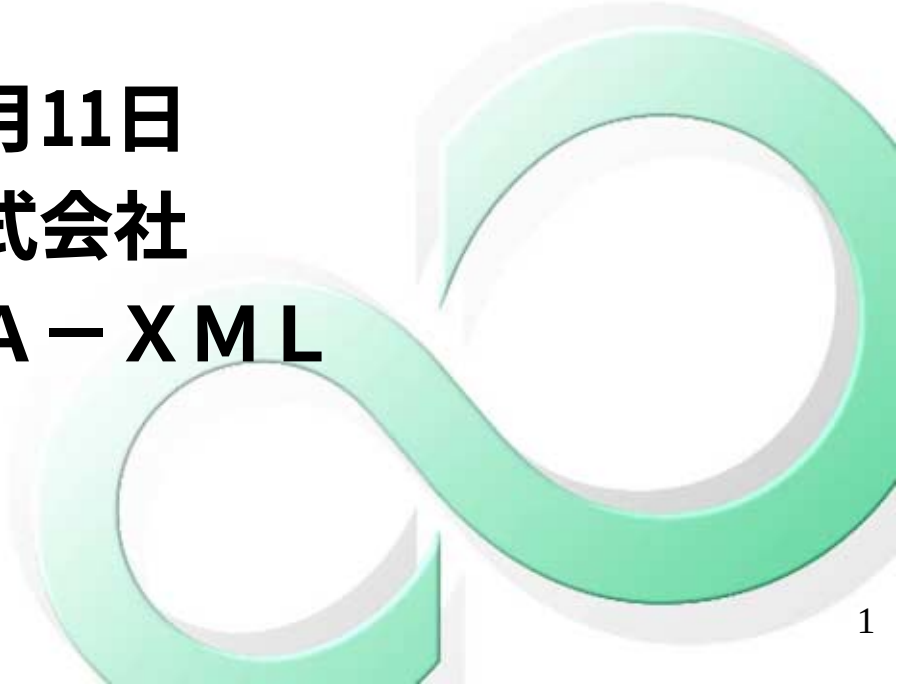
分散システム設計実装上のポイントとアーキテクチャ

2002年11月11日
富士通株式会社
プロジェクトＡ－ＸＭＬ



内容

- XML
  - アーキテクチャ例
  - 設計実装上のポイント
- XML webサービス
  - アーキテクチャ例
  - 設計実装上のポイント

XML

- アーキテクチャの一例
- XMLシステムの設計実装上のポイント
  - XMLを操作するコードの書き方
  - どこまでXML形式で持ちまわるか
  - XMLのDB格納方式
  - XML電文設計
  - セキュリティ

XML

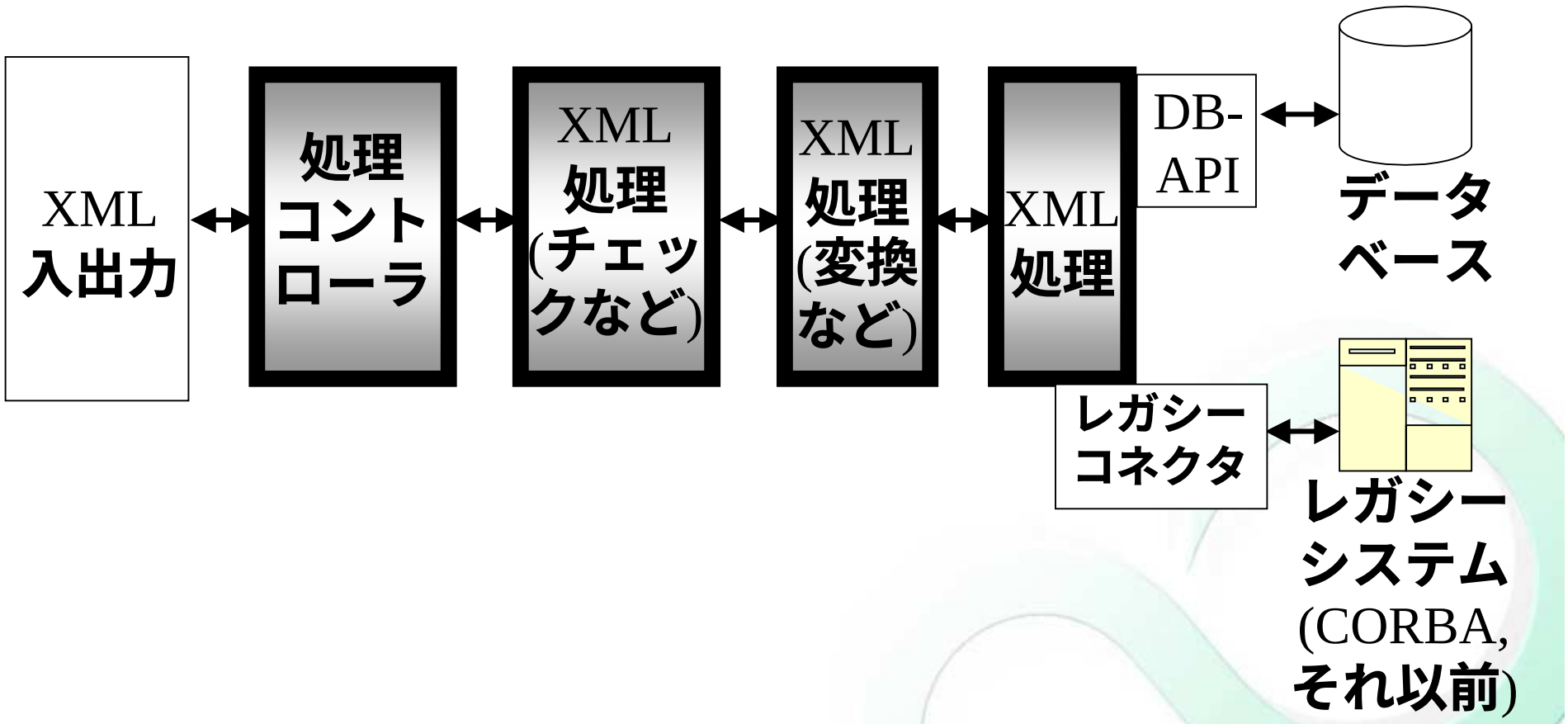
疎な連携を支援

- **粒度の大きなサブシステム同士の結合に有効**
- **データフォーマットの変更に伴うプログラム変更が少なくなる**

多くのツールが存在(増加中)

- **多くのベンダ，団体，個人がツール，ライブラリ，コンポーネントを開発**

XMLを扱うシステムアーキテクチャの一例



XMLを利用するシステムの設計実装ポイント

- XMLを操作するコードの書き方
- どこまでXML形式で持ちまわるか
- XMLのDB格納法
- XML電文設計
- セキュリティ

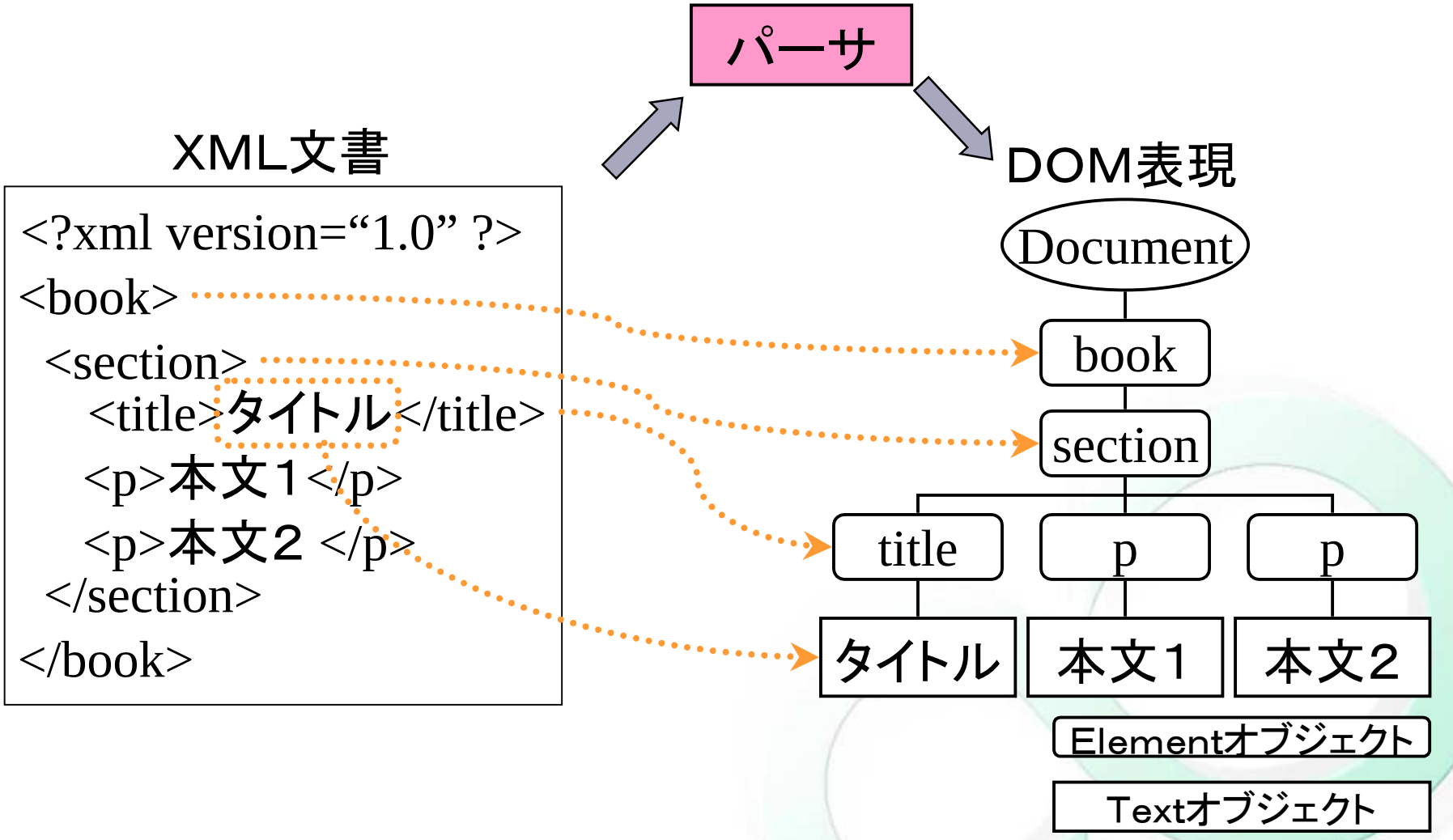
XMLを操作するコードの書き方

操作の方式

- DOM
- SAX
- オブジェクト

XML操作：DOM

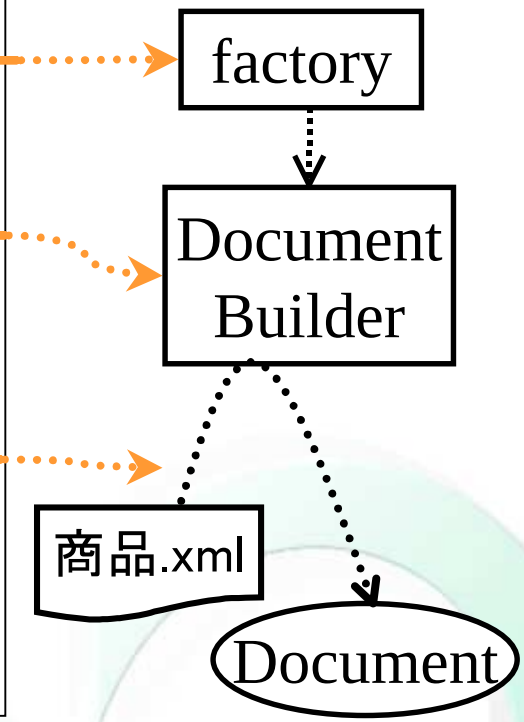
XMLをオブジェクトのツリー構造で表現



パーサの準備とXMLのパース

```
DocumentBuilderFactory docBuilderFactory =
  DocumentBuilderFactory.newInstace();
DocumentBuilder documentBuilder =
  docBuilderFactory.newDocumentBuilder();

Document document =
  documentBuilder.parse( “商品.xml”);
```



