Java EE 7 アプリケーション設計ガイド
- Java API for JSON Processing編



日本アイ・ビー・エム　システムズ・エンジニアリング株式会社

© 2015 IBM Corporation

Disclaimer

- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、この資料の内容は2015年3月現在の情報であり、製品の新しいリリース、PTFなどによって動作、仕様が変わる可能性があるのでご注意下さい。
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。
- ＩＢＭ、ＩＢＭロゴおよびibm.comは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。他の製品名およびサービス名等は、それぞれＩＢＭまたは各社の商標である場合があります。現時点でのＩＢＭの商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。
- 当資料をコピー等で複製することは、日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム　システムズ・エンジニアリング株式会社の承諾なしではできません。
- 当資料に記載された製品名または会社名はそれぞれの各社の商標または登録商標です。
- JavaおよびすべてのJava関連の商標およびロゴは Oracleやその関連会社の米国およびその他の国における商標または登録商標です。
- Microsoft, Windows および Windowsロゴは、Microsoft Corporationの米国およびその他の国における商標です。
- Linuxは、Linus Torvaldsの米国およびその他の国における登録商標です。
- UNIXはThe Open Groupの米国およびその他の国における登録商標です。

© 2015 IBM Corporation

目次

1. はじめに
2. Java API for JSON Processing 1.0
   - Object Model API
   - Streaming API
3. Hint&Tips
4. 参考資料

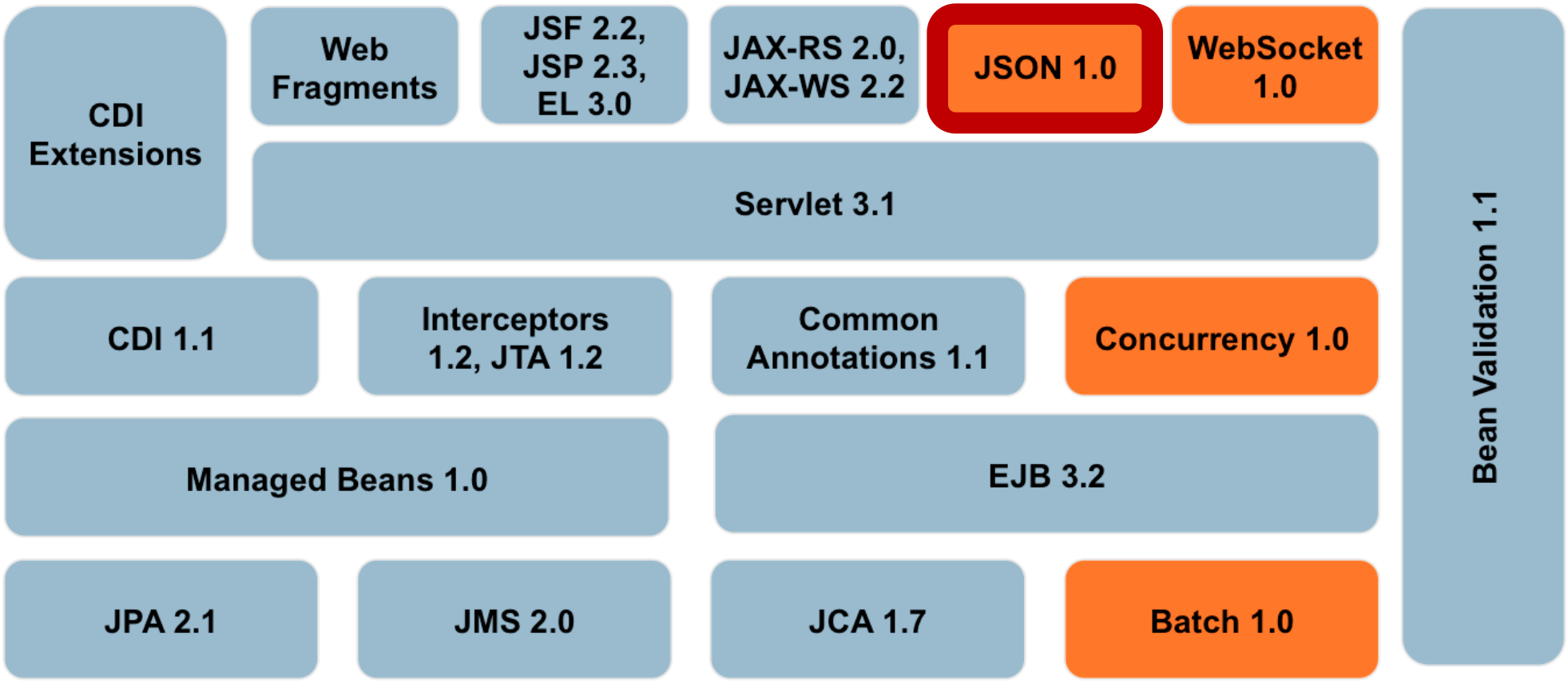
© 2015 IBM Corporation

1. はじめに

© 2015 IBM Corporation

Java EE 7 に含まれる仕様 (JSR一覧)

