ZOO Project

オープンソース WEB プロセシングサービスエンジンの展望

社会事業部　企画室
林　　博文
帝塚山学院大学　リベラルアーツ学部
吉田　大介

1. はじめに

地理空間情報システム（GIS）の業界では近年、2005 年の Google のマップサービスの開始を機に地理空間情報の自由化と相互利用が始まり、現在日本では、国土交通省や国土地理院をはじめとする多くの行政機関や団体で GIS データが公開され、自由に利用ができるようになった。国際的にも、「Gov2.0」というキーワードと共に各国が公共データの民間活用に積極的に取組んでいる。

GIS データが自由に活用できることは良いことであるが、一般の人々にとって、GIS の生データをそのまま活用するというのは技術的にもコスト的にも敷居が高く、もっと簡単にデータが扱え、自分たちの目的の実現のために利用したいと考えている。これを実現するための基盤技術が「ウェブプロセシングサービス（Web Processing Service：WPS）」と呼ばれるものである。最近話題のクラウドコンピューティングの基盤的な技術である WPS は、現在 GIS 業界においてもっとも注目されている技術で、様々な分野で応用が可能である。

1.1　WPS と ZOO

WPS は利用者が必要とするデータや処理をサーバー側に URL の引数として要求して、XML で結果を返答するサービスである。例えば Google API のように、処理の詳細を知らなくても、「住所」を入力して「緯度経度」を得ることがとても簡単にできるようになる。WPS を介して GIS データやその他のサービスを利用することで、データ交換や処理サービスの共通化を図ることができ、巨大なデータを取り扱う上でのコスト削減、さらに利用者のアプリケーションが様々なデータを利用することが可能になる。

WPS はネットワークを介して GIS データ等を交換する手順やデータの共通仕様に基づいて、機能仕様が決定されている[1]。標準を定めているのは Open Geospatial Consortium（OGC [2]）で、規格を通じた地理空間情報の共有に取り組む国際標準化団体である。OGC 規格に準拠した WPS エンジンをオープンソースで開発しているプロジェクトは世界に4つあり、「52°　North　WPS」、「deegree WPS」、「pyWPS」、そして「ZOO」である。

WPS はサーバー側でプログラム言語を使用してサービスを記述することで、クライアントからのリクエストにより GIS データの解析を行い、処理結果を利用者に返す機能を実現している。プログラム言語はプロダクトにより様々で、中でも最後発の「ZOO」は、多言語に対応したインタフェース（表1）と簡単な記述によるサービスの実装と、チェイン機能（サービスとサービスの連携）を実現している。

表 1 対応言語

（◎は ZOO Kernel ネイティブサポート、○はオプション調整が必要）

言語名	サポート	コンパイラ/インタプリタ
C / C++	◎	GCC4
Python	○	Python interpreter
Fortran	○	g90
PHP	○	PHP embedded
Java	○	Java SDK
Javascript	○	SpiderMonkey
Perl	○	Perl interpreter

「ZOO」という名称の由来は、この多言語対応によって、オープンソースの利用者や開発者が既に所有している技術（プログラム）を、ウェブサービス化するというアイディアが、オープンソースの各プロダクトが持つ、象やペンギン、牛などのマスコットキャラクタを集めた動物園（ZOO）に見立てていることから来ている。

ZOO プロジェクトは FOSS4G 2008 カンファレンスの間に、Gerald FENOY 氏、 Nicolas BOZON 博士、 Venkatesh RAGAHAVAN 教授によって構想され、大阪市立大学、帝塚山学院大学、応用技術株式会社等幾つかのパートナーが積極的なプロダクトの支援を行っている。

ZOO は「ZOO プロジェクト[3]」コミュニティにより開発サポートが行われており、MIT/X11 ライセンスで提供され、ソースコードとバイナリは商用として自由に使用することが出来る。オープンソース地理空間情報の国際団体、OSGeo 財団が主催する「FOSS4G 2010 カンファレンス」で配布された「OSGeoLiveDVD[4]」に収録されており、汎用 PC を使用して DVD や USB から起動することで、ZOO を簡単に学習することが出来る。

2. ZOO の仕組み

ZOO はサーバー側でユーザーからのリクエストを受け、タスク管理を行う「ZOO Kernel」、処理サービスを記載するメタデータファイル（.zcfg）によりウェブサービス化されたサービスである「ZOO Service」、サービスチェインを実現するスクリプト「ZOO API」から構成される（図 1）。