DOMへのアクセス

DOM　API

- Node#getChildNodes()
- Node#getParentNode()
- Document#getDocumentElement()
- Document#getElementsByTagName(String 要素名)
- Element#getElementsByTagName(String 要素名)
- Element#getTagName()
- Element#getAttribute(String 属性名)
- Node#getNodeValue()
- …

DOMの変更

DOM　API

Node#appendChild(Node 子ノード)

Node#insertBefore(Node 追加ノード,Node 対象ノード)

Node#removeChild(Node 子ノード)

Element#setAttribute(String 属性名,String 値)

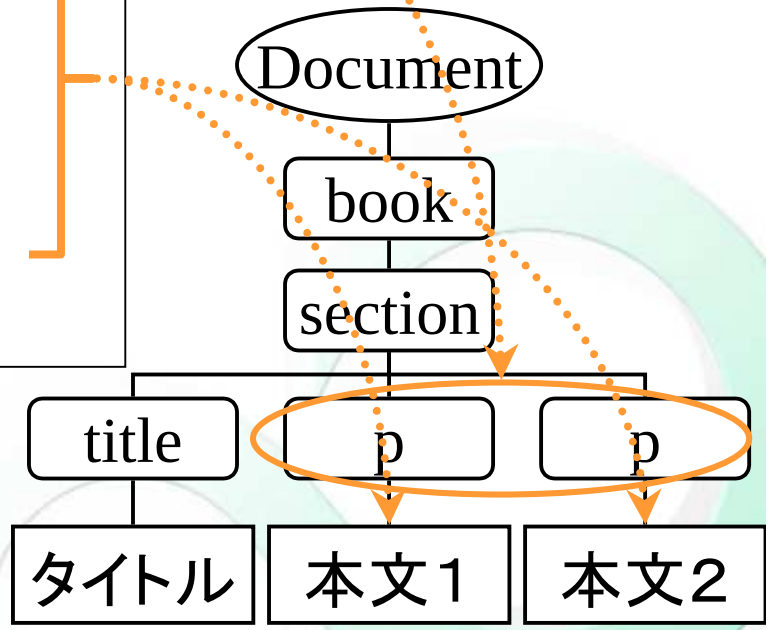
Element#removeAttribute(String 属性名)

Node#setValueNode(String 値)

…

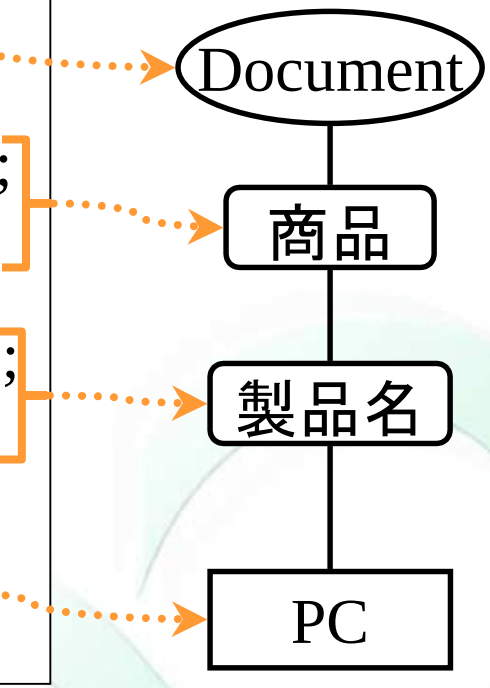
DOMアクセス例

```
NodeList nl =
  document.getElementsByTagName(“p”);
for (int i = 0; i < nl.getLength(); i++){
  Element elem = (Element)nl.item(i);
  NodeList textlist = elem.getChildNodes();
  Node text = textlist.item(0);
  String s = text.getNodeValue();
  text.setNodeValue(s+”. ”);
}
```



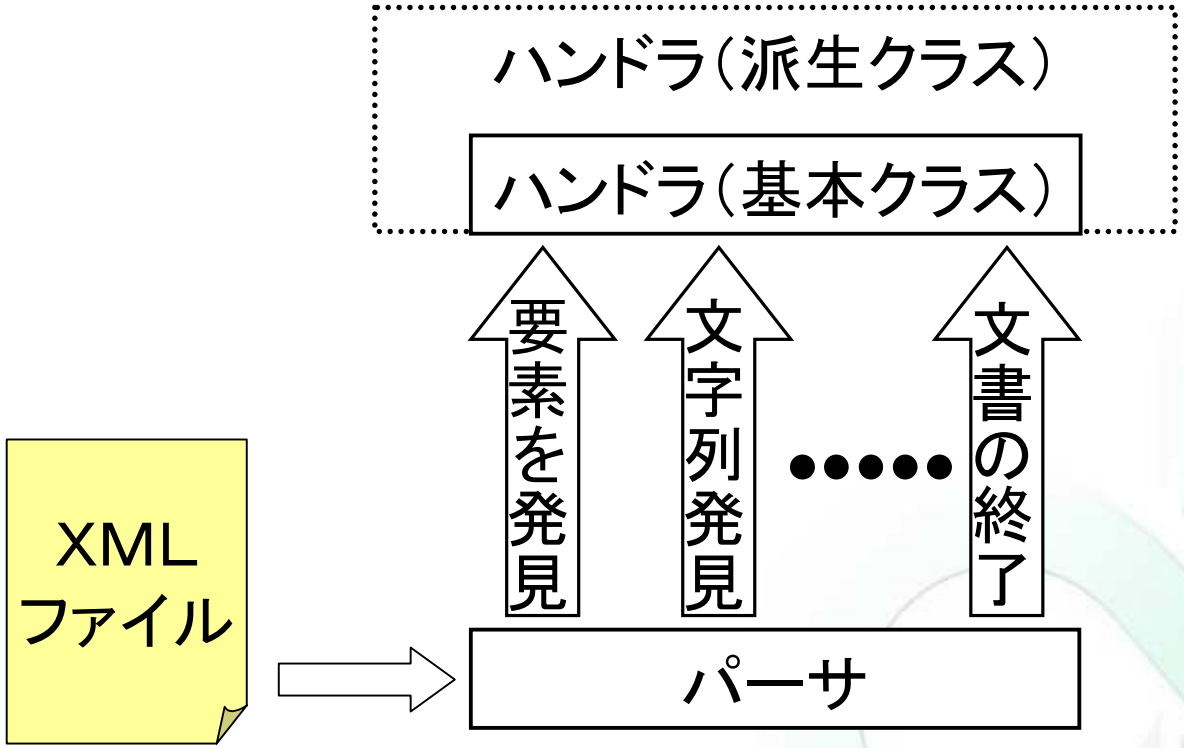
DOM生成例

```
Document doc =
  documentBuilder.newDocument();
Element product = doc.createElement( “商品”);
doc.appendChild(product );
Element name = doc.createElement( “製品名”);
product.appendChild( name);
Text txt = doc.createText( “PC”);
name.appendChild( txt);
```