- Java EE 7では以下の仕様が含まれており、４つの仕様が新規に追加されている
- 当ガイドではJava EE 7で新規追加されたJava API for JSON Processing 1.0を対象とする



© 2015 IBM Corporation

WebSphere Application Serverのサポート状況 （2015年3月現在）

- Liberty Profile
  - –Java API for JSON Processing 1.0をサポートしている（WAS V8.5.5.4～）
  - –その他のJava EE 7の仕様の一部をサポートしはじめている
    - Servlet 3.1/WebSocket 1.0/Concurrency Utilities for Java EE 1.0
    - 詳細は下記URLを参照
    - http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/rwlp_feat.html?cp=SSAW57_8.5.5%2F1-0-2-2-0

- Full Profile（従来のWebSphere Application Server）
  - –2015年2月に開発意向表明のあったJava EE 7対応版のFullプロファイル（出荷時期未定）で対応予定

© 2015 IBM Corporation

JSONとは

- JSON (JavaScript Object Notation)
  - データとその構造をあらわす表記法。データ・フォーマットの一つ
  - JavaScriptのオブジェクト表記法であり、JavaScriptとの親和性が高い
  - シンプル、テキストベース
  - Ajax環境で、XMLの代替として、非常に多く使用される
  - 2006年に、RFC 4627 (http://www.rfc-editor.org/rfc/rfc4627.txt) として公開されている
  - JSONの構文
    - JSONのデータ構造はオブジェクトと配列の２つだけ定義されている
      - { } オブジェクト　{"key":"value"}
      - [ ] 配列
    - 7つのvalue typeが定義されている
      - String, number, object, array, true, false, null

```
{
   "firstName": "John", "lastName": "Smith", "age": 25,
   "address" : {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021"
   },
   "phoneNumber": [
      { "type": "home", "number": "212 555-1234" },
      { "type": "fax", "number": "646 555-4567" }
   ]
}
```


© 2015 IBM Corporation

2. Java API for JSON Processing 1.0

© 2015 IBM Corporation

Java API for JSON Processing 1.0 概要

- Java API for JSON Processing(JSON-P)とは
  - JSONデータの解析、生成、変換、照会を行うためのAPIを提供する
- 仕様
  - JSR 353で仕様化されており、プログラミングモデルとして下記 2 つのモデルが定義されている
  - Object Model (javax.jsonパッケージ)
    - メモリ上にJSONデータを表すツリー構造のオブジェクトを作成する
    - XMLのDocument Object Model(DOM) APIに類似したAPI
  - Streaming Model (javax.json.streamパッケージ)
    - イベントベースで1度に1エレメントずつJSONデータを解析する
    - 1度に1エレメントずつストリームへ書き込む
    - XMLのStreaming API for XML(StAX)に類似したAPI

© 2015 IBM Corporation

LibertyでJSON-P 1.0を使うには



- 前提条件
  - WAS V8.5.5 FP4(8.5.5.4)以降を利用する
  - JSON-P 1.0を使用する場合は、Java SE 7 以降が前提
  - 個別にJSON-Pをインストールすることも可能
    - <Install_Root>/bin/featureManager install jsonp-1.0 --when-file-exists=ignore
    - jsonp-1.0がインストールされているかどうかは下記コマンドで確認できる
    - <Install_Root>/bin/productInfo featureInfo

- Libertyに必要な構成
  - server.xml内のfeatureManagerにjsonp-1.0を追加する

```
<featureManager>
  <feature>jsonp-1.0</feature>
</featureManager>
```