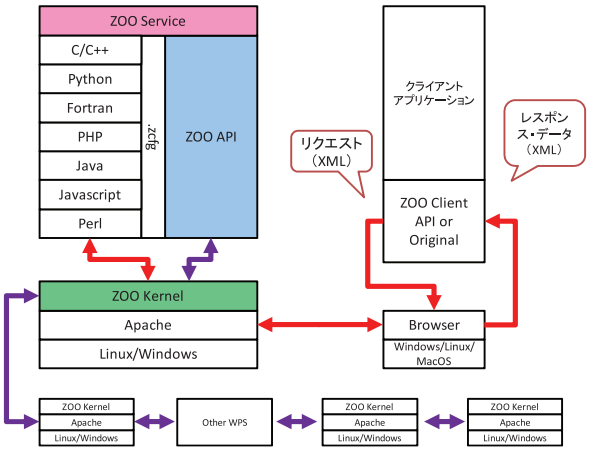

図 1 ZOO のシステム構成

2. 1 ZOO Kernel

ZOO Kernel について開発者がコードを追加する必要はない。サポート言語を追加する場合は必要となるが、コミュニティが開発者のサポートを行っている。[3][6][7]

ZOO Kernel は Apache ウェブサーバーから呼び出された際に、受け取ったリクエストを解析し、必要な処理を呼び出すタスクマネジメントの役割を実行する。ユーザーが送信するリクエストは次のような URL である。

```
http://localhost/cgi-bin/zoo_loader.cgi?
Request=GetCapabilities&Service=WPS
```

GetCapabilities リクエストでは、WPS の実行可能なサービスを下記のように返答する。

```
−<wps:Capabilities xsi:schemaLocation="http://www.opengis.net/wps
 /wps/1.0.0/wpsGetCapabilities_response.xsd" service="WPS" version="
 −<ows:ServiceIdentification>
    <ows:Title>The Zoo WPS Server</ows:Title>
    <ows:Abstract>FOSS4G 2010 - WorkShop ZooWPS.</ows:Abstrac
    <ows:Fees>None</ows:Fees>
    <ows:AccessConstraints>none</ows:AccessConstraints>
   −<ows:Keywords>
      <ows:Keyword>WPS</ows:Keyword>
      <ows:Keyword>GIS</ows:Keyword>
      <ows:Keyword>buffer</ows:Keyword>
    </ows:Keywords>
    <ows:ServiceType>WPS</ows:ServiceType>
    <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
  </ows:ServiceIdentification>
```

2. 2 サービスの実装

サービスは ZOO Kernel がプログラムの必要な機能を呼び出すことで生成される。必要な機能については、「.zcfg」の拡張子を持つ ZOO メタファイルの記述により、サーバー側にサービスとして実装される。「.zcfg」形式での記載例は次の通りである。

```
[Boundary]
    Title = Compute boundary.
    Abstract = A new geometry object is created and returned
containing the boundary of the geometry on which the method is
invoked.
    processVersion = 1
    storeSupported = true
    statusSupported = true
    serviceProvider = ogr_service.zo
    serviceType = C
    <MetaData>
      title = Demo
    </MetaData>
    <DataInputs>
     [InputPolygon]
     Title = Polygon to compute boundary
     Abstract = URI to a set of GML that describes the polygon.
      minOccurs = 1
      maxOccurs = 1
      <MetaData lang="en">
       title = Mon test
      </MetaData>
      <ComplexData>
       <Default>
        mimeType = text/xml
        encoding = UTF-8
        schema = http://fooa/gml/3.1.0/polygon.xsd
       </Default>
       <Supported>
        mimeType = text/xml
        encoding = base64
        schema = http://fooa/gml/3.1.0/polygon.xsd
       </Supported>
      </ComplexData>
    </DataInputs>
    <DataOutputs>
     [Result]
      Title = The geometry created
      Abstract = The geometry containing the boundary of the
geometry on which the method is invoked.
      <MetaData lang="en">
       title = Mon test
      </MetaData>
      <ComplexData>
       <Default>
        mimeType = application/json
        encoding = UTF-8
        extension = js
        asReference = true
       </Default>
       <Supported>
        mimeType = text/xml
        encoding = UTF-8
        schema = http://fooa/gml/3.1.0/polygon.xsd
        extension = xml
       </Supported>
       <Supported>
        mimeType = text/xml
        encoding = base64
        schema = http://fooa/gml/3.1.0/polygon.xsd
        extension = xml
       </Supported>
      </ComplexData>
  </DataOutputs>
```

.zcfg の構造は単純で、3つのブロックからなる。

1) サービスプロバイダ

最初の行はクライアントリクエストで指定される「Identifier」名を記載する。例では[Boundary]を指定しており、これにより ogr_service.zo プログラムの Boundary 関数が呼び出される。DataInputs では、関数の変数を設定する。 サービスのタイトル、概要、モジュール名、言語、動作規定について記載を行う。他に追加事項があれば<metadata>タグ内に記載する。データ処理結果の保存やステータス情報の通知機能が設定可能である。

