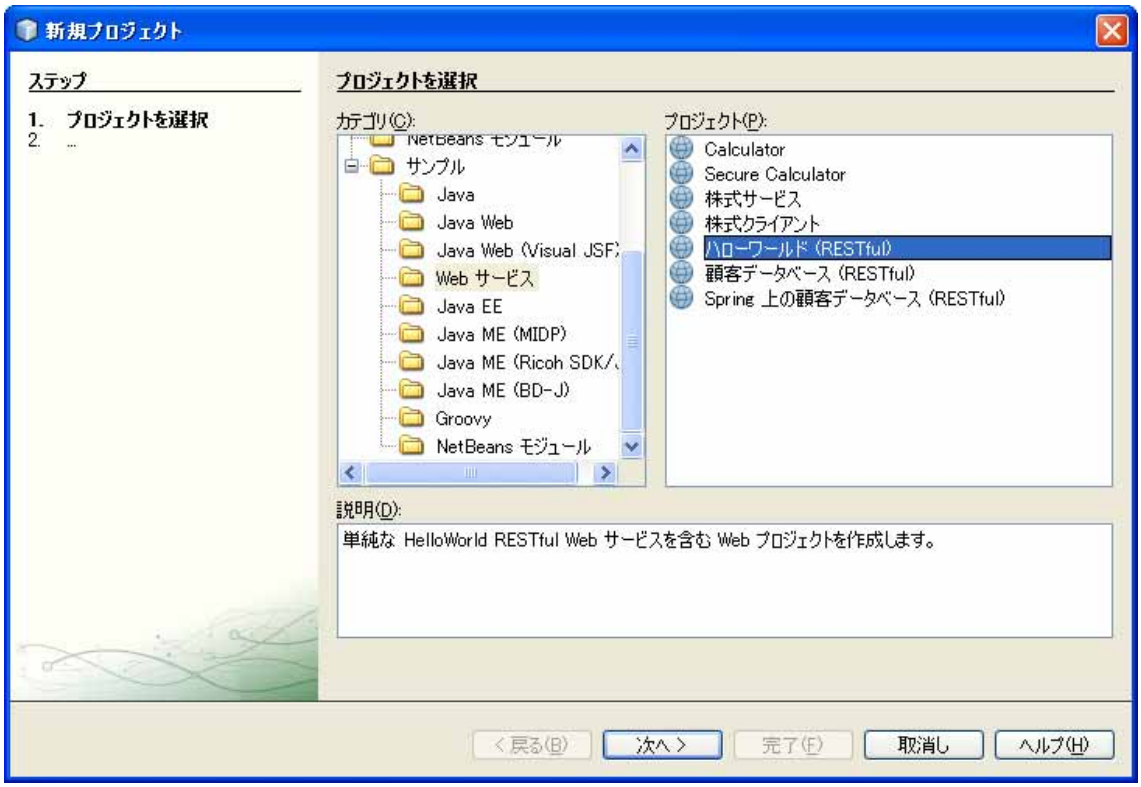
3.JAX-RS の参照実装による RESTful Web サービスの実際

JAX-RS の参照実装である Jersey を使い実際に RESTful Web サービスをどのように作成するのか、サンプルを参考に追っていきます。なお、サンプルには NetBeans6.5.1 付属のサンプル「ハローワールド（RESTful）」を使用します。NetBeans は http://ja.netbeans.org/からダウンロードできます。NetBeans は複数のエディションが存在しますが、サポートテクノロジの「Java Web および EE」が含まれるものをダウンロードしてください。

今回開発環境に、NetBeans を採用します。

JAX-RS では URI とクラスのマッピング、HTTP メソッドに対してどの Java メソッドを割り当てるか、といった RESTful Web サービスを実現するためにアノテーションを用います。NetBeans で新規プロジェクトウィザードを起動し、下の図のプロジェクトを作成してください。

