XML以外のオブジェクト直列化技法の紹介

中部大学　経営情報学部
経営情報学科　前田 和昭
kaz@acm.org
kmaeda@gmail.com

感謝

ここで話ができるのは皆様のおかげです

- 小林先生
- Seagaia Meetingの皆様
- 中部大学で応援していくれる皆様
- 独立行政法人科学技術振興機構(JST)の関係者様

ありがとうございます

内容

本講演では，Javaから利用可能なオブジェクト直列化の技法（XML以外がメイン）をいくつか取り上げ，定量的・定性的な特徴について述べます．取り上げる手法は，XStream, Protocol Buffers, Apache Avro, Thrift, Jackson, JsonLibなどです．

また，別のアプローチとして，講演者が独自に設計した構造化データ表現RibONについて述べます．
（あまり時間がないかも）

アジェンダ

1. 自己紹介
2. XMLへの想い
3. Starbucks大好き
4. オブジェクト直列化...XML, JSONなど
5. 自分オリジナルな話を少しだけ
   (時間があれば...)

自己紹介

自己紹介

教育（文系：経営情報学科）

- Java プログラミング, ソフトウェア開発

研究

- Webアプリケーション, XML関連技術
- コンパイラ関連

開発（商用製品開発）

- フロントエンド: VHDL, Java, SQL, C#, VisualBasicなど
- 実装: ??Kライン/1年
  - 逆エンジニアリングツール
  - Javaコンパイラフロントエンド
  - ダイアグラムエディタのためのライブラリ

自己紹介

昔々:

- 学生だった頃: BSD Unix, GCC on VAX
  - GCCの最初のリリースは,
    1987年3 月22日だったらしい（26年前）
- 開発
  - 1987〜2010：クローズドな商用ソフトウェア
  - SUN, NeXT, PowerBook, MacBook
- 2011年の大震災
  - オープンソースで世の中に何か貢献したい！

www.curlos.org

XMLへの想い

XML Everywhere

XML is a buzzword you will see everywhere on the Internet, but it's also a rapidly maturing technology with powerful real-world applications, particularly for the management, display, and organization of data.

- D. Hunter, J. Rafter, et al.,
  Beginning XML, 4th ed., Wiley, 2007.

どこでもXML．多数のアプリケーションへ

XML

XMLは多数のアプリケーションに広がる

XMLには,

「良いところ」と「不満なところ」

XMLの良いところ

プラットフォーム独立

- プログラミング言語
- オペーレーティングシステム
- コンピュータ

標準化

豊富なライブラリ

- DOM, SAX, XPath, XSLT, SOAP など

XMLへの７つの不満

1. 醜いタグが一杯
2. 人が読めるけど難読
3. 要素中心か属性中心．どちらが良いか困惑
4. 仕様書が膨大で学習が困難

   XML Schema, XSLT, XML Query, WS-*
5. その仕様書からの実装が困難
6. 名前空間でXML文書は複雑化
7. DOM APIとアプリケーションAPIのミスマッチ

XML以外の直列化を試したくなった

オブジェクト直列化の話

自宅近くでOrder

STARBUCKS®

アピタ高蔵寺店
#614　TEL 0568-94-5831

1 G コーヒー フラペチーノ	460
ショット	50
ホイップ	50
ヘーゼルナッツシロップ	50
キャラメルソース	0
チョコレートソース	0
チョコレートチップ	50
合計(1点)	660
(内消費税	31)
現金	1,000
(内消費税等	31)
お釣り	340

020357503302　200749　2011/09/12 20:22:02

自宅近くでOrder

Order

- Grande
- Coffee Frappuccino
- September 12, 2011,460 yen

Options

- Shot, 50 yen
- Extra whip, 50 yen
- Hazelnut, 50 yen
- Caramel sauce, 0 yen
- Chocolate sauce, 0 yen
- Chocolate chip, 50 yen

Starbucks

構造化データ (Structured Data) の例

XMLでStarbucks Order

```
<order>
   <size>Grande</size>
   <product>Coffee Frappuccino</product>
   <date>September 12, 2011</date>
   <price>460</price>
   <options>
      <option price="50">Shot</option>
      <option price="50">Extra whip</option>
      <option price="50">Hazelnut</option>
      <option price="0">Caramel sauce</option>
      <option price="0">Chocolate sauce</option>
      <option price="50">Chocolate chip</option>
   </options>
</order>-
```