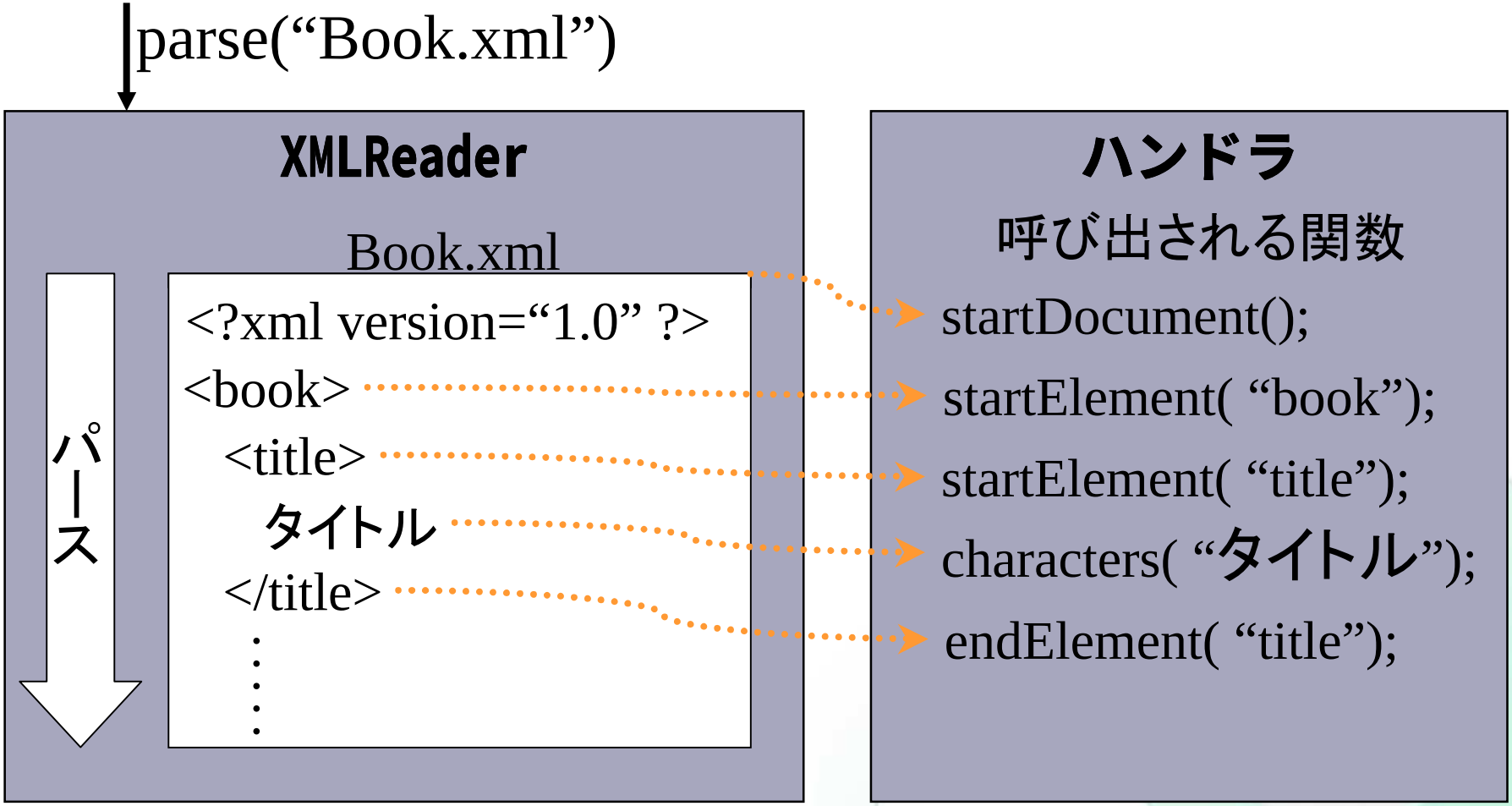


XML解析：SAX

- ハンドラインタフェースを実装したクラス（ハンドラ）を作りパーサにセット.
- XMLをパースするとハンドラのメソッドが呼び出される.



SAXによるパージング



SAXパーサのAPI

SAX API

XMLReaderFactory#createXMLReader()

- SAXパーサ（XMLReaderインスタンス）の生成

XMLReader#setContentHandler(Handler ハンドラ)

- SAXイベントを受け取るハンドラを設定

XMLReader#parse(InputSource xml)

- XMLのパースを実行

SAXイベント（ハンドラメソッド）

SAX API

ContentHandler#startDocument()

- ＸＭＬ文書の開始を通知

ContentHandler#startElement(String, AttributeList)

- 要素の開始を通知

ContentHandler#characters(char[], int, int)

- 要素内容などの文字列を通知

ContentHandler#endElement(String)

- 要素の終了を通知

ContentHandler#endDocument()

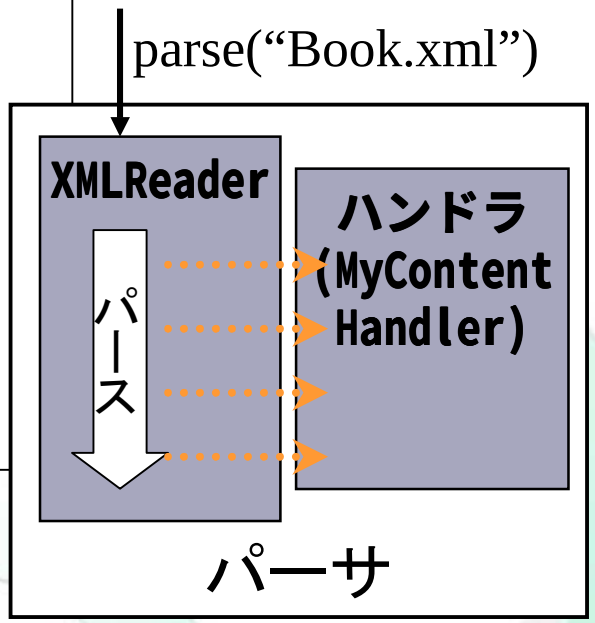
- 文書の終了を通知

SAXによるパースの実行

```
// パーサを生成
XMLReader reader
    = XMLReaderFactory.createXMLReader
                ("org.apache.xerces.parsers.SAXParser");

//ハンドラを設定
reader.setContentHandler(new MyContentHandler());

//XML文書をパース
reader.parse("Book.xml");
```



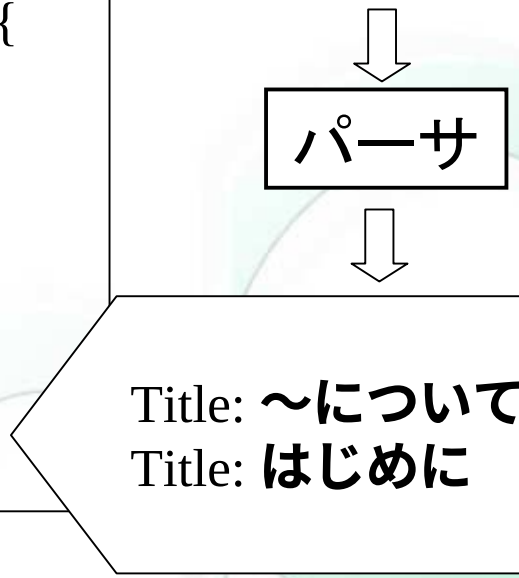
SAXハンドラ (1/2)

```
/** 要素titleの子テキストを表示するハンドラ */
public class MyContentHandler extends DefaultHandler {
  /** 要素titleの子テキストを保持する変数 */
  private StringBuffer title = null;

  /** 要素の開始を処理 */
  public void startElement(String namespaceURI,
        String localName, String qName, Attributes atts) {

    if (localName.equals("title")) {
      title = new StringBuffer();
    } else if (localName.equals(...)) {
      ...
    }
  }
```

```
<?xml version="1.0" ?>
<book>
  <title>～について</title>
  <chapter>
    <title>はじめに</title>
  </chapter>
  ...
</book>
```



SAXハンドラ (2/2)

```
  /** 文字列を処理 */
  public void characters(char[] ch, int start, int length) {
    if (title != null) {
      title.append(ch, start, length);
    }
  }

  /** 要素の終了を処理 */
  public void endElement(String namespaceURI,
                         String localName, String qName) {
    if (localName.equals("title")) {
      System.out.println("Title: " + title.toString());
      title = null;
    }
  }
}
```

```
<?xml version="1.0" ?>
<book>
  <title>〜について</title>
  <chapter>
    <title>はじめに</title>
  </chapter>
  ...
</book>
```


パーサ

Title: **〜について**
Title: **はじめに**

演習 1

1. 右の入力XMLに対する出力は？

```
<?xml version="1.0" ?>
<book>
 <title>〜について<sub>XXに捧ぐ</sub></title>
 <chapter>
  <title>はじめに<sup><b>1</b></sup></title>
 </chapter>
 ...
</book>
```

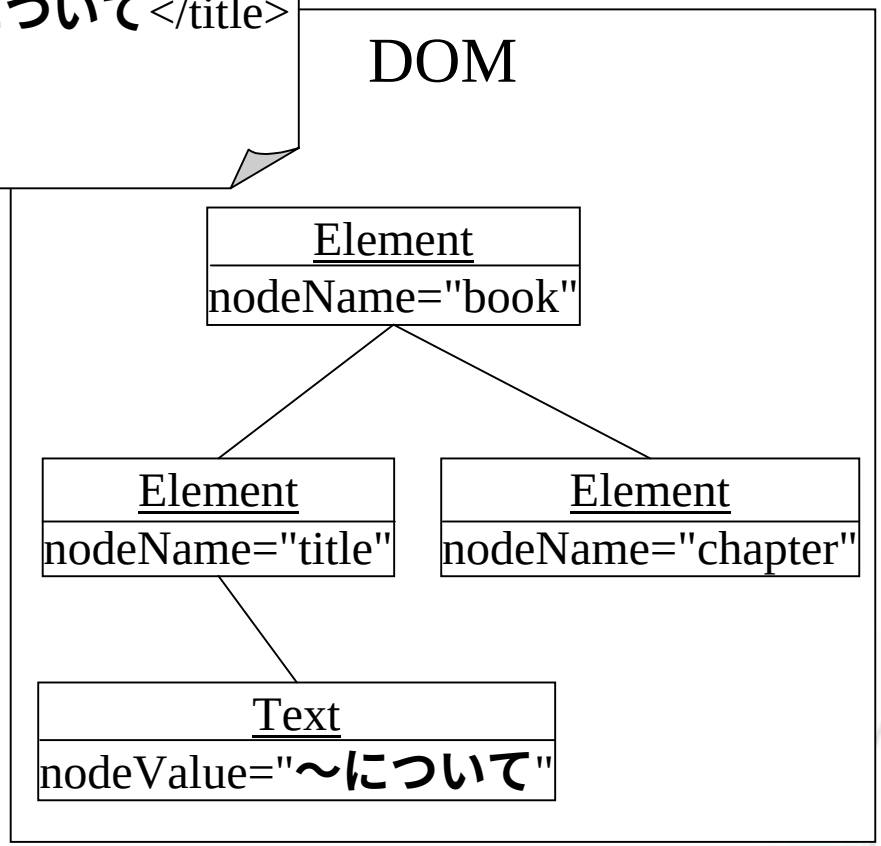
2. つぎの内容を出力するためのソース変更

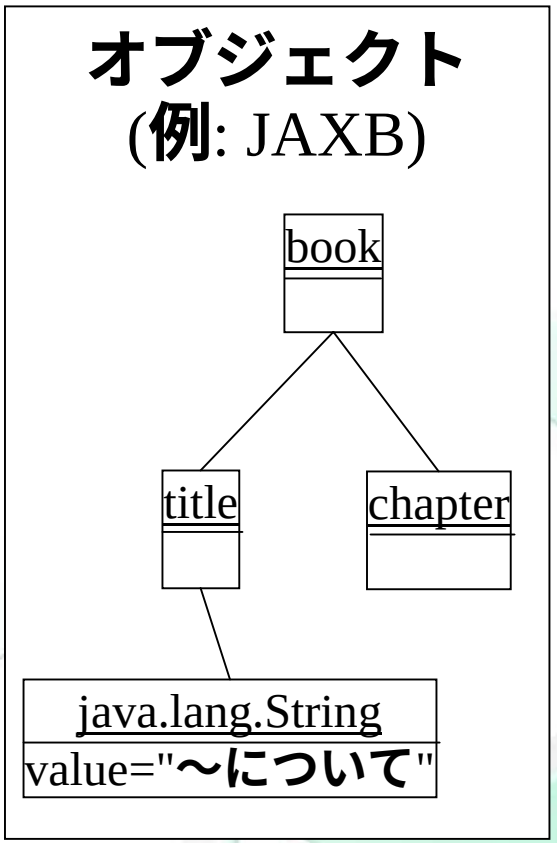
Title: 〜について
Title: はじめに

オブジェクトによる操作

XMLの構造を反映したクラスで操作

```
<?xml version="1.0" ?>
<book>
  <title>～について</title>
  <chapter/>
</book>
```