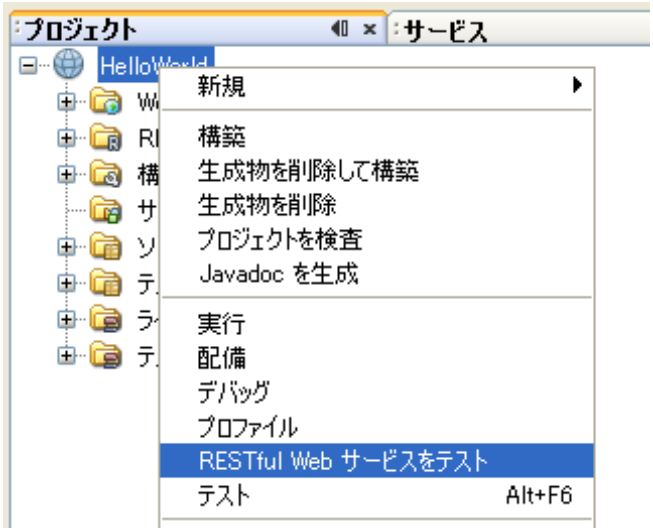
図 4：プロジェクト作成ウィザード画面



ウィザードはデフォルト値で進めて下さい。ウィザードを完了すると Hello World プロジェクトが作成されます。このプロジェクトの Java ソースは HelloWorldResource だけです。

なお、サンプルを実行するには、作成したプロジェクトを右クリックし、「RESTful Web サービスをテスト」を選択すると、プロジェクトがデフォルトの AP サーバに配備され、実行可能な状態になります。

図 5：サンプルの起動方法



「RESTfull Web サービスをテスト」を選択すると、サンプルプロジェクトが実行可能となるのと同時に、試験を行うためのクライアントがブラウザに表示されますので、そこからサービスにリクエストを送り、結果を確認することができます。

図 6：サンプル操作画面

3.1. HTTP メソッド

HTTP の標準メソッドと Java メソッドの対応付けを HTTP メソッドに応じたアノテーションで設定します。HTTP メソッドのアノテーションは。「２．（ウ）」で挙げた HTTP メソッドの先頭に”@”をつけたものとなります。（例：HTTP メソッドの GET⇒@GET）サンプルでは以下のようになっています。

リスト 1：HTTP メソッドアノテーションの例

```
@GET
@Produces("text/html")
public String getXml() {
```

例として取り上げた”getXml”メソッドはクラスのアノテーションで定義した URI に GET メソッドでアクセスした場合に処理が実行されます。なお、”@Produces”については後述しますので、ここでは気にしないで下さい。

3.2. URI マッピング

URI マッピングはクラス、またはメソッドに URI のパスを割り当てる設定です。アノテーションは”@Path”を使用します。
サンプルでは以下のようになっています。

リスト 2：@Path の例

```
@Path("/helloWorld")
public class HelloWorldResource {
```

このアノテーションの設定で HelloWorldResource クラスが”/helloWorld”に割り当てられました。このサンプルの場合、URI は ”http://localhost:8080/HelloWorld/resources/helloWorld”となります。

今回のサンプルにはありませんが、クラスとメソッドの両方に”@Path”のアノテーションを設定した場合、メソッドの”@Path”はクラスに設定した”@Path”のサブリソースとなります。では、サンプルの”getXml”メソッドに”@Path”アノテーションを追加し、サブリソースを作ってみます。

リスト 3：サブリソースのメソッド宣言の例

```
@GET
@Path("subHello")          ← この行を追加
@Produces("text/html")
public String getXml() {
```

ソースを書き換え、プロジェクトを右クリックし、「REST サービスをテスト」を選択すると、テスト画面が以下のように変わり、サブリソースが作られたことが確認できます。

図 7：サブリソース作成後のサンプル操作画面

3.3. レスポンスデータの形式

レスポンスデータの形式は”@Produces”アノテーションで設定します。リスト 1 のメソッドの例で説明しなかった部分がこのレスポンスデータの形式を設定している箇所になります。”@Produces”アノテーションは、同じ URI かつ同じ HTTP メソッドであっても複数種類の表現を返せるようにする仕組みです。HTTP リクエストのヘッダにはコンテンツタイプを指定しています。サンプルではコンテンツタイプが”text/html”のリクエストに対して”getXml”メソッドの結果を渡します。

では、テスト画面で実際に確認してみましょう。テスト用の画面で右ペインのリソース名がリンクになっているのでクリックしてください。

図 8：リソース名クリック後のサンプル操作画面

図 8 の画面ではあらゆるリクエストを helloWorld リソースに送ることができます。ここで、初期状態のままテストボタンを押してみましょう。そうすると、画面の下に「状態: 406 (Not Acceptable)」が表示されていると思います。これは”@Produces”アノテーションにない形式を要求したため、HTTP プロトコルのエラーコード 406 を返しました。