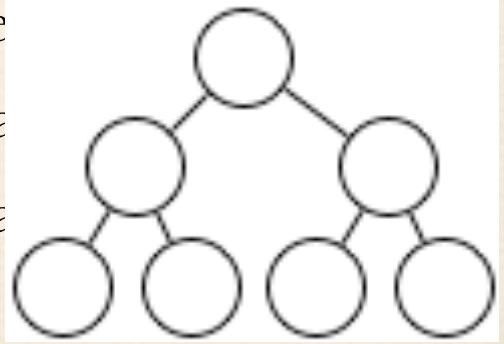
構造化データ

Order

- Grande
- Coffee Frappuccino
- September 12, 2011, 460 yen

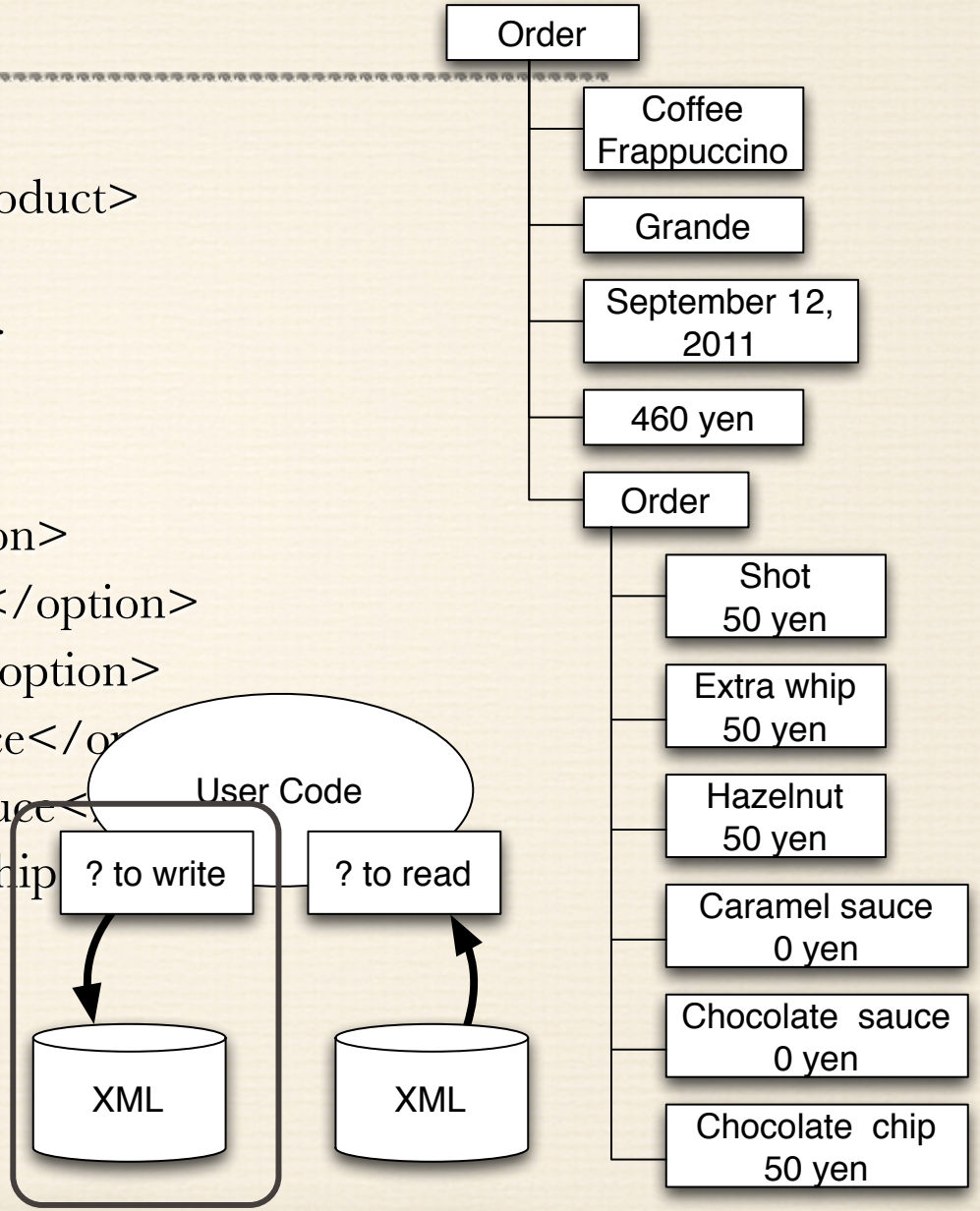
Options

- Shot, 50 yen
- Extra whip, 50 yen
- Hazelnut, 50 yen
- Carame
- Chocola
- Chocola



XMLでStarbucks Order

```
<order>
   <product>Coffee Frappuccino</product>
   <size>Grande</size>
   <date>September 12, 2011</date>
   <price>460</price>
   <options>
      <option price="50">Shot</option>
      <option price="50">Extra whip</option>
      <option price="50">Hazelnut</option>
      <option price="0">Caramel sauce</or
      <option price="0">Chocolate sauce<
      <option price="50">Chocolate chip
   </options>
</order>
```



直列化

Starbucks Order (修正版)

```
<order>
  <size>Grande</size>
  <product>Coffee Frappuccino</product>
  <date>March 11, 2013</date>
  <price>3.14</price>
  <currency>JPY</currency>
  <count>6</count>
  <options>
    <option price="50">Shot</option>
    <option price="50">Extra whip</option>
    .....
  </options>
</order>
```

```
-rw-r--r--  1 kaz  staff   547  3 18 13:14 hand.xml
```

オブジェクト直列化

XML, JSONなど

1. 直列化 - Java

```
public class Order implements Serializable{
    CupSize size;
    String product;
    String date;
    double  price;
    String  currency;
    int         count;
    List<Option>   options;
    .....getter and setter.....
}
```

```
public enum CupSize {
    Small, Tall, Grande, Venti;
}
```

スキーマ定義は別に必要なし

直列化 - Java

```
FileOutputStream output
      = new FileOutputStream("starbucks-java.dat");

ObjectOutputStream os
      = new ObjectOutputStream(output);

os.writeObject(thisOrder);

os.close();
```

ObjectOutputStream はリフレクションを使用

バイナリで直列化 - Java

```
% hexdump -C starbucks-java.dat | head -11
00000000  ac ed 00 05 73 72 00 18  6f 72 67 2e 63 75 72 6c  |....sr..org.curl|
00000010  6f 73 2e 6a 61 76 61 6f  62 6a 2e 4f 72 64 65 72  |os.javaobj.Order|
00000020  3e 5b 27 ec dc 3b 4c 03  02 00 07 49 00 05 63 6f  |>['..;L....I..co|
00000030  75 6e 74 44 00 05 70 72  69 63 65 4c 00 08 63 75  |untD..priceL..cu|
00000040  72 72 65 6e 63 79 74 00  12 4c 6a 61 76 61 2f 6c  |rrencyt..Ljava/l|
00000050  61 6e 67 2f 53 74 72 69  6e 67 3b 4c 00 04 64 61  |ang/String;L..da|
00000060  74 65 71 00 7e 00 01 4c  00 07 6f 70 74 69 6f 6e  |teq.~..L..option|
00000070  73 74 00 10 4c 6a 61 76  61 2f 75 74 69 6c 2f 4c  |st..Ljava/util/L|
00000080  69 73 74 3b 4c 00 07 70  72 6f 64 75 63 74 71 00  |ist;L..productq.|
00000090  7e 00 01 4c 00 04 73 69  7a 65 74 00 1c 4c 6f 72  |~..L..sizet..Lor|
000000a0  67 2f 63 75 72 6c 6f 73  2f 6a 61 76 61 6f 62 6a  |g/curlos/javaobj|
```

-rw-r--r-- 1 kaz staff **575** 3 18 13:17 starbucks-java.dat

-rw-r--r-- 1 kaz staff **547** 3 18 13:14 hand.xml

```
final static short STREAM_MAGIC = (short)0xaced;
final static short STREAM_VERSION = 5;
final static byte TC_NULL = (byte)0x70;
final static byte TC_REFERENCE = (byte)0x71;
final static byte TC_CLASSDESC = (byte)0x72;
final static byte TC_OBJECT = (byte)0x73;
```

Hexdumpは面白い

```
% cat Hello.java
public class Hello {
    public static void main(String[] args){
        System.out.println("Hello");
    }
}

% hexdump -C Hello.class | head -11
00000000  ca fe ba be 00 00 00 32  00 1c 0a 00 06 00 0f 09  |.......2........|
00000010  00 10 00 11 08 00 12 0a  00 13 00 14 07 00 12 07  |................|
00000020  00 15 01 00 06 3c 69 6e  69 74 3e 01 00 03 28 29  |.....<init>...()|
00000030  56 01 00 04 43 6f 64 65  01 00 0f 4c 69 6e 65 4e  |V...Code...LineN|
00000040  75 6d 62 65 72 54 61 62  6c 65 01 00 04 6d 61 69  |umberTable...mai|
00000050  6e 01 00 16 28 5b 4c 6a  61 76 61 2f 6c 61 6e 67  |n...([Ljava/lang|
00000060  2f 53 74 72 69 6e 67 3b  29 56 01 00 0a 53 6f 75  |/String;)V...Sou|
00000070  72 63 65 46 69 6c 65 01  00 0a 48 65 6c 6c 6f 2e  |rceFile...Hello.|
00000080  6a 61 76 61 0c 00 07 00  08 07 00 16 0c 00 17 00  |java............|
00000090  18 01 00 05 48 65 6c 6c  6f 07 00 19 0c 00 1a 00  |....Hello.......|
000000a0  1b 01 00 10 6a 61 76 61  2f 6c 61 6e 67 2f 4f 62  |....java/lang/Ob|
```

直列化データの比較

サイズ

- 手入力のXML: 547 bytes
- Java 直列化: 575 bytes

特徴の比較

	スキーマは必要？	コンパイラは必要？	アスキーかバイナリか	サポートする言語
Java直列化	No	No	バイナリ	Java

2. XStream

XMLによる直列化のためのライブラリ

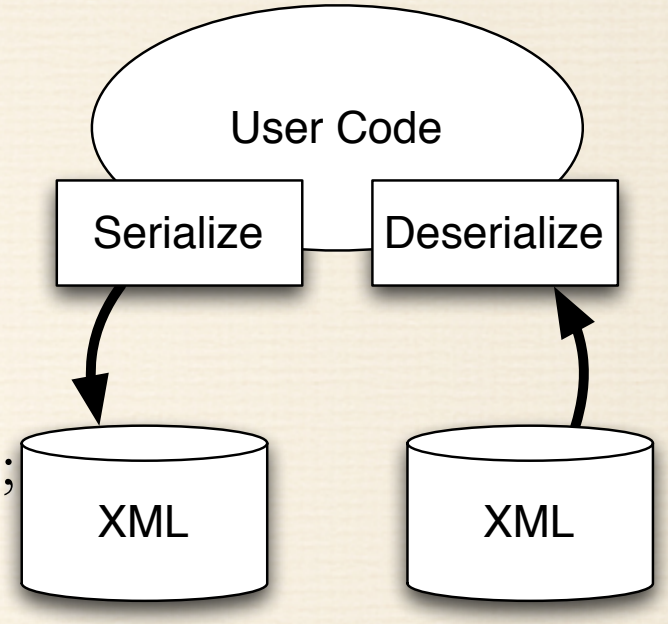
- http://xstream.codehaus.org/ (ver. 1.4.4)