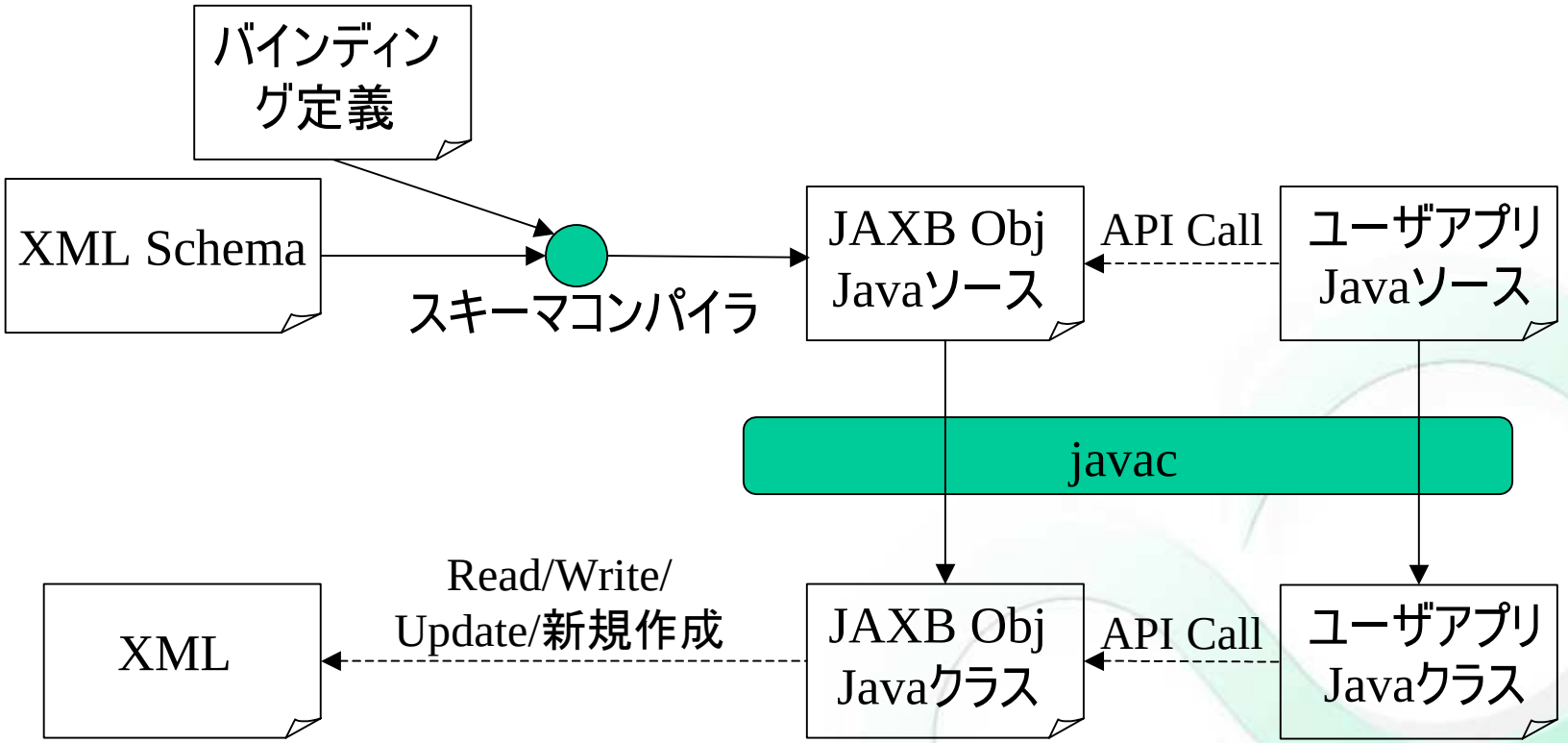




オブジェクトによる操作

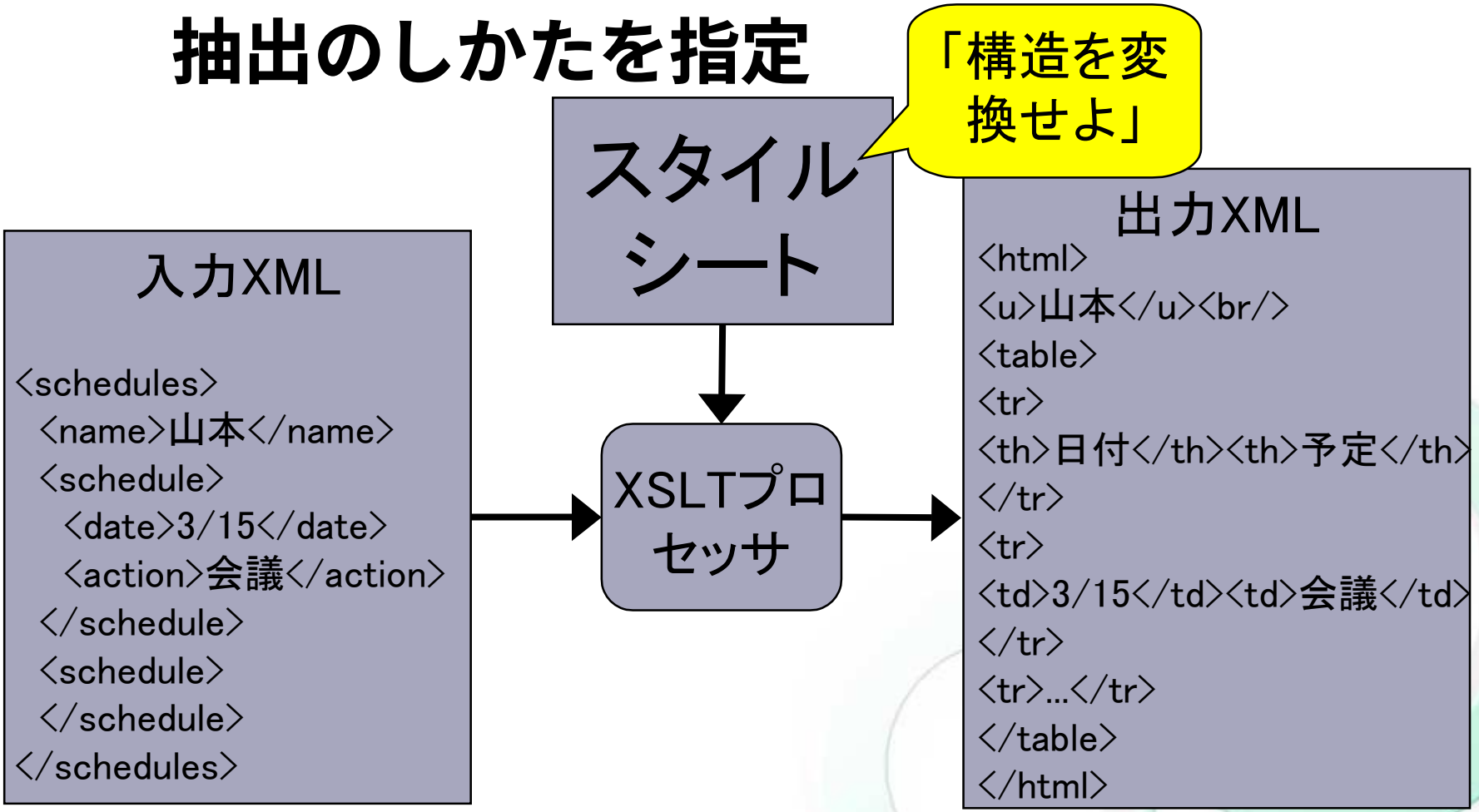
XMLの構造に合わせてクラスを用意

- JAXBの場合



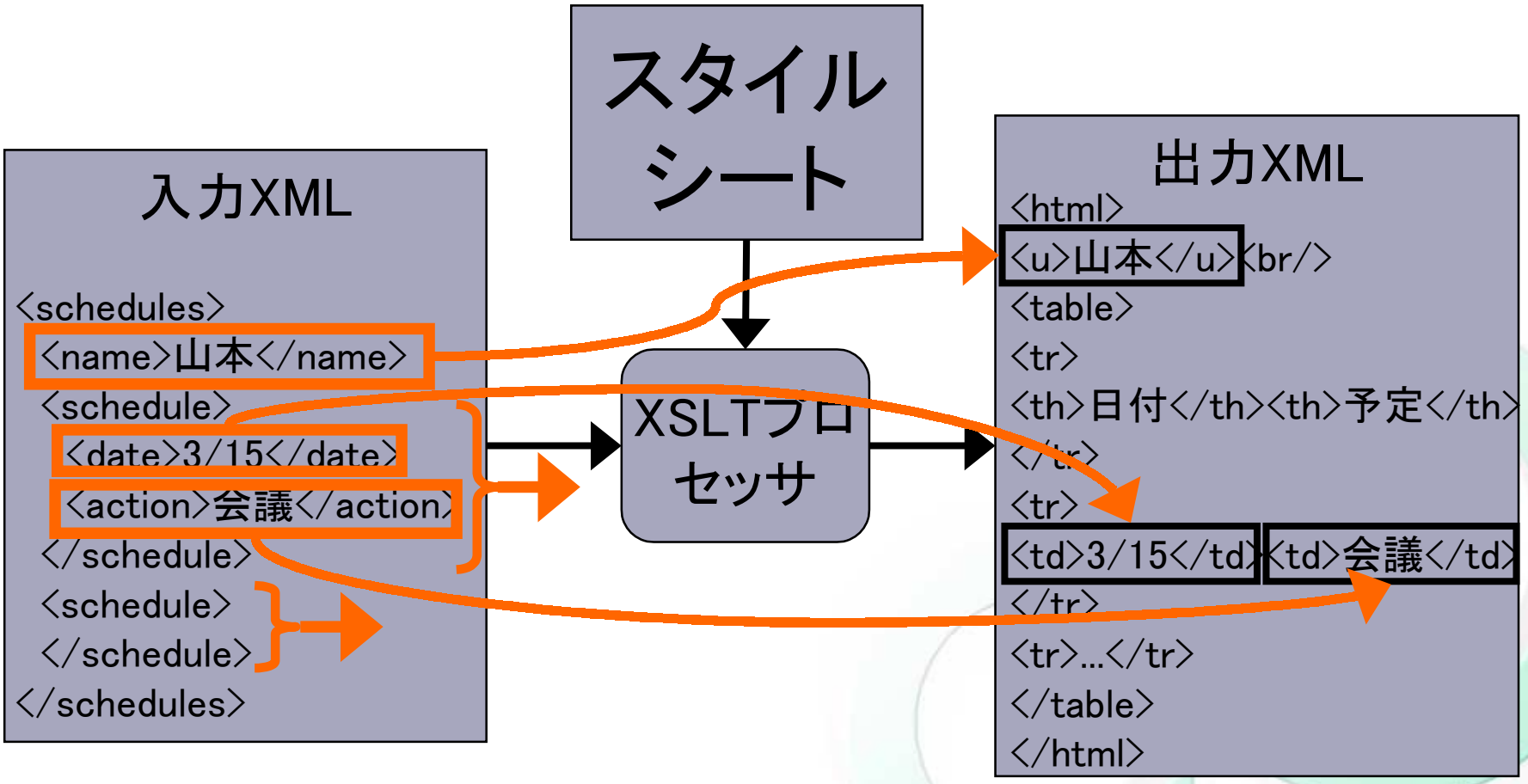
XML操作：XSLT（スタイルシート）

XSLT（スタイルシート）でデータの変換・抽出のしかたを指定

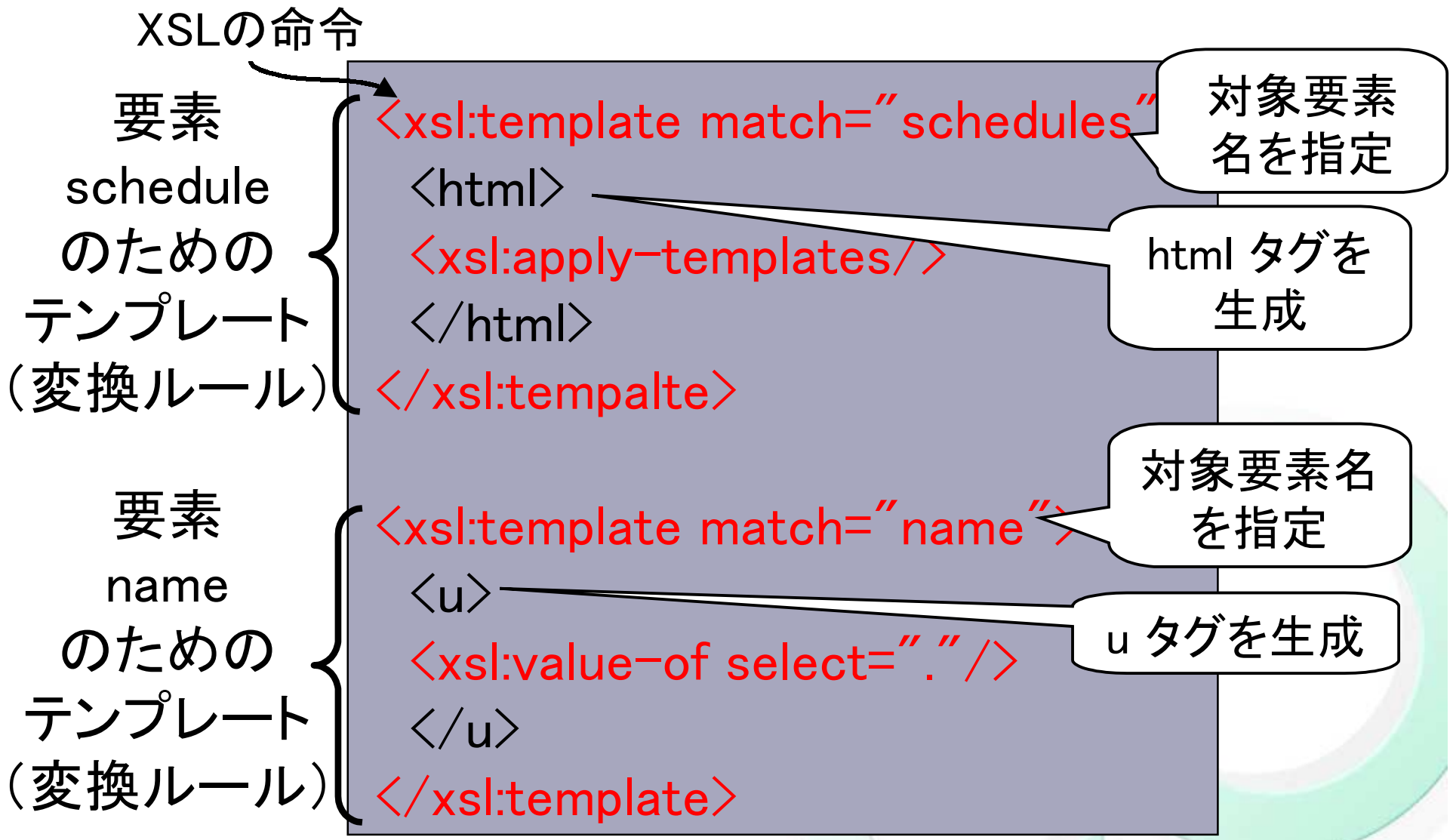


XML編集：XSLT（スタイルシート）

具体的には，ノードのマッピングを指定



スタイルシートの例



JAXP： XML編集のAPI

変換

TransformerFactory#newTransformer(Source スタイルシート)

- 指定されたスタイルシートで変換するTransformerを作る.

Transformer#transform(Source 入力XML, Result 出力XML)

- 入力XMLを変換して出力する.

Transformer#setOutputProperty(String 名前, String 値)

- 出力プロパティを設定する.
  例： setOutputProperty(OutputKeys.ENCODING, "Shift_JIS") → シフトJISで出力する.

入力・出力 (DOM, SAX, Stream が指定可能)

StreamSource#StreamSource(String ファイル名)

- 入力（指定されたファイルから読み込むタイプ）を作る.

StreamSource#StreamSource(InputStream 入力ストリーム)

- 入力（ストリームから読み込むタイプ）を作る.

StreamResult#StreamResult(OutputStream 出力ストリーム)

- 出力（ストリームへ書き出すタイプ）を作る.

XMLの操作方式の選択

アプリで重視する項目によって選択

- DOM
  - 長所:複雑な構造を持つXMLをランダムに参照，修正
  - 短所:低速．メモリを多く消費
- SAX
  - 長所:高速なパース
  - 短所:複雑な処理を行うユーザアプリが書きにくい
- XSLT
  - 長所:再コンパイルなしに変更が可能
  - 短所:処理が遅い．複雑な変換を書くとあとのメンテが大変
- オブジェクト
  - 長所:ユーザアプリにXMLを意識させない
  - 短所:XMLのスキーマが変るたびに再コンパイルが必要

どこまでXML形式で持ちまわるか

- XMLを用いることで開発の効率化が図れる