2)インプット

クライアントから送信されるリクエストについての記載を行う。<DataInputs>タグ内に変数名を記載する。例では[InputPolygon]を定義している。この変数は、実際に作成するモジュール(各言語で記述することが可能)で[Boundary]という関数を記載した際、リクエスト引数に含まれる InputPolygon キーワード以下が、サーバー側で処理可能なデータ形式と、初期値を指定している。データ形式については、<Supported>タグで複数定義することが出来る。

3)アウトプット

クライアントに返答するレスポンスについての記載を行う。<DataOutputs>タグ内に変数名を記載する。例では[Result]を定義している。インプットと同様に、データフォーマットについて複数定義することが出来る。

2. 3 サーバー処理モジュール

サーバー処理モジュールの記述例は次の通りである。

```
#include "ogr_api.h"
#include "service.h"
extern "C" {
#include <libxml/tree.h>
#include <libxml/parser.h>
#include <libxml/xpath.h>
#include <libxml/xpathInternals.h>
//<YOUR SERVICE CODE AND OTHER UTILITIES FUNCTIONS>
int Boundary(maps*& conf, maps*& inputs, maps*& outputs){
    OGRGeometryH geometry, res;
    map* tmp=getMapFromMaps(inputs, "InputPolygon", "value");
    if(tmp==NULL)
      return SERVICE_FAILED;
    map* tmp1=getMapFromMaps(inputs, "InputPolygon", "mimeType");
if(strncmp(tmp1->value, "application/json", 16)==0)
      geometry=OGR_G_CreateGeometryFromJson(tmp->value);
    else
      geometry=createGeometryFromWFS(conf, tmp->value);
    res=OGR_G_GetBoundary(geometry);
    tmp1=getMapFromMaps(outputs, "Result", "mimeType");
    if(strncmp(tmp1->value, "application/json", 16)==0){
        addToMap(outputs->content, "value",
          OGR_G_ExportToJson(res));
        addToMap(outputs->content, "mimeType", "text/plain");
    }
    else{
        addToMap(outputs->content, "value",
          OGR_G_ExportToGML(res));
    }
    outputs->next=NULL;
    OGR_G_DestroyGeometry(geometry);
    OGR_G_DestroyGeometry(res);
    return SERVICE_SUCCEEDED;
}
<略>
}
```

この例では先ほど.zcfg で定義した、Boundary 関数を C 言語で実装している。ZOO のモジュールには通常 C 言語にあるローダー部の main()関数がなく、.zcfg で定義した Boundary()関数を作成する。Python や他の言語についても同様の処理記載で、サービスを動かすことが出来るようになる。なお、プリコンパイルが必要な C、C++、Fortran などは、make であらかじめコンパイルしたオブジェクトファイルを作成しておく必要がある。

コンパイル後のオブジェクトファイル名は、.zcfg に記載した「serviceProvider=<オブジェクトファイル名>」の行になる。 DataInputs と DataOutputs で

指定した形式に従った、入力データパースと出力データパースを、実際の処理関数の前後に呼び出すことで、ZOO Kernel を通じてアクセスを提供している。

2. 4 クライアント処理

クライアント処理は、ZOO WPS をサービスするサーバーにリクエストの URL 文字列を送ることで実行される。呼び出す機能は「Identifier」以降に記載する。

下記はクライアント側のリクエスト URL である。

```
http://localhost/cgi-bin/zoo_loader.cgi?request=Execute&service=WPS&version=1.0.0&Identifier=Boundary&DataInputs=InputPolygon=Reference@xlink:href=http%3A%2F%2Flocalhost%3A8082%2Fgeoserver%2Fows%3FSERVICE%3DWFS%26REQUEST%3DGetFeature%26VERSION%3D1.0.0%26typename%3Dtopp%3Astates%26SRS%3DEPSG%3A4326%26FeatureID%3Dstates.15
```

この例では ZOO WPS サーバーに対して Boundary 機能を呼び出し、入力データとして InputPolygon のデータとして、GeoServer に格納された地図レイヤー「states」から、「FeatureID=15」のデータを返すように指定している。今回の例では、.zcfg にリクエスト結果を JSON 形式として指定しているので、次のようなレスポンスが得られる。