© 2015 IBM Corporation

2. Java API for JSON Processing 1.0
- Object Model API
- Streaming API

© 2015 IBM Corporation

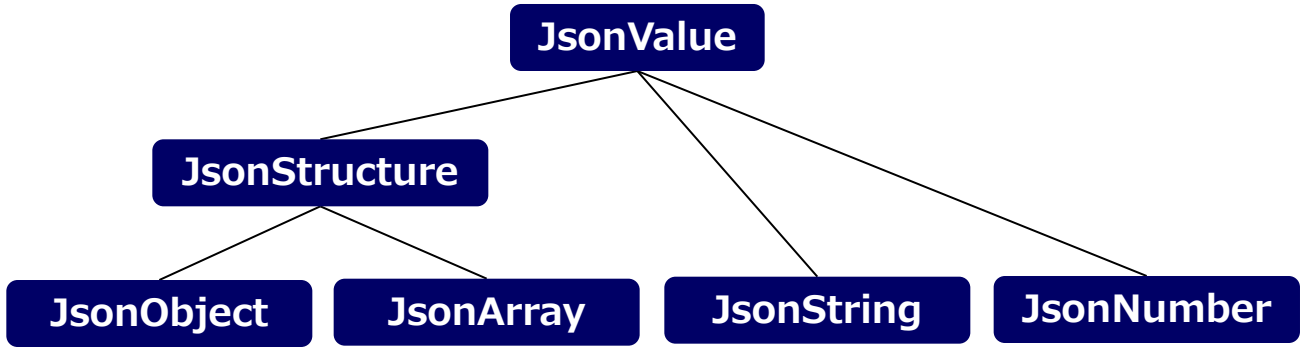
Object Model APIにおけるJSONデータの表現について

- JSONデータを表現するインターフェースは以下のとおり

インターフェース	説明
JsonValue	JSONデータのエレメントを表現する。JsonStructure,JsonString,JsonNumberのスーパータイプ。
JsonStructure	JsonValueのサブタイプ。JSONデータのオブジェクトあるいは配列を表す。
JsonObject JsonArray	JsonStructureのサブタイプ。 JsonObjectはオブジェクト、JsonArrayは配列を表す。
JsonString JsonNumber	JsonValueのサブタイプ。 JsonStringは文字列、JsonNumberは数値を表す。

- EnumとしてJsonValueのタイプが定義されている

Enum	説明
JsonValue.ValueType	JsonValueオブジェクトのタイプを示す。 • OBJECT • ARRAY • STRING • NUMBER • TRUE • FALSE • NULL



© 2015 IBM Corporation

Object Modelの作成やストリームへの読み取り/書き込みのためのAPI

- ストリームからの読み取り、Object Modelの作成、ストリームへの書き込みに関するクラスおよびインターフェース

API	説明
Json	JSON Processingのオブジェクトを作成するためのファクトリークラス。 createReaderやcreateWriter、createObjectBuilderなどの静的メソッドが定義されている。
JsonReader	ストリームからJSONデータを読み込み、メモリ上にObject Modelを作成する。
JsonWriter	メモリ上のObject Modelをストリームに書き出す。
JsonObjectBuilder	アプリケーションコードからエレメントを追加していくことでJsonObjectを作成する。
JsonArrayBuilder	アプリケーションコードからエレメントを追加していくことでJsonArrayを作成する。

- 例外に関するインターフェース

API	説明
JsonException	JSON処理中に例外が発生したことを示す。

© 2015 IBM Corporation

Object Model API：JSONデータを読み込んでObject Modelを作成する

- テキストファイル内のJSONデータを読み込み、メモリ上にObject Modelを作成する
  - JsonReaderインターフェースを利用する

```
import java.io.FileReader;
import javax.json.Json;
import javax.json.JsonReader;
import javax.json.JsonStructure;
...
//テキストファイルからJSONデータを読み込むためにFileReaderクラスをインスタンス化する
//jsondata.txtにJSONデータが格納されている
FileReader fr = new FileReader("jsondata.txt");

//JsonReaderを作成する
JsonReader reader = Json.createReader(fr));

//JSONデータを読み込み、Object ModelであるJsonObjectあるいはJsonArrayを作成する
JsonStructure jsonst = reader.read();

//作成したJSONvalueをJSONテキストにして標準出力に出力する
System.out.println(jsonst.toString());
```

jsondata.txt

```
{
   "firstName": "John", "lastName": "Smith", "age": 25,
   "address" : {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021"
   },
   "phoneNumber": [
      { "type": "home", "number": "212 555-1234" },
      { "type": "fax", "number": "646 555-4567" }
   ]
}
```