  - 疎結合なシステムの連携には大変適している
  - **数多くのツール，ライブラリ，コンポーネントも使える**
- 一方でXMLはあくまでデータ

  次のような場合XMLではないほうがよいことも
  - 密結合なシステム間で処理速度が求められる場合
  - データとしてではなく，業務知識について責任を持つクラスとして扱ったほうがよい場合

    なお，JAXBなどで出来るクラスは単なるデータ型で「一人前のクラス」ではない

XMLのDB格納方式

- RDB
- XMLDB
- ファイル

RDBへの格納

・テーブルに分解して格納

XML文書

```
<?xml version=“1.0” ?>
<order>
  <item id="1">
    <name>鉛筆</name>
    <count>12</count>
  </item>
  <item id="2">
    <name>ペン</name>
    <count>25</count>
  </item>
</order>
```

RDB

id	name	count
1	鉛筆	12
2	ペン	25

RDBが高速に検索できる形での格納．XMLスキーマの変更には弱い

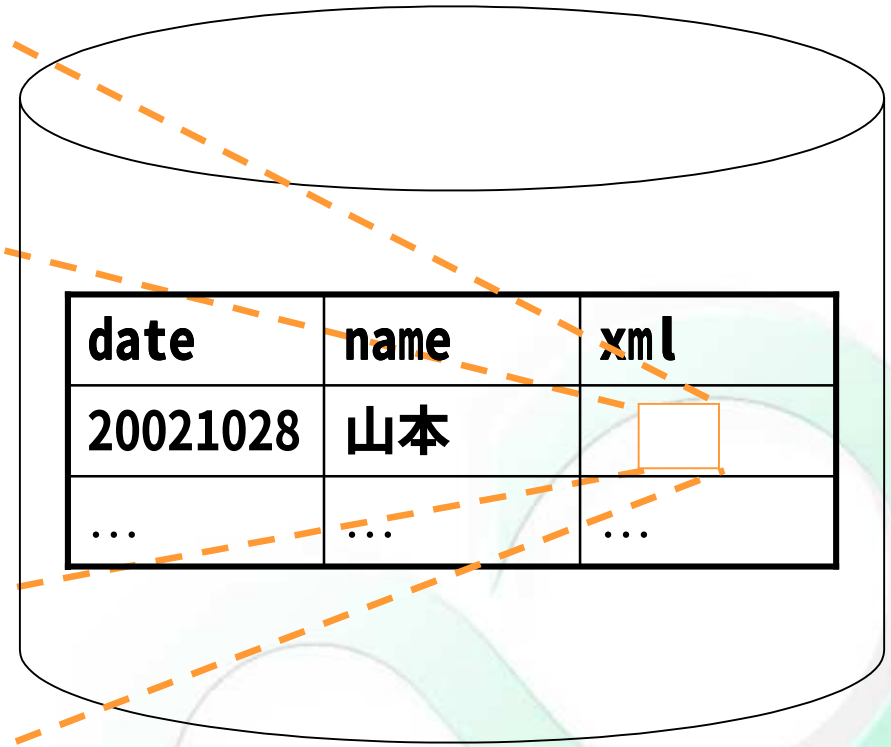
RDBへの格納

- 1カラム(BLOBカラム)に格納
  (検索に必要な項目は別カラムへ)

XML文書

```
<?xml version="1.0" ?>
<application>
  <date>20021028</date>
  <type>新規</type>
  <person>
    <name>山本</name>
    <tel>03-4567-8910</tel>
    <gender>M</gender>
    <addr>目黒区大岡山2-12-1</addr>
  </person>
</application>
```

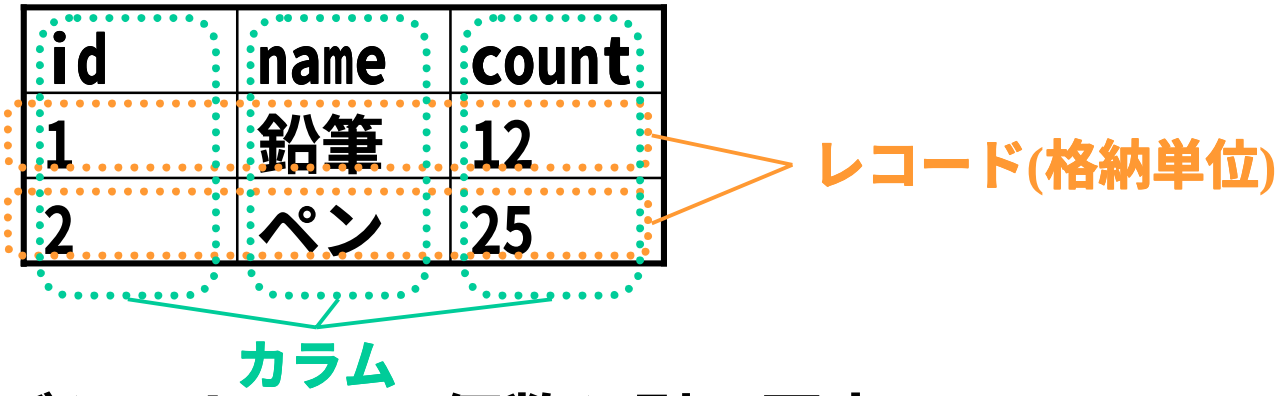
RDB



XMLスキーマの変更に対応できる格納方式

参考: RDBの概略

- データを2次元の表に格納する



- 表ごとにカラムの個数と型は固定
- カラムの値を条件にレコードを検索
- 条件を組み合わせて複雑な検索を行う
- カラムは数値，文字列のほか，大きなサイズのバイナリを格納できる．このバイナリを格納できるカラム型がBLOB型

XMLDBへの格納

XMLDBとは

XMLの構造に適した検索ができる格納システムの総称

ノードを特定した検索

例えば「要素typeの値が"新規"であるXML文書を取出す」

実装はさまざま(OODB，RDB，独自形式...)