特徴

- 使いやすい
- 性能が良い？
- すっきりしたXML
- 何も修正しなくて良い
- グラフ構造でも大丈夫

XStream

```
Order thisOrder = new Order();
thisOrder.setSize(CupSize.Grande);
thisOrder.setProduct("Coffee Frappuccino");
thisOrder.setDate("March 11,2013");
thisOrder.setPrice(3.14);
thisOrder.setOptions(new ArrayList<Option>());
.....
```



Writer

```
FileOutputStream fos
      = new FileOutputStream("starbucks-xstream.xml");
XStream xstream = new XStream();
xstream.toXML(thisOrder, fos);
```

Reader

```
FileInputStream fis
      = new FileInputStream("starbucks-xstream.xml");
XStream xstream = new XStream();
Order thisOrder = (Order)xstream.fromXML(fis);
```

直列化データ - XStream (1)

```
<org.curlos.xstream.Order>
  <size>Grande</size>
  <product>Coffee Frappuccino</product>
  <date>March 11,2013</date>
  <price>3.14</price>
  <currency>JPY</currency>
  <count>6</count>
  <options>
    <org.curlos.xstream.Option>
      <name>Shot</name>
      <price>50</price>
    </org.curlos.xstream.Option>
```

```
-rw-r--r--  1 kaz  staff  954  3 18 14:08 starbucks-xstream.xml
```

直列化データ - XStream (2)

```
<order>
  <size>Grande</size>
  <product>Coffee Frappuccino</product>
  <date>March 11,2013</date>
  <price>3.14</price>
  <currency>JPY</currency>
  <count>6</count>
  <options>
    <option>
      <name>Shot</name>
      <price>50</price>
    </option>
```

```
FileOutputStream fos
      = new FileOutputStream("starbucks-xstream.xml");
XStream xstream = new XStream();
xstream.alias("order", Order.class);
xstream.alias("option", Option.class);
xstream.toXML(thisOrder, fos);
```

```
-rw-r--r--  1 kaz  staff  688  3 18 14:08 starbucks-xstream.xml
```

直列化データの比較

サイズ

- 手入力のXML: 547 bytes
- Java 直列化: 575 bytes
- XStream: 688 bytes

二つのXML

- 手入力のXML

```
<options>
  <option price="50">Shot</option>
  <option price="50">Extra whip</option>
```

- XStreamによるXML

```
<options>
  <option>
    <name>Shot</name>
    <price>50</price>
  </option>
  <option>
    <name>Extra whip</name>
    <price>50</price>
```

特徴の比較

	スキーマは必要か？	コンパイラは必要か	アスキーかバイナリか	サポートする言語
Java直列化	No	No	バイナリ	Java
XStream	No	No	アスキー (XML)	Java

3. IDL: Cで直列化

DIANA: 古い技術

- Descriptive Intermediate Attributed Notation for Ada
- ADAの意味を記述するための中間言語 (1980年代)
- DIANAは **IDL** で定義された

IDL

- コンパイラ内部フェーズ間のインタフェースを提供
  - フロントエンド (構文解析) と
  - ミドルエンド (意味解析)

スキーマ定義 - IDL

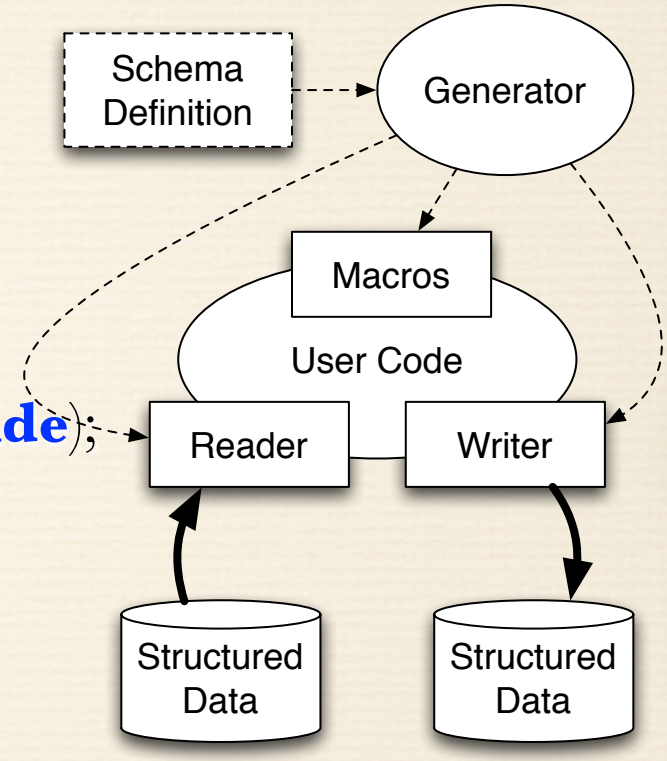
Starbucks スキーマ定義

```
Structure order Root Order Is
   Order   => size         : CupSize
              product      : String,
              date         : String,
              price        : Rational,
              currency     : String,
              count        : Integer,
              options      : Seq Of Option;
   For CupSize Use Representation Enumerated;
   CupSize ::= Small | Tall | Grande | Venti;
   Small =>; Tall =>; Grande =>; Venti => ;
   Option => name    : String,
             price   : Integer;
End
```

Reader/Writer in C - IDL

```
Order thisOrder;
SEQOption options;
Option    thisOption;
FILE *o_out;
thisOrder = NOrder; /* malloc */
thisOrder->size = GrandeToCupSize(NGrande);
thisOrder->product = "Coffee Frappuccino";
thisOrder->date="March 11,2013";
thisOrder->price=3.14;
```

スキーマコンパイラ



Writer

```
FILE *o_out=fopen("starbucks-idl.ascii","w");
order_out(o_out,thisOrder);
```

Reader

```
FILE *o_in=fopen("starbucks-idl.txt","r");
thisOrder = order_in(o_in);
```

アスキー - IDL

```
Order[size      Grande;
options <Option[name    "Shot";
price   50]
 Option[name    "Extra whip";
price   50]
 Option[name    "Hazelnut";
price   50]
 Option[name    "Caramel sauce";
price   0]
 Option[name    "Chocolate sauce";
price   0]
 Option[name    "Chocolate chip";
price   50]
>;
date    "March 11,2013";
price   3.14;
currency        "JPY";
count   6;
product "Coffee Frappuccino"]
```