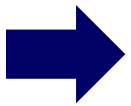

© 2015 IBM Corporation

Object Model API：アプリケーションコードでObject Modelを作成する

- アプリケーションコード上でObject Modelを作成することが可能
  - Json.createObjectBuilder()およびJson.createArrayBuilder()メソッドを使用してJsonObject/JsonArrayを作成できる
  - オブジェクトの参照"jsonobj"はツリーのトップを表しており、
    ツリー内をナビゲーションしたり、ストリームへ書き込むことができる

下記JSONデータをアプリケーションコードで作成したい場合

```
{
   "firstName": "John", "lastName": "Smith", "age": 25,
   "address" : {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021"
   },
   "phoneNumber": [
      { "type": "home", "number": "212 555-1234" },
      { "type": "fax", "number": "646 555-4567" }
   ]
}
```



```
import javax.json.Json;
import javax.json.JsonObject;
...
JsonObject jsonobj = Json.createObjectBuilder()
    .add("firstName", "John")
    .add("lastName", "Smith")
    .add("age", 25)
    .add("address", factory.createObjectBuilder()
        .add("streetAddress", "21 2nd Street")
        .add("city", "New York")
        .add("state", "NY")
        .add("postalCode", "10021"))
    .add("phoneNumber", factory.createArrayBuilder()
        .add(factory.createObjectBuilder()
            .add("type", "home")
            .add("number", "212 555-1234"))
        .add(factory.createObjectBuilder()
            .add("type", "fax")
            .add("number", "646 555-4567")))
    .build();
```


© 2015 IBM Corporation

Object Model API：Object Modelのツリー構造をナビゲーションする (1/2)

- メモリ上に作成されたObject Modelはツリー構造になっており、ツリー内の各要素にランダムアクセスが可能

```
public static void navigateTree(JsonValue tree, String key) {
    if (key != null)
        System.out.print("Key " + key + ": ");
    //JsonValueインターフェースのgetValueType()メソッドは
    //JsonValue型(Enum JsonValue.ValueType)を戻す
    switch(tree.getValueType()) {
        case OBJECT:
            System.out.println("OBJECT");
            JsonObject object = (JsonObject) tree;
            //java.util.Mapインターフェースから継承されたKeySet()メソッドで
            //JSONオブジェクトのkeyのセットを取得する
            for (String name : object.keySet())
                //get()メソッドでJSONオブジェクトのkeyの名前を取得する
                navigateTree(object.get(name), name);
            break;
        case ARRAY:
            System.out.println("ARRAY");
            JsonArray array = (JsonArray) tree;
            for (JsonValue val : array)
                navigateTree(val, null);
            break;
            //つづく→
```

```
            // ←つづき
        case STRING:
            JsonString st = (JsonString) tree;
            System.out.println("STRING " + st.getString());
                break;
        case NUMBER:
            JsonNumber num = (JsonNumber) tree;
            System.out.println("NUMBER " + num.toString());
            break;
        case TRUE:
        case FALSE:
        case NULL:
            System.out.println(tree.getValueType().toString());
            break;
    }
}
```


© 2015 IBM Corporation

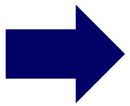
Object Model API：Object Modelのツリー構造をナビゲーションする (2/2)

- 前頁のnavigateTree()メソッドを実行した際の標準出力への出力結果は以下のとおり

```
//jsondata.txtからJSONデータからJsonValueオブジェクトを作成し、navigateTreeメソッドの引数に指定する
navigateTree((JsonValue)Json.createReader(new FileReader("jsondata.txt")).read(),null);
```

jsondata.txt

```
{
   "firstName": "John", "lastName": "Smith", "age": 25,
   "address" : {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021"
   },
   "phoneNumber": [
      { "type": "home", "number": "212 555-1234" },
      { "type": "fax", "number": "646 555-4567" }
   ]
}
```



標準出力

```
OBJECT
Key firstName: STRING John
Key lastName: STRING Smith
Key age: NUMBER 25
Key address: OBJECT
Key streetAddress: STRING 21 2nd Street
Key city: STRING New York
Key state: STRING NY
Key postalCode: STRING 10021
Key phoneNumber: ARRAY
OBJECT
Key type: STRING home
Key number: STRING 212 555-1234
OBJECT
Key type: STRING fax
Key number: STRING 646 555-4567
```