検索性能，格納性能，信頼性もさまざま

→各性能ともすぐれたものがない

APIもさまざま

- 標準化をめざしているインタフェース:

  http://www.xmldb.org/xapi/

格納方式の選択 (1)

システム要件に合わせて選択

- RDB

  長所:障害発生時に復旧する機能，ノウハウが充実

  - カラムにマッピング

    長所:ノード単位の検索が速い．ノードの値の変更も高速

    短所:XMLのスキーマの変化に弱い

  - 1カラムに格納

    長所:XMLのスキーマの変化に対応しやすい

    短所:あらかじめ想定されたノード以外の検索に難．

    短所:ノード1つの変更でもXML全体を格納しなおす必要あり

  (各DBベンダは短所を克服する製品をリリース中)

格納方式の選択 (2)

システム要件に合わせて選択

- XMLDB
  - 長所:XMLの構造を意識した検索が得意
  - 短所:障害発生時に対処する機能が弱いDBもある
  - (各DBベンダは短所を克服する製品をリリース中)
- ファイル
  - 長所:システムが簡素．読み出しだけならこれで十分な場合も
  - 短所:大部分をアプリ開発者が作らなければならない

XML電文設計

- 業務分析の結果や既存のDBのスキーマ等から洗い出す
- XMLを使ってインタラクションする部分を明確にし，インタラクションに必要なデータと構造を作成して，XML電文とする
- 標準的なボキャブラリをまえもって調査しておく

セキュリティ

XMLに対する暗号化や署名

セキュリティをかけることで，だれのどんな行為を防げるか

- **暗号化で防げる行為**

  盗聴：通信経路(ネットワーク，ルータ)上での盗聴

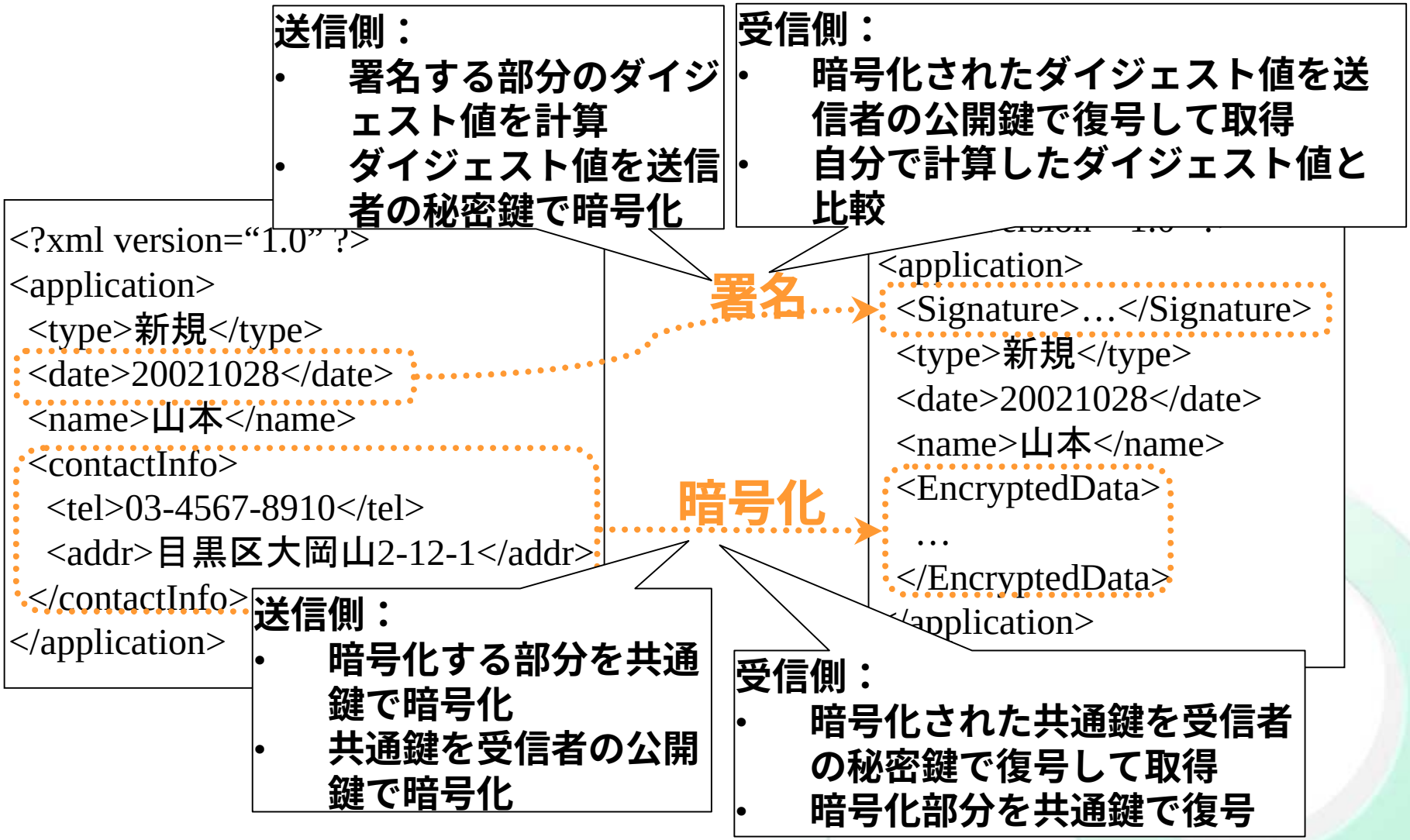
- **署名で防げる行為**

  改竄：通信経路上でのデータ改竄

  なりすまし：自分以外の人，サイトを装って情報を送信/取得

  否認：送信/受信したデータを送信/受信していないと主張

XMLの暗号化・署名



XMLの部分ツリーだけ暗号化/署名することが可能

セキュリティ

盗聴

受信者の秘密鍵を持つ者しか暗号化データを復号できない

改竄

少しでも文書を変更するとダイジェスト値が大きく異なる
送信者の秘密鍵を持つ者しかダイジェスト値を暗号化できない

なりすまし

送信者の秘密鍵を持つ者しかダイジェスト値を暗号化できない

否認

ダイジェスト値を送信者の公開鍵で復号できることで，
値を暗号化したのが送信者の秘密鍵を持つ者と断定できる

送信者/受信者の秘密鍵を持つ者が本人であることは別途確認が必要
→ 認証局が身元を保証．認証局の証明書で本人確認．

XML webサービス

- アーキテクチャの一例
- 設計実装のポイント
  - Webサービスインタフェースの分析
  - Webサービスのインテグレーション
  - 他者から提供されるwebサービスとの連携
  - UDDIの利用

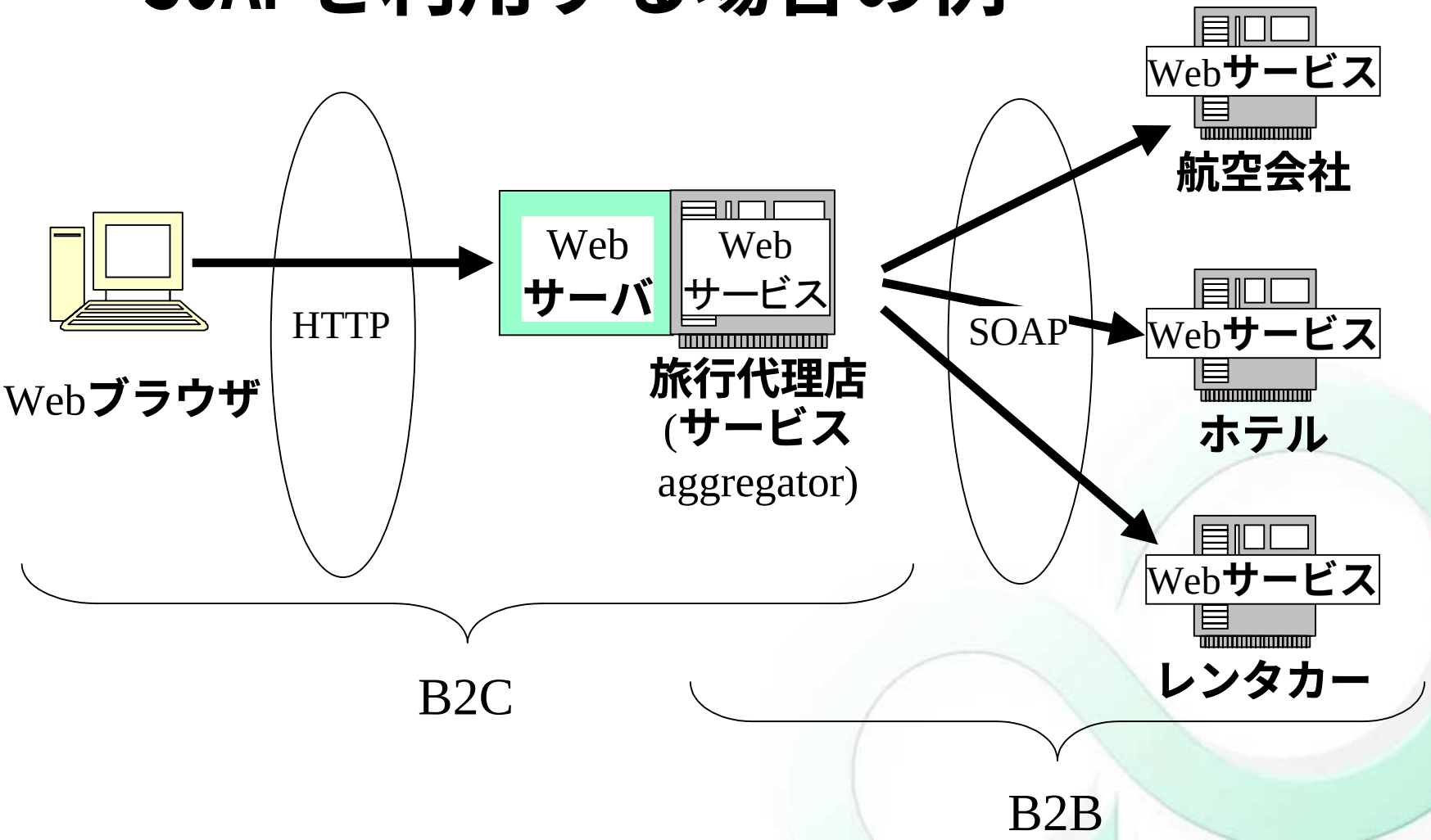
XML webサービス

XML分散システムの１実現形態

- Webサービス
  - = インターネット標準のプロトコルを利用してアクセス可能な，アプリケーションコンポーネント
- XML webサービス
  - = XMLでデータ交換を行うwebサービス