-rw-r--r-- 1 kaz staff **364** 3 11 06:23 starbucks-idl.ascii

バイナリ - IDL

```
% hexdump -C starbucks-idl.binary
00000000  fe 00 00 00 00 80 03 06  03 4a 50 59 0d 4d 61 72  |.........JPY.Mar|
00000010  63 68 20 31 31 2c 32 30  31 33 06 80 02 04 53 68  |ch 11,2013....Sh|
00000020  6f 74 32 80 02 0a 45 78  74 72 61 20 77 68 69 70  |ot2...Extra whip|
00000030  32 80 02 08 48 61 7a 65  6c 6e 75 74 00 80 02 0d  |2...Hazelnut....|
00000040  43 61 72 61 6d 65 6c 20  73 61 75 63 65 00 80 02  |Caramel sauce...|
00000050  0f 43 68 6f 63 6f 6c 61  74 65 20 73 61 75 63 65  |.Chocolate sauce|
00000060  32 80 02 0e 43 68 6f 63  6f 6c 61 74 65 20 63 68  |2...Chocolate ch|
00000070  69 70 32 04 33 2e 31 34  12 43 6f 66 66 65 65 20  |ip2.3.14.Coffee |
00000080  46 72 61 70 70 75 63 63  69 6e 6f 80 01 fd        |Frappuccino...|
```

Begin=> fe , End=>fd

(Binary) -rw-r--r-- 1 kaz staff **142** 3 11 06:29 starbucks-idl.binary

(Ascii) -rw-r--r-- 1 kaz staff **364** 3 11 06:23 starbucks-idl.ascii

直列化データの比較

サイズ

- 手入力のXML: 547 bytes
- Java直列化: 575 bytes
- XStream: 688 bytes
- IDL (アスキー): 364 bytes
- IDL (バイナリ): 142 bytes

特徴の比較

	スキーマは必要か？	コンパイラは必要か？	アスキーかバイナリか	サポートする言語
Java直列化	No	No	バイナリ	Java
XStream	No	No	アスキー (XML)	Java
IDL	Yes	Yes	両方	C, Pascal

4. Protocol Buffers

Googleは，内部でRPCプロトコルとして使っているらしい

- http://code.google.com/p/protobuf/ (ver. 1.4.4)

柔軟で効率よく，構造化データを直列化するための自動化された機構を提供

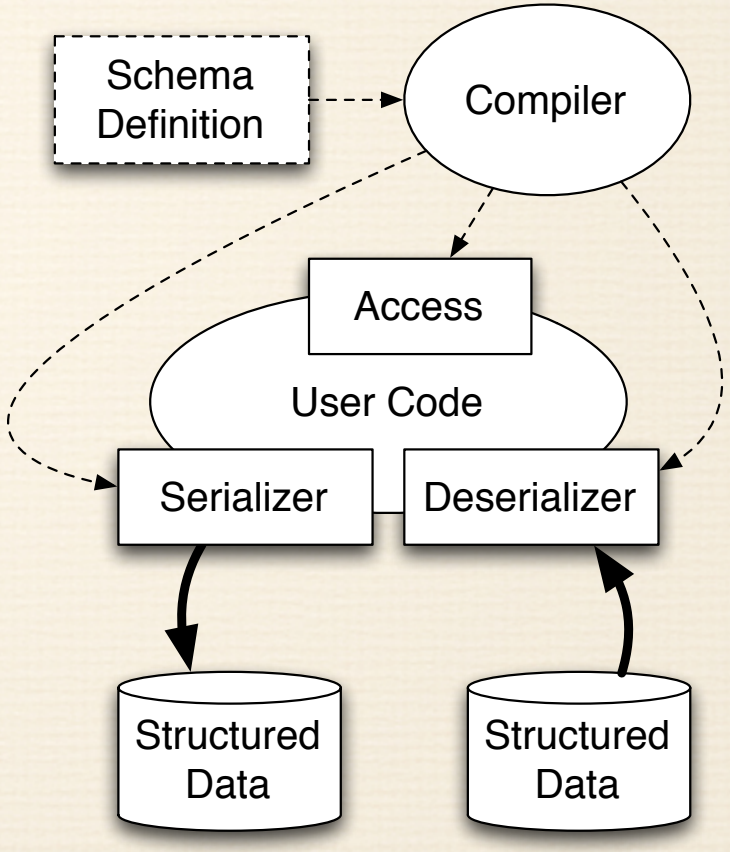
３言語をサポート：Java, C++, Python

Protocol bufferコンパイラ (protoc)

- メッセージ定義を読み，指定言語のクラスを生成

スキーマ定義 - Protocol Buffers

```
option java_package = "org.curlos.protobuf";
message Order {
  enum CupSize {  Small = 1; Tall = 2; Grande = 3; Venti = 4;  }
  required CupSize size = 1;
  required string product = 2;
  required string date = 3;
  required double  price = 4;
  required string currency = 5;
  required int32  count = 6;
  message Option {
    required string name = 1;
    required int32 price = 2;
  }
  repeated Option options = 7;
}
```



直列化 - Protocol Buffers

```
Order.Builder thisOrder = Order.newBuilder();
thisOrder.setSize(Order.CupSize.Grande);
thisOrder.setProduct("Coffee Frappuccino");
thisOrder.setDate("March 11,2013");
thisOrder.setPrice(3.14);
thisOrder.setCurrency("JPY");
thisOrder.setCount(6);
Order.Option.Builder thisOption = Order.Option.newBuilder();
thisOption.setName("Shot");    thisOption.setPrice(50);
thisOrder.addOptions(thisOption);
```

Writer

```
FileOutputStream fos
    = new FileOutputStream("starbucks-protobuf.dat");
thisOrder.build().writeTo(fos);
```

Reader

```
FileInputStream fis
    = new FileInputStream("starbucks-protobuf.dat");
Order thisOrder = Order.parseFrom(fis);
```

直列化データ - Protocol Buffers

```
00000000  08 03 12 12 43 6f 66 66  65 65 20 46 72 61 70 70  |....Coffee Frapp|
00000010  75 63 63 69 6e 6f 1a 0d  4d 61 72 63 68 20 31 31  |uccino..March 11|
00000020  2c 32 30 31 33 21 1f 85  eb 51 b8 1e 09 40 2a 03  |,2013!...Q...@*.|
00000030  4a 50 59 30 06 3a 08 0a  04 53 68 6f 74 10 32 3a  |JPY0.:...Shot.2:|
00000040  0e 0a 0a 45 78 74 72 61  20 77 68 69 70 10 32 3a  |...Extra whip.2:|
00000050  0c 0a 08 48 61 7a 65 6c  6e 75 74 10 00 3a 11 0a  |...Hazelnut..:..|
00000060  0d 43 61 72 61 6d 65 6c  20 73 61 75 63 65 10 00  |.Caramel sauce..|
00000070  3a 13 0a 0f 43 68 6f 63  6f 6c 61 74 65 20 73 61  |:...Chocolate sa|
00000080  75 63 65 10 32 3a 12 0a  0e 43 68 6f 63 6f 6c 61  |uce.2:...Chocola|
00000090  74 65 20 63 68 69 70 10  32                       |te chip.2|

-rw-r--r--   1 kaz  staff  153  3 19 18:32 starbucks-protobuf.dat
```

0Varintint32, int64, uint32, uint64, sint32, sint64, bool, enum

0x08 = 0000 1000 (00001=>tag1,
000=>type variant used for int32, int64, bool, enum, et al.)