© 2015 IBM Corporation

Object Model API：Object Modelをストリームへ書き込む

- メモリ上に作成したObject Modelをストリームへ書き込む
  - JsonWriterインターフェースを利用する（アウトプットストリームをパラメーターとして指定する）
  - writeObject()メソッドでJsonObjectをストリームに書き込むことが可能

```
import javax.json.JsonObject;
import java.io.StringWriter;
import javax.json.JsonWriter;

JsonObject jsonobj = xxxxx;        //JsonObjectを作成
・・・・
StringWriter stWriter = new StringWriter();

JsonWriter jsonWriter = Json.createWriter(stWriter);

//JsonObjectをストリングストリームに書き込む
jsonWriter.writeObject(jsonobj);
jsonWriter.close();

String jsonData = stWriter.toString();
System.out.println(jsonData);
```

try-with-resources文を使用してJsonWriterを
自動的にクローズする場合のサンプルは以下のとおり

```
StringWriter stWriter = new StringWriter();

try (JsonWriter jsonWriter = Json.createWriter(stWriter)) {

    jsonWriter.writeObject(jsonobj);

}
```


© 2015 IBM Corporation

2. Java API for JSON Processing 1.0

- Object Model API
- Streaming API

© 2015 IBM Corporation

Streaming APIで利用する主なAPI

- インターフェース

API	説明
JsonParser	ストリームやObject ModelからJSONデータを読むことができるイベントベースのParserを表す
JsonGenerator	一度にひとつのエレメントをストリームに書き出す

- Enum

API	説明
JsonParser.Event	JsonParserのイベント。以下10個のイベントが定義されている START_OBJECT/START_ARRAY/END_OBJECT/END_ARRAY/KEY_NAME/VALUE_STRING/ VALUE_NUMBER/VALUE_TRUE/VALUE_FALSE/VALUE_NULL

- 例外

API	説明
JsonGenerationException	不適当なJSONが生成されたことを示す
JsonParsingException	不適当なJSONを解析したことを示す

© 2015 IBM Corporation

Streaming API：JsonParserによるJSONデータの解析

- Streaming APIはJSONテキストを解析する最も効率的なアプローチ
- イベントベースのJsonParserによってストリームおよびObject ModelからJSONデータを解析することができる

```
JsonParser parser = Json.createParser(new StringReader(jsondata));  //JsonParserインスタンスを取得
while (parser.hasNext()){                                  //parser.hasNext()メソッドで繰り返し処理を行う
   JsonParser.Event event = parser.next();          //次の解析のためのイベントを返す
   switch(event){
      case START_ARRAY:
      case END_ARRAY:
      case START_OBJECT:
      case END_OBJECT:
      case VALUE_FALSE:
      case VALUE_NULL:
      case VALUE_TRUE:
         //イベント名を返す
         System.out.println(event.toString());
         break;
      case KEY_NAME:
         //KEY_NAMEの名前を返す
         System.out.print(event.toString() + "  " + parser.getString() + " - ");
         break;
      case VALUE_STRING:
      case VALUE_NUMBER:
         //string値あるいはnumber値を返す
         System.out.println(event.toString() + "  " + parser.getString());
         break;
      }
```

```
START_OBJECT
KEY_NAME firstName - VALUE_STRING John
KEY_NAME lastName - VALUE_STRING Smith
KEY_NAME age - VALUE_NUMBER 25
KEY_NAME address - START_OBJECT
KEY_NAME streetAddress - VALUE_STRING 21 2nd Street
KEY_NAME city - VALUE_STRING New York
KEY_NAME state - VALUE_STRING NY
KEY_NAME postalCode - VALUE_STRING 10021
END_OBJECT
KEY_NAME phoneNumber - START_ARRAY
START_OBJECT
KEY_NAME type - VALUE_STRING home
KEY_NAME number - VALUE_STRING 212 555-1234
END_OBJECT
START_OBJECT
KEY_NAME type - VALUE_STRING fax
KEY_NAME number - VALUE_STRING 646 555-4567
END_OBJECT
END_ARRAY
END_OBJECT
```


© 2015 IBM Corporation

Streaming API：JsonGeneratorによるJSONデータの生成

- JsonGeneratorを使用してJSONデータを
  ストリームに書き込むことかできる
  - 1度に1要素ずつ書き込む
  - writeStartObject() ・・・Objectを開始する文字を書き込む
  - writeStartArray() ・・・配列を開始する文字を書き込む
  - write() ・・・現在のオブジェクトあるいは配列コンテキストに書き込む
  - writeEnd() ・・・現在のコンテキストの終了を書き込む