また、一つのメソッドに複数のレスポンスデータの型を関連付けることができます。”@Produces”アノテーションの設定を下記のように記述すると複数のレスポンスデータの型に関連づきます。

リスト 4：複数のレスポンスデータの型に対応させるアノテーションの例

```
@Produces({"application/xml", "application/json"})
```

このアノテーションが設定されたメソッドはレスポンスデータ型が”application/xml”の時と”application/json”の時に呼び出されるようになります。

3.4. リクエストデータの形式

リクエストデータの形式は”@Consumes”アノテーションで設定します。 ”@Consumes”アノテーションはリクエストのヘッダのコンテンツタイプに応じたメソッドが選択されます。これは、”@Produces”アノテーションとよく似ています。“@Produces”との違いは、リクエストデータのボディ部のデータが引数に入るところです。

リスト 5：@Consumes の例

```
@PUT
@Consumes("application/xml")
public void putXml(String content) {
```

この例では、リクエストのヘッダのコンテンツタイプが”application/xml”であるリクエストを受け取り、リクエストのボディ部が String 型の content に代入されます。

3.5. パスパラメータの取得

パスパラメータは”@PathParam”アノテーションでパスの特定の部分を引数として受け取りメソッド内で使用するためのアノテーションです。このパラメータは”@Path”アノテーションと組み合わせて使用します。

”@Path”アノテーションではリソースの名称を設定するためのアノテーションですが、アノテーションの引数を” {customerId}/”のように中括弧を指定するとその部分は可変な文字が設定されることを表します。この可変部分を受け取るために、以下のように”@PathParam”アノテーションをメソッドの引数部分に記述します。

リスト 6：@PathParam の例

```
@Path("{customerId}/")
public CustomerResource getCustomerResource(
        @PathParam("customerId") Integer id) {
```

これで、このメソッドに可変のパス”customerId”を変数 id として使用することができます。

3.6. クエリパラメータの取得

クエリパラメータの取得は”@QueryParm”アノテーションで設定します。クエリパラメータは URI の末尾にクエスチョンマークを付けてパラメータを付加するものを指しています。

リスト 7：クエリパラメータの例

```
http://localhost:8080/HelloWorld/resources/helloWorld?param=123
```

このクエリパラメータを取得するにはメソッド宣言を下記のようにします。

リスト 8：”@QueryParam”の例

```
public String getXml(@QueryParam(“param”) int num) {
```

このクエリパラメータと同様のアノテーションが”@FormParam”アノテーションです。これはフォームの入力内容をそのフォーム名に関連付けて取得することができます。

4.まとめと JAX-RS の今後

JAX-RS の元となっている思想である RESTful Web サービスについて説明し、JAX-RS の参照実装である Jersey で RESTful Web サービスをどのように作るか具体的な例を元に説明しました。

現在、REST のシンプルさから来る利便性の高さのため、インターネット上で提供されている Web サービスの殆どは RESTful Web サービスという状況です。そして、最近ではクラウドコンピューティングの分野でも RESTful な API が提供されています。

このように REST が広まる中で、Java は RESTful Web サービスを提供する環境について、PHP や Perl、Ruby と比べて遅れを取ってきました。しかし JAX-RS により、RESTful Web サービスを Java で実現することが可能な環境が整ってきており、いよいよ Java で RESTful Web サービスを作る機会が増えてくるのではないか、と思われます。

5.参考文献

- 『RESTful Web サービス』
  著者：Leonard Richardson、Sam Ruby
  発行所：株式会社オライリー・ジャパン
  ISBN：978-4-87311-353-1

- 『WEB+DB PRESS Vol.42』（特集３ 現場で使える REST）
  発行所：株式会社技術評論社
  ISBN：978-4-7741-3331-7

- 『The Java Community Process(SM) Program - JSRs: Java Specification Requests - detail JSR# 311:』（英語）
  URL：http://jcp.org/en/jsr/detail?id=311

- 『Architectural Styles and the Design of Network-based Software Architectures』（英語）
  URL：http://roy.gbiv.com/pubs/dissertation/top.htm

- 『RESTful Web サービスについて - NetBeans IDE 6.5 チュートリアル:』
  URL：http://www.netbeans.org/kb/docs/websvc/rest_ja.html

- 『Studying HTTP』
  URL：http://www.studyinghttp.net/

開発部
多田　丈晃