0x12 = 0001 0010 (00010 => tag2, 010=>Length-delimited: string)

直列化データの比較

サイズ

- 手入力のXML: 547 bytes
- Java直列化: 575 bytes
- XStream: 688 bytes
- IDL (アスキー): 364 bytes
- IDL (バイナリ): 142 bytes
- Protocol Buffers: 153 bytes

Comparison of Features

	スキーマは必要か？	コンパイラは必要か？	アスキーかバイナリか	サポートする言語
Java直列化	No	No	バイナリ	Java
IDL	Yes	Yes	両方	C, Pascal
XStream	No	No	アスキー (XML)	Java
Protocol Buffer	Yes	Yes	バイナリ	Java, C++, Python

JSON

http://www.json.org/

Introducing JSON

العربية Български 中文 Český Dansk Nederlandse English Esperanto Française Deutsch Ελληνικά עברית Magyar Indonesia
Italiano 日本 한국어 فارسی Polski Português Română Русский Српски Slovenščina Español Svenska Türkçe Tiếng Việt

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

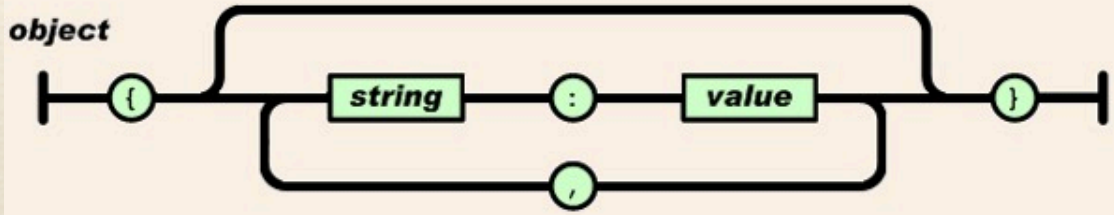
JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

In JSON, they take on these forms:

An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma).



object
{}
{ *members* }
members
pair
pair , *members*
pair
string : *value*
array
[]
[*elements*]
elements
value
value , *elements*
value
string
number
object
array
`true`
`false`
`null`

---

string

Yaccで書いた構文規則

```
object  : LCURLY membersList
                                RCURLY
        | LCURLY RCURLY   ;
membersList : pair
        | membersList COMMA pair    ;
pair    : STRING COLON value   ;
value   : STRING
        | INTEGER
        | FLOAT
        | object
        | array
        | TRUE
        | FALSE
        | NULL    ;
array   : LBRACE elementsList
                                RBRACE
        | LBRACE RBRACE    ;
elementsList : value
        | elementsList COMMA value   ;
```

www.curlos.org/slides/

Serialization Techniques:

- Standard Object Serialization in Java
- XStream - http://xstream.codehaus.org/
- Apache Avro - http://avro.apache.org/
- Thrift - http://thrift.apache.org/
- ProtoBuf - http://code.google.com/p/protobuf/

Libraries to serialize in JSON:

- Flexjson - http://flexjson.sourceforge.net/
- Gson - http://code.google.com/p/google-gson/
- Jackson - http://jackson.codehaus.org/
- Json-lib - http://json-lib.sourceforge.net/
- JsonMarshaller - http://code.google.com/p/jsonmarshaller/
- Json-Smart - http://code.google.com/p/json-smart/

Protostuff - http://code.google.com/p/protostuff/

(March 9, Friday, 2012)

Source code of yacc & lex to parse a JSON file are here. » json-src.zip

The code is written in Java, and it uses two code generators: byaccj and jflex.
If you need to generate source code from scan.l, you can get it with executing

jflex scan.l

If you generate source code from parse.y, you can get it with executing

byaccj -J -Jclass=JsonParser -Jpackage=json -vd parse.y

5. Apache Avro

http://avro.apache.org/ (ver.1.7.4)
特徴:

- 豊富なデータ構造
- コンパクトで，処理が速いバイナリ形式
- RPC (Remote Procedure Call)で使うことを考慮
- 動的言語での利用を考慮
- 直列化のためのコード生成は必要ない

Schema Definition - Avro

```
{
 "namespace": "org.curlos.avro",
 "name": "Order",
 "type": "record",
 "fields": [
   {"name": "size", "type": "int"},
   {"name": "product", "type": "string"},
   {"name": "date",   "type": "string"},
   {"name": "price",  "type": "double"},
   {"name": "currency","type": "string"},
   {"name": "count",   "type": "int"},
   {"name": "options","type": {
         "type": "array",
         "items": org.curlos.avro.Option
     }
   } ] }
```

```
{
 "namespace": "org.curlos.avro",
 "name": "Option",
 "type": "record",
 "fields": [
   {"name":"name","type":"string"},
   {"name":"price","type":"int"}
 ] }
```

直列化 - Avro

```
File fileOption = new File("schema-option.avro");
File fileOrder = new File("schema-order.avro");
Schema schemaOption = AvroUtils.parseSchema(fileOption);
Schema schemaOrder  = AvroUtils.parseSchema(fileOrder);
```

1

```
List<GenericRecord> array = new ArrayList<GenericRecord>();
GenericRecord thisOption = new GenericData.Record(schemaOption);

thisOption.put("name", "shot");
thisOption.put("price", 50);
array.add(thisOption);
```

2

```
FileOutputStream fos = new FileOutputStream("starbucks-avro.dat");
EncoderFactory factory = EncoderFactory.get();
Encoder encoder = factory.binaryEncoder(fos, null);
GenericDatumWriter<GenericRecord> writer =
        new GenericDatumWriter<GenericRecord>(schemaOrder);
writer.write(recordOrder, encoder);
```

3

Serialized Data - Avro

```
% hexdump -C starbucks-avro.dat
00000000  00 24 43 6f 66 66 65 65  20 46 72 61 70 70 75 63  |.$Coffee Frappuc|
00000010  63 69 6e 6f 1a 4d 61 72  63 68 20 31 31 2c 32 30  |cino.March 11,20|
00000020  31 33 1f 85 eb 51 b8 1e  09 40 06 4a 50 59 0c 0c  |13...Q...@.JPY..|
00000030  08 73 68 6f 74 64 14 45  78 74 72 61 20 77 68 69  |.shotd.Extra whi|
00000040  70 64 10 48 61 7a 65 6c  6e 75 74 00 1a 43 61 72  |pd.Hazelnut..Car|
00000050  61 6d 65 6c 20 73 61 75  63 65 00 1e 43 68 6f 63  |amel sauce..Choc|
00000060  6f 6c 61 74 65 20 73 61  75 63 65 64 1c 43 68 6f  |olate sauced.Cho|
00000070  63 6f 6c 61 74 65 20 63  68 69 70 64 00           |colate chipd.|

-rw-r--r--   1 kaz  staff  125  3 19 18:51 starbucks-avro.dat
```

JSONで直列化 - Avro

```
File fileOption = new File("schema-option.avro");
File fileOrder = new File("schema-order.avro");
Schema schemaOption = AvroUtils.parseSchema(fileOption);
Schema schemaOrder  = AvroUtils.parseSchema(fileOrder);

List<GenericRecord> array = new ArrayList<GenericRecord>();
GenericRecord thisOption = new GenericData.Record(schemaOption);
thisOption.put("name", "shot");
thisOption.put("price", 50);
array.add(thisOption);

FileOutputStream os = new FileOutputStream("starbucks-avro.json");
EncoderFactory factory = EncoderFactory.get();
Encoder  encoder = factory.jsonEncoder(schemaOrder, fos);
GenericDatumWriter<GenericRecord> writer =
      new GenericDatumWriter<GenericRecord>(schemaOrder);
writer.write(recordOrder, encoder);
encoder.flush();
```

JSONで直列化 - Avro

```
% cat starbucks-avro.json

{"size":0,"product":"Coffee Frappuccino","date":"March
11,2013","price":3.14,"currency":"JPY","count":6,"options":
[{"name":"shot","price":50},{"name":"Extra whip","price":
50},{"name":"Hazelnut","price":0},{"name":"Caramel
sauce","price":0},{"name":"Chocolate sauce","price":50},
{"name":"Chocolate chip","price":50}]}

-rw-r--r--   1 kaz  staff   316  3 19 18:51 starbucks-avro.json
```