  - 詳細はAPIドキュメントを参照
  - http://docs.oracle.com/javaee/7/api/javax/json/stream/JsonGenerator.html

```
FileWriter writer = new FileWriter("test.txt");
JsonGenerator gen = Json.createGenerator(writer);
gen.writeStartObject()
    .write("firstName", "John")
    .write("lastName", "Smith")
    .write("age", 25)
    .write("streetAddress", "21 2nd Street")
    .write("city", "New York")
    .write("state", "NY")
    .write("postalCode", "10021")
    .writeStartArray("phoneNumbers")
        .writeStartObject()
            .write("type", "home")
            .write("number", "212 555-1234")
        .writeEnd()
        .writeStartObject()
            .write("type", "fax")
            .write("number", " 646 555-4567")
        .writeEnd()
    .writeEnd()
.writeEnd();
gen.close();
```


© 2015 IBM Corporation

Hint&Tips

© 2015 IBM Corporation

Object Model APIとStreaming APIの比較

- JSON-Pで提供されているObject Model APIとStreaming APIを比較すると以下のような特徴がある

比較軸	Object Model API	Streaming API
柔軟性	○	-
効率性	-	○
ストリームからの読み込み/書き込み	オブジェクト単位	1 度に1エレメントずつ
コードの記述量	少ない	多い
処理速度	若干遅い	若干早い
メモリ使用量	若干多い	若干少ない
XMLの類似API	Document Object Model(DOM) API	Streaming API for XML(StAX)
用途	ツリー内のコンテンツにランダムアクセスが必要な処理に適している	エレメントの処理に残りのデータを必要としないようなローカル処理に適している

© 2015 IBM Corporation

Java EE RESTful Webサービス(JAX-RS)との関係

- JAX-RSのリソース・メソッドでJSON-Pを利用することが可能
- Java EE 7ではJSONデータをJavaオブジェクトにバインディングする際にはJAXBが利用される
- JSON-PはJSON Bindingをサポートしていない
  - 今後、JAXB for XMLに類似したJSRとしてJSON Bindingが検討されている

- 詳細は「Java EE 7 アプリケーション設計ガイド –JAX-RS 2.0編」を参照

© 2015 IBM Corporation

WebSocketでJSON-Pを使う

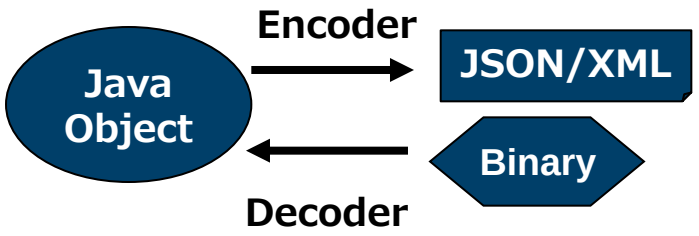
- JSON形式でWebSocketメッセージの送受信を行う場合
  EncoderおよびDecoderクラスでJavaオブジェクトとJSONデータを変換する際にJSON-Pを利用することができる

Encoderクラス

```
public class EncoderOne  implements Encoder.Text<Message> {
    @Override
    public String encode(Message msg) throws EncodeException {
        JsonObject jsonObj = Json.createObjectBuilder()
                                 .add("subject", msg.getSubject())
                                 .add("content", msg.getContent()).build();
        return jsonObj.toString();
    }
    ....
}
```

Decoderクラス

```
public class DecoderOne implements Decoder.Text<Message> {
    @Override
    public Message decode(String s) throws DecodeException {
        JsonObject jsonObj = Json.createReader(new StringReader(s)).readObject();
        Message msg = new Message();
        msg.setSubject(jsonObj.getString("subject"));
        msg.setContent(jsonObj.getString("content"));
        return msg;
    }
    .....
}
```




```
public class Message {
  private String subject;
  private String content;
  public String getSubject() {
    return subject;
  }
  public void setSubject(String subject) {
    this.subject = subject;
  }
  public String getContent() {
    return content;
  }
  public void setContent(String content) {
    this.content = content;
  }
}
```


© 2015 IBM Corporation

参考資料

- Java™ EE 7 Specification APIs
- http://docs.oracle.com/javaee/7/api/
- The Java EE 7 Tutorial
- http://docs.oracle.com/javaee/7/tutorial/doc/home.htm
- Knowlede Center - WebSphere Application Server Liberty Core 8.5.5
- http://www-01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/as_ditamaps/was855_welcome_liberty_core.html
- WASDev
- https://developer.ibm.com/wasdev/downloads/

© 2015 IBM Corporation