レスポンスは XML 形式で得られるので、JSON や他の XML パースライブラリで容易に対応でき、処理能力の低い端末でも比較的軽いデータ処理が実現できる。

```
<wps:Execute service="WPS" version="1.0.0"
xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 ../wpsExecute_request.xsda">
<ows:Identifier>Boundary</ows:Identifier>
<wps:DataInputs>
  <wps:Input>
    <ows:Identifier>InputPolygon</ows:Identifier>
    <wps:Data>
    <wps:ComplexData mimeType="application/json">
    {"type": "MultiPolygon", "coordinates": [[[[-105.998360,
    31.393818 ], [ -106.212753, 31.478128 ], [ -106.383041,
    31.733763 ], [ -106.538971, 31.786198 ], [ -106.614441,
    31.817728 ], [ -105.769730, 31.170780 ], [ -105.998360,
    31.393818 ]]], [[[-94.913429, 29.257572 ], [ -94.767380,
    29.342451 ], [ -94.748405, 29.319490 ], [ -95.105415,
    29.096958 ], [ -94.913429, 29.257572 ]]]]}
    </wps:ComplexData>
    </wps:Data>
  </wps:Input>
</wps:DataInputs>
<wps:ResponseForm>
<wps:ResponseDocument>
<wps:Output>
<ows:Identifier>Result</ows:Identifier>
<ows:Title>Area serviced by playground.</ows:Title>
<ows:Abstract>Area within which most users of this playground will live.
</ows:Abstract>
```

3. 適応分野と展開

今回紹介した ZOO の機能は GIS データの実装を例にしている。ZOO のサーバー側処理の実装は開発者に任されており、GIS データ処理に限らずアイディア次第で様々なシステムに応用が可能である。

ZOO WPS を使用した応用例は、ZOO Project のホームページ[3]に掲載されているデモページ（ZOO Demos）から確認することができる。

3. 1 地理情報システム分野

既存の GIS サーバーを利用した地理情報解析処理や自社開発のデータ処理を、ZOO WPS により、国際規格に準拠したサービスとしてネットワークへ提供することが出来る。

既に特殊なデータ解析プログラムを持っており、利用者のアプリケーションから、共通の手順でデータ解析結果を供給することで、基礎データとロジックの流出を抑えてサービス事業を展開したいというような場合、特に有効なビジネスモデルを構築することが出来る。

3．2　産業分野

OGC で規格化されている「Sensor Observation Service (SOS) Interface Standard」に基づいた、センサー機器と共に利用することで、ZOO WPS をセンサーデータの収集とデータマネジメントに利用することが出来る。SOS は規格上、毎秒 1 万点というような膨大なシーケンスデータのリアルタイム処理には向かないが、環境センサーからの定時的なデータ収集や産業センサーからの状態取得などモニタリング系の処理であれば、ZOO は十分能力を発揮できる。

3．3　通信分野

通信分野での ZOO の応用では、携帯端末を利用したデータ解析サービスなどの実装が可能である。携帯端末では通話やアクセサリなどのユーザーサービスが主に実装されているが、大量のデータ解析と CPU 能力を必要とするアプリケーションを端末内に格納するのは困難である。共通化された通信手順でこうした解析データを端末が受け取ることができれば、アプリケーションを比較的自由に構築することが出来る。

ZOO の連携プロジェクトとして、goGPS Project[8] がある。goGPS は 1 周波型の RTK 測位や、リアルタイム補正処理を行えるオープンソースプロダクトである。goGPS と ZOO の連携では、携帯電話や端末装置などの安価な GPS モジュールでも高価な GPS と同様の高精度位置情報を取得することが可能になる。

4．おわりに

応用技術株式会社は、産業・社会・エンジニアリングの3つの軸を持つ会社で、様々な分野でのアプリケーション構築とデータ解析を経験として蓄積している。ZOO の持つウェブプロセシングサービスのポテンシャルは、応用技術株式会社のこれまでの経験を生かしたビジネスを、さらに展開する機会を与えてくれている。ZOO を利用することで、蓄積されたナレッジまたは技術をインターネットにサービスすることが出来るようになり、新しいビジネスの芽が生まれる。

この機会に是非、ZOO を利用したクラウドシステム構築をご検討ください。

＜参考文献＞

1)「Practical introduction to ZOO：The Open WPS Platform」（FOSS4G2010、2010.09、GeoLabs）

＜参考アドレス＞

[1] http://www.opengeospatial.org/standards/wps
[2] http://www.opengeospatial.org/
[3] http://www.zoo-project.org/
[4] http://www.osgeo.org/
[5] http://live.osgeo.org/
[6] zoo-discuss@gisws.media.osaka-cu.ac.jp
[7] #zoo_project@irc.freenode.net
[8] http://www.gogps-project.org/