直列化データの比較

サイズ

- 手入力のXML: 547 bytes
- Java直列化: 575 bytes
- **XStream: 688 bytes**
- IDL (アスキー): 364 bytes
- IDL (バイナリ): 142 bytes
- Protocol Buffers: 153 byes
- **Avro (バイナリ): 125 bytes**
- Avro (JSON): 316 bytes

特徴の比較

	スキーマは必要か？	コンパイラは必要か？	アスキーかバイナリか	サポートされる言語
Java直列化	No	No	バイナリ	Java
XStream	No	No	アスキー (XML)	Java
IDL	Yes	Yes	両方	C, Pascal
Protocol Buffer	Yes	Yes	バイナリ	Java, C++, Python
Avro	Yes	No	両方 (JSON)	C, C++, C#, Java, PHP, Python

6. JSONで直列化

Java用直列化のJSONライブラリ

- Flexjson
- Gson
- Jackson
- Json-lib
- JsonMarshall,
- JsonSmart

JSONで直列化 (2)

Java用のJSONによる直列化 ライブラリ

- Flexjson
- Gson
- Jackson
- Json-lib
- JsonMarshall, JsonSmart

```
-rw-r--r--  1 kaz  staff  323  3 19 19:47 starbucks-flexjson.txt
-rw-r--r--  1 kaz  staff  323  3 19 19:47 starbucks-gson.txt
-rw-r--r--  1 kaz  staff  323  3 19 19:48 starbucks-jackson.txt
-rw-r--r--  1 kaz  staff  323  3 19 19:48 starbucks-jsonMarshaller.txt
-rw-r--r--  1 kaz  staff  323  3 19 19:49 starbucks-jsonSmart.txt
-rw-r--r--  1 kaz  staff  323  3 19 19:48 starbucks-jsonlib.txt
```

7. Protostuff

http://code.google.com/p/protostuff/
Protocol Buffersを拡張

いくつかの形式をサポート

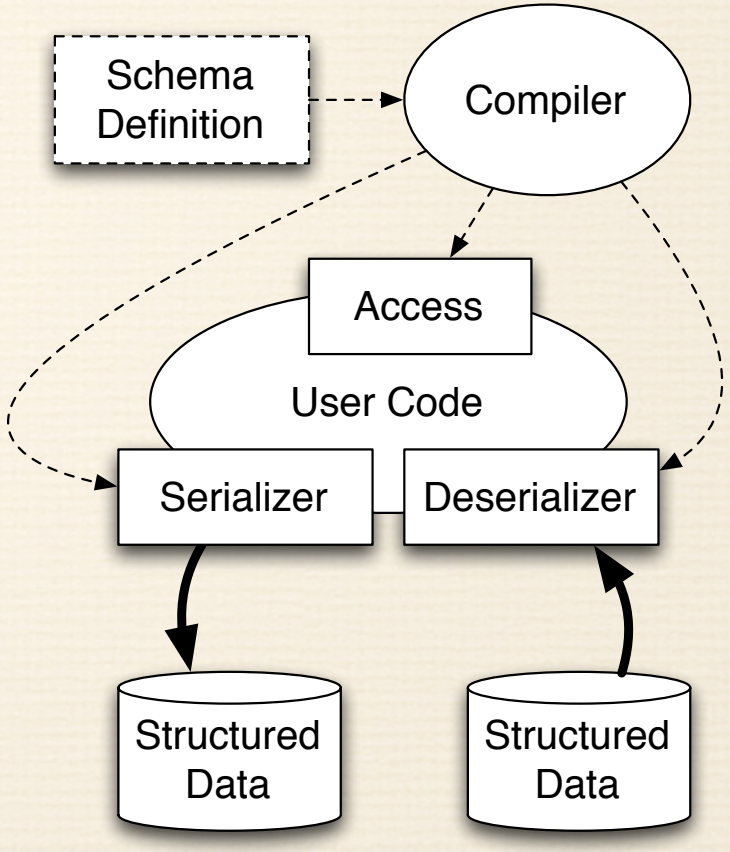
- protostuff (native), protobuf, JSON,
- smile (バイナリ json), XML, Yaml, KVP

スキーマ定義は必要，以下の両方サポート

- コード生成　(スキーマをコンパイル時に使用)
- コード生成不要 (スキーマを実行時に使用)

スキーマ定義 - Protocol Stuf.

```
option java_package = "org.curlos.protostuff";
message Order {
  enum CupSize {  Small = 1; Tall = 2; Grande = 3; Venti = 4;  }
  required CupSize size = 1;
  required string product = 2;
  required string date = 3;
  required double  price = 4;
  required string currency = 5;
  required int32  count = 6;
  message Option {
    required string name = 1;
    required int32 price = 2;
  }
  repeated Option options = 7;
}
```



Serialization - Protostuff

```
Order anOrder = new Order();   // org.rugson.protobuf.Order
anOrder.setSize(Order.CupSize.Grande);
anOrder.setProduct("Coffee Frappuccino");
.....
List<Option> options = new ArrayList<Option>();
Option anOption = new Option(); // org.rugson.protobuf.Option
anOption.setName("Shot");    anOption.setPrice(50);
anOrder.addOptions(thisOption);
```

Writer

```
PrintWriter pw = new PrintWriter("protostuff.txt");
Schema<Order> schema = Order.getSchema();
JsonIOUtil.writeTo(pw, anOrder, schema, false);
```

Reader

```
FileReader fr = new FileReader("protostuff.txt");
Order thisOrder = new Order();
JsonIOUtil.mergeFrom(fr, thisOrder, schema, false);
```

Yaml - Protostuff

```
--- !Order
1: 2
2: Coffee Frappuccino
3: March 11,2013
4: 3.14
5: JPY
6: 6
7: !Option[]
  -
    1: shot
    2: 50
  -
    1: Extra whip
    2: 50
  -
    1: Hazelnut
    2: 0
  -
    1: Caramel sauce
    2: 0
  -
    1: Chocolate sauce
    2: 50
  -
    1: Chocolate chip
    2: 50
```

8. MessagePack

http://msgpack.org/
速い処理，小さいフォーマット

多言語サポート

- Ruby, Python, Perl, C/C++, Java, Scala, PHP, Lua
- JavaScript, Node.js, Haskell, C#, Objective-C, Erlang
- D, OCaml, Go, LabVIEW, Smalltalk

スキーマ定義

- 基本的に必要ない
- msgpack-idl がスキーマ定義からコード生成

JavaだとAnnotationが便利

```
@Message
public class Order {
    public String size;
    public String product;
    public String date;
    public double price;
    public String currency;
    public int count;
    public List<Option> options;
}
```

```
@Message
public class Option {
    public String name;
    public int price;
}
```

```
MessagePack msgpack = new MessagePack();
byte[] bytes = msgpack.write(thisOrder);


Order thisOrder = msgpack.read(bytes, Order.class);
```

スキーマ定義 - MessagePack

```
namespace org.curlos.mpack
enum CupSize {
  1: Small
  2: Tall
  3: Grande
  4: Venti
}
message Option {
  1: string name
  2: int price
}
message Order {
  1: CupSize size
  2: string product
  3: string date
  4: double price
  5: string currency
  6: int count
  7: list<Option> options
}
```

```
msgpack-idl -g java starbucks-mpack.idl
```