XML-Webサービスのアーキテクチャの一例

SOAPを利用する場合の例



XML Webサービスのアーキテクチャの一例

- SOAPにメッセージを乗せる
  - メッセージの内容はデータ，または
  - リモート呼び出し(RPC)のシグネチャ・引数・戻り値
    (現在は後者が多い)
- さらにHTTPにSOAPを乗せる

SOAP on HTTP の例

HTTP

SOAP

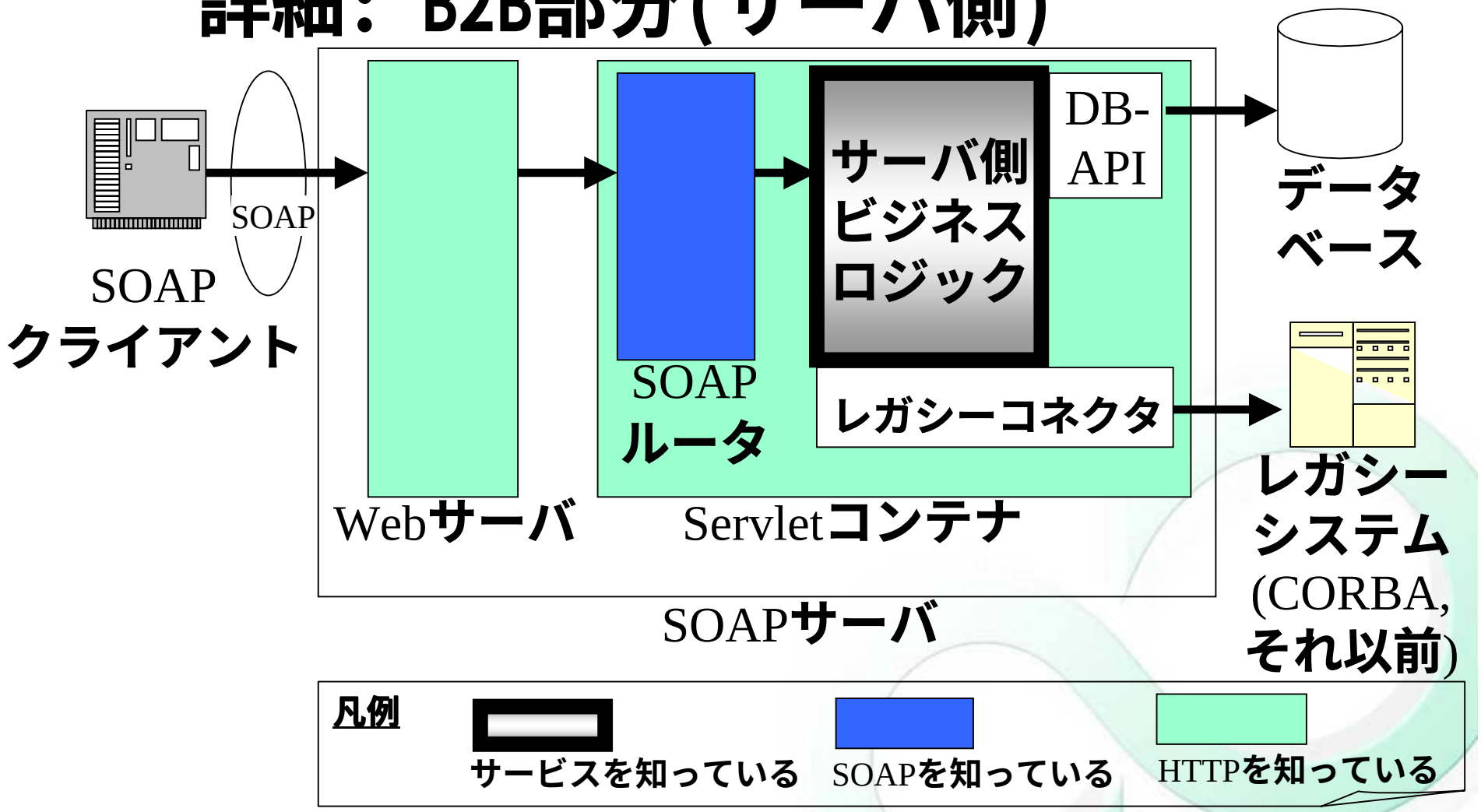
メッセージ

```
POST /some-service/SomeServicePort HTTP/1.1
Host: some-service-provider-a:8080
Content-Type: text/xml; charset=utf-8
Content-Length: 501
Connection: close
Accept-Encoding:
SOAPAction:

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:getImportantInfoByKeyword xmlns:ns1="http://someservice.org"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<keyword xsi:type="xsd:string">keyword</keyword>
</ns1:getImportantInfoByKeyword>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

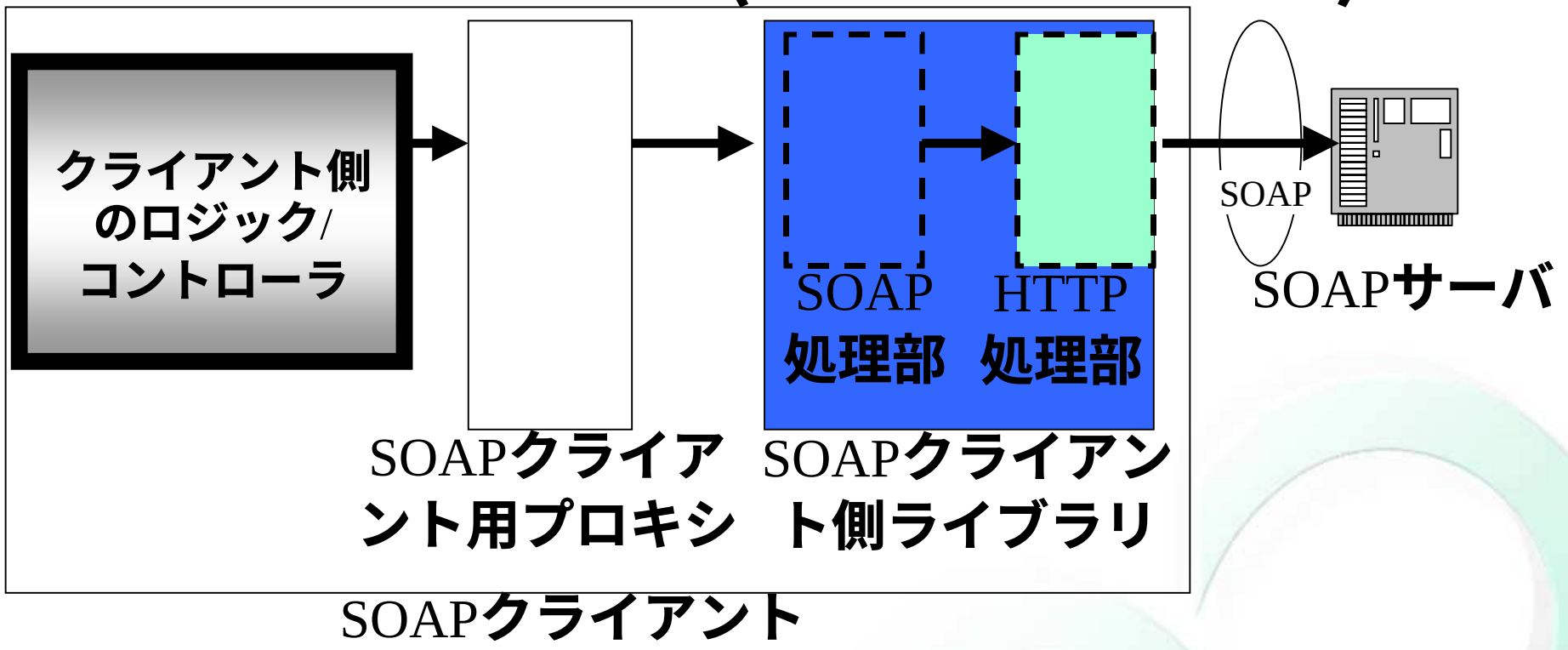
実装アーキテクチャの一例

詳細： B2B部分(サーバ側)



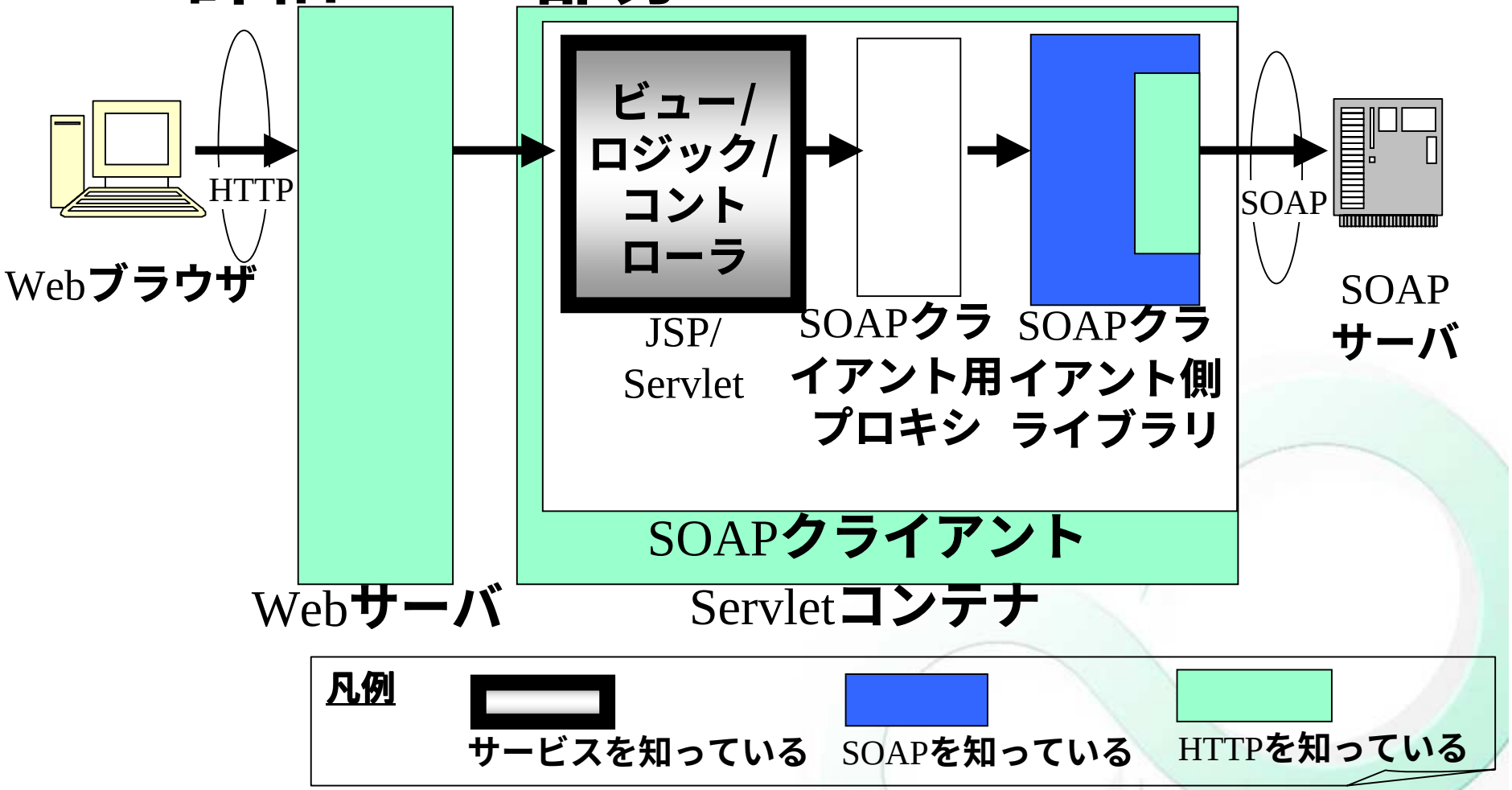
実装アーキテクチャの一例