直列化 - MessagePack

```
% hexdump -C starbucks-java.dat | head -11
00000000  97 a6 47 72 61 6e 64 65  b2 43 6f 66 66 65 65 20  |..Grande.Coffee |
00000010  46 72 61 70 70 75 63 63  69 6e 6f ad 4d 61 72 63  |Frappuccino.Marc|
00000020  68 20 31 31 2c 32 30 31  33 cb 40 09 1e b8 51 eb  |h 11,2013.@...Q.|
00000030  85 1f a3 4a 50 59 06 96  92 a4 53 68 6f 74 32 92  |...JPY....Shot2.|
00000040  aa 45 78 74 72 61 20 77  68 69 70 32 92 a8 48 61  |.Extra whip2..Ha|
00000050  7a 65 6c 6e 75 74 00 92  ad 43 61 72 61 6d 65 6c  |zelnut...Caramel|
00000060  20 73 61 75 63 65 00 92  af 43 68 6f 63 6f 6c 61  | sauce...Chocola|
00000070  74 65 20 73 61 75 63 65  32 92 ae 43 68 6f 63 6f  |te sauce2..Choco|
00000080  6c 61 74 65 20 63 68 69  70 32                    |late chip2|
```

-rw-r--r-- 1 kaz staff **138** 5 17 02:22 starbucks-mpack.dat

直列化データの比較

File Size

- Hand-coded XML: 547 bytes
- Java serialization: 575 bytes
- **XStream: 688 bytes**
- IDL ascii: 364 bytes
- IDL binary: 142 bytes
- Protocol Buffers: 153 byes
- **Avro binary: 125 bytes**
- Avro JSON: 316 bytes
- Protostuff Yaml: 287 bytes

<参考: 他の手法>

Thrift binary: 150 bytes
Thrift JSON: 316 bytes
MessagePack: 138 bytes

簡単な実験

実験概説

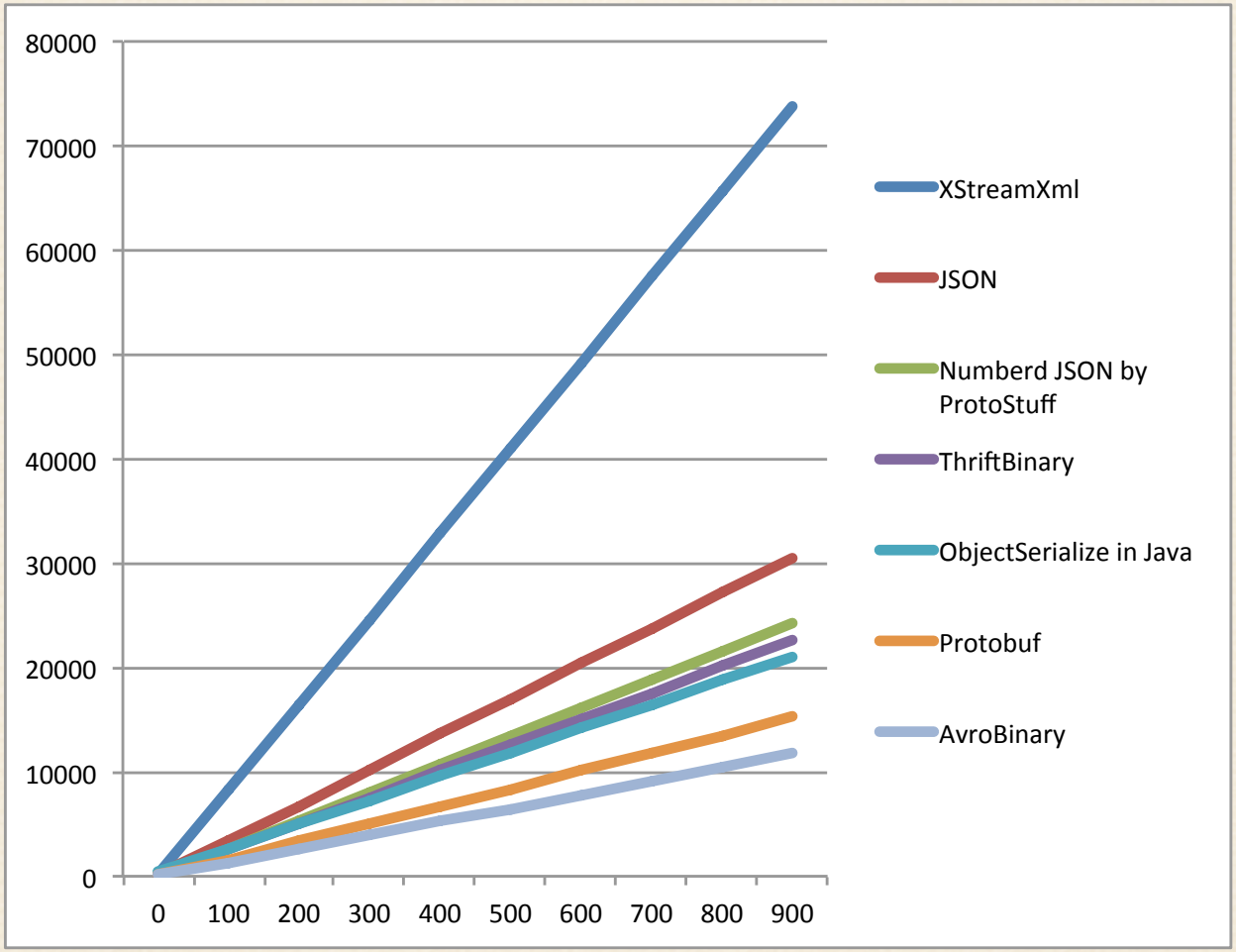
- ハードウェア
  - iMac with Intel Core2 Duo 2.66GHz, 2GB memory
- ソフトウェア
  - Mac OS X version 10.6.8, Java version 1.6.0_29
- 10種類のコーヒー注文で，以下の数のオプション付き
  - 0, 100, 200, 300, 400, 500, 600, 700, 800, 900
- Javaでプログラム
  - オプション付きのOrderオブジェクトを一つ作成
  - 500回直列化・逆直列化．データ棄却（暖機運転）
  - 500回直列化・逆直列化．データ測定（実運転）

Order with 100 options

```
<order>
 <size>Grande</size>
 <product>Coffee Frappuccino</product>
 <date>March 3,2013</date>
 <price>3.14</price>
 <count>100</count>
 <options>
  <option>
   <name>option0000</name>
   <price>0</price>
  </option>
  <option>
   <name>option0001</name>
   <price>1</price>
  </option>
  <option>
   <name>option0002</name>
   <price>2</price>
```

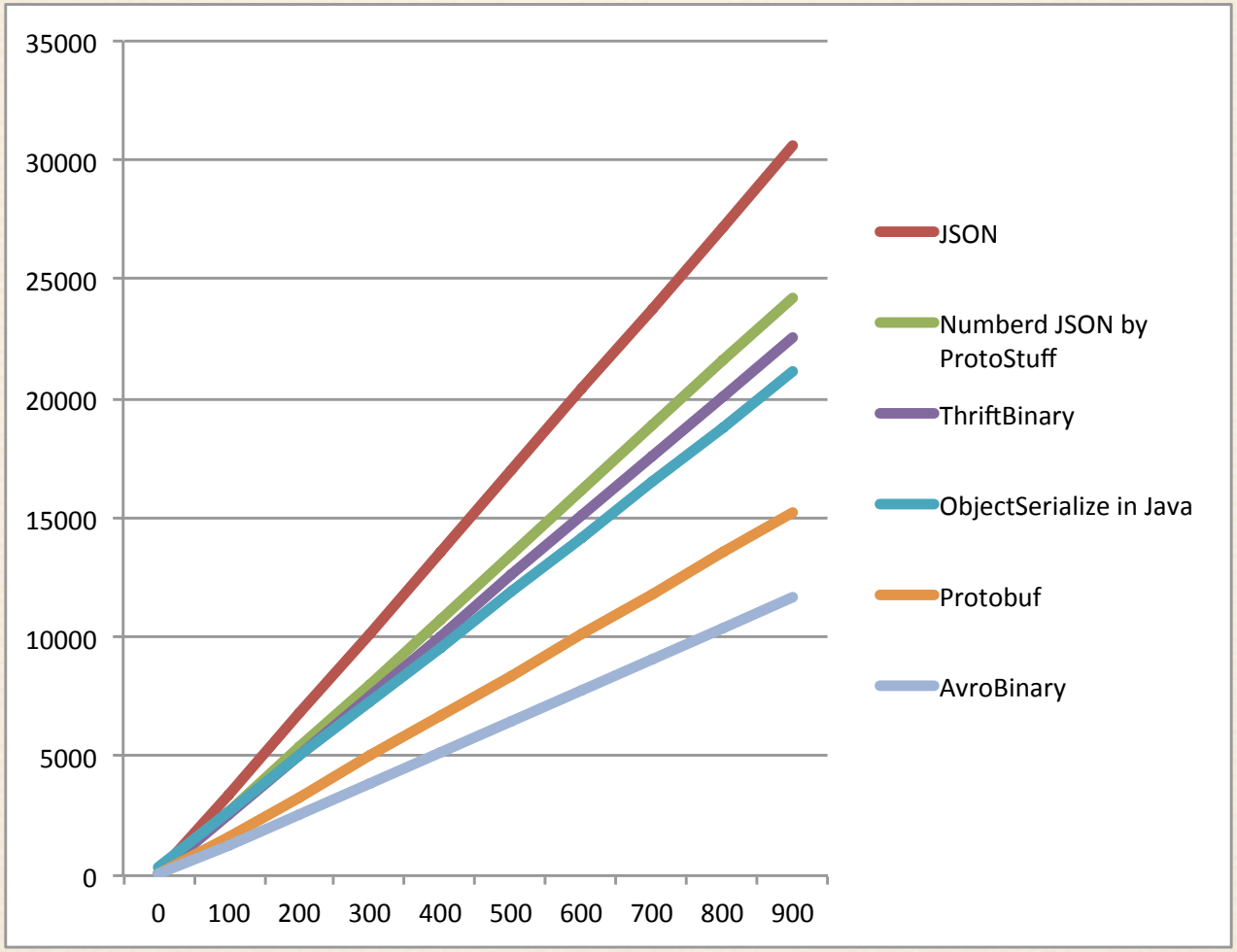
Size of Serialized Data (1)

- All data (Bytes)



Size of Serialized Data (2)

- delete XStream



Numbered JSON (Protostuff)

- JSON

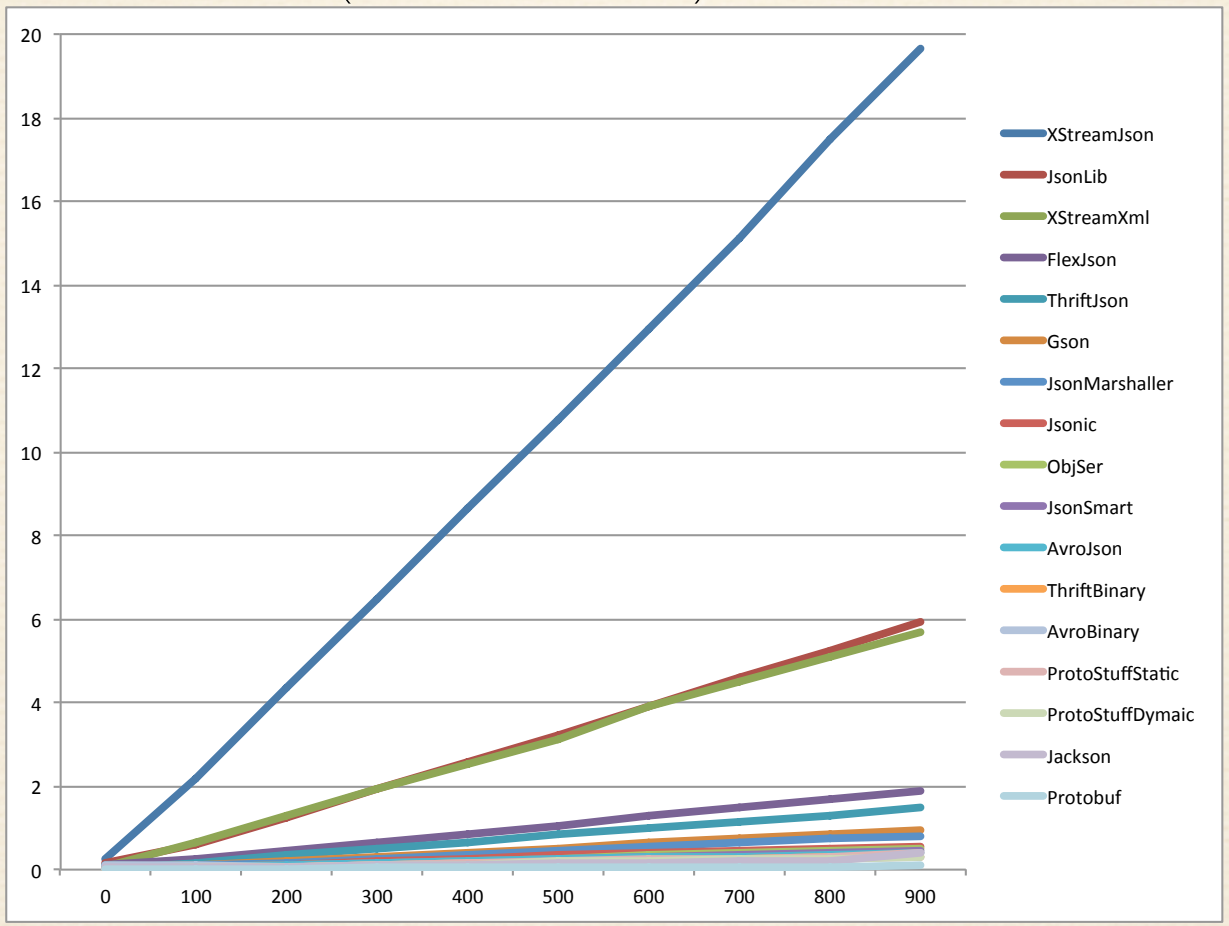
{"size":2,"product":"Coffee Frappuccino","date":"March 3,2013","price":3.14,"count":0}

- Numbered JSON

{"1":2,"2":"Coffee Frappuccino","3":"March 3,2013","4":3.14,"5":0}

Time to Serialize

- All (ms measured by System.currentTimeMillis())
  - Measured time (500 iterations) / 500



ここまでの経験から

前田としては,

- データ読み書き簡単なアスキー形式
- 簡単なスキーマ定義
- 性能を上げるためにコード生成
- グラフ構造をサポート

する直列化技法が欲しい

まとめ

オブジェクト直列化の技法を

いくつか紹介

- Serialization in Java, IDL, XStream, Protocol Buffers
- Apache Avro, MessagePack

比較

- 簡単な記述でサイズを比較

ありがとうございました

前田和昭

Kazuaki Maeda

kaz@acm.org
kmaeda@gmail.com