詳細： B2B部分(クライアント側)



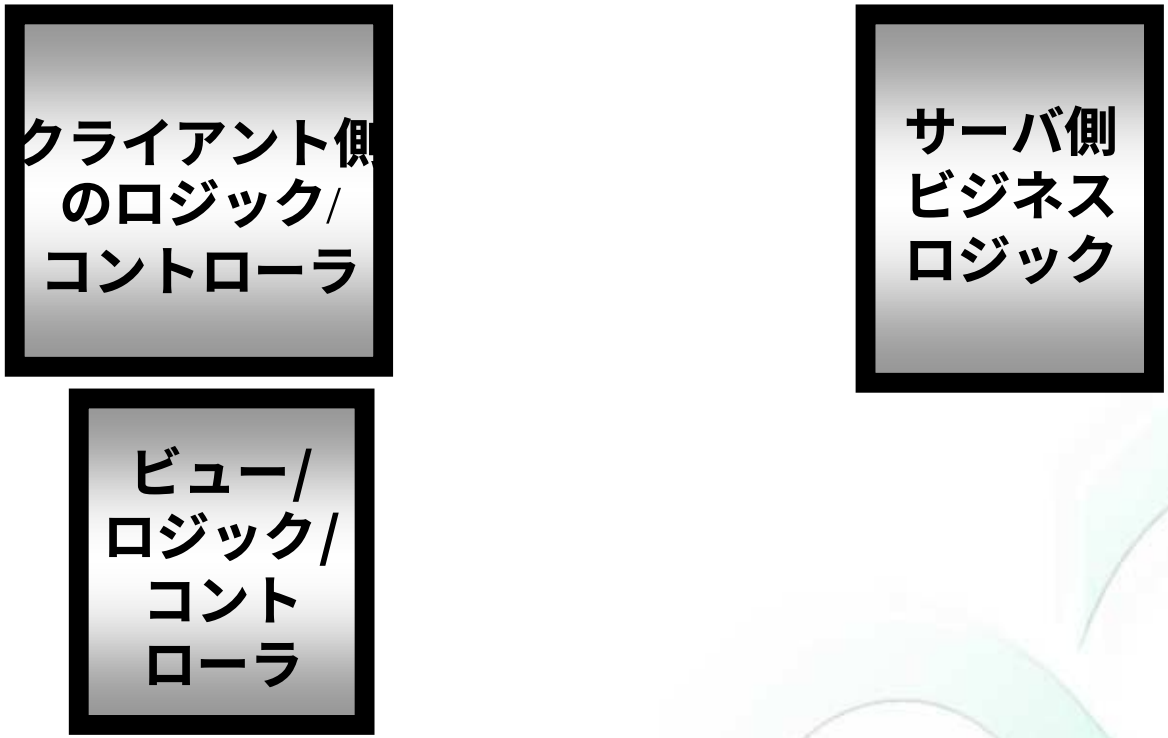
実装アーキテクチャの一例

詳細： B2C部分



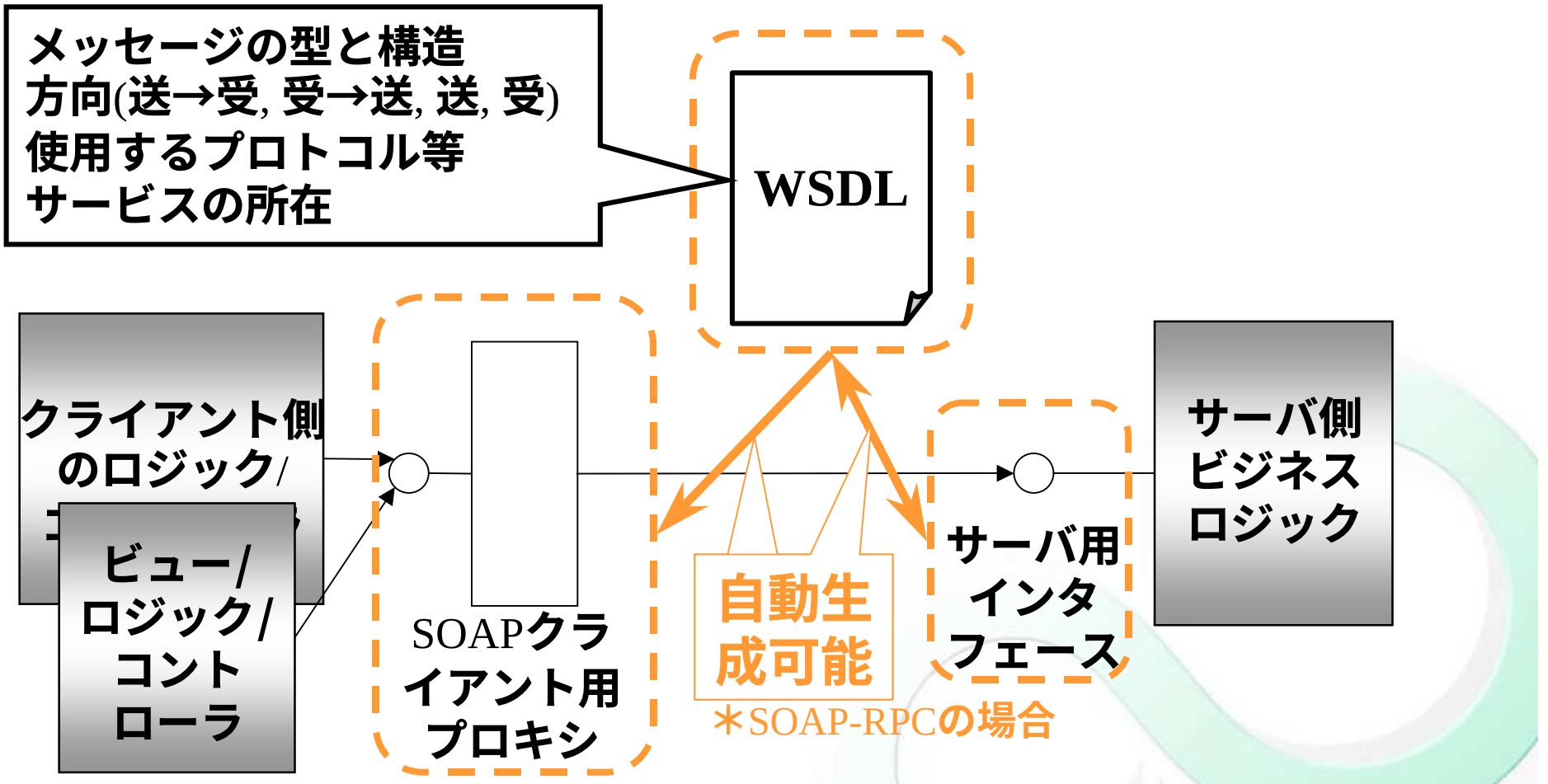
実装が必要な個所

クライアント側，サーバ側のロジック



インタフェースの記述

SOAPのインタフェースはWSDLで記述



サービスの創出

実装は比較的容易→何をサービスとするか?

Web サービス化できそうな内容

- **情報**
  辞書，天気予報，時刻表（駅前探険倶楽部)，配送状況 など
- **機能**
  web ページ情報検索（google)，商品受注，
  経路探索アルゴリズム（駅前探険倶楽部）など
- **Webサービスのaggregation**

演習 2

情報や機能の組み合わせで出来るサービス（思いついた例）を１つ挙げる

- サービスを提供するサーバは遠くにあり，反応が遅いかもしれない
  - →密に結合する必要があるサービスは避ける
  - ・例えばワードプロセッサ機能を提供するサービス（キーやマウスの操作ごとにメッセージをやりとり）
- できればクライアント側が人間ではなくコンピュータであるサービスを

Webサービス設計実装上のポイント

- Webサービスインタフェースの分析
- Webサービスのインテグレーション
- 他者から提供されるwebサービスとの連携
- UDDIの利用

Webサービスインタフェースの分析

使われ方を想定してインタフェースを定める

- 通常のオブジェクト指向による分析とやりかたは同じ
- このとき分析結果からwebサービスで送受信するXML電文の分析も行う

Webサービスのインテグレーション

ビジネスプロセスをどう実装するか

例えば旅行代理店の業務「お客様の旅行日程，行き先を伺い，チケットの手配，ホテル予約を行い，結果をお客様に伝える．」

Webサービスをつなげるプロセス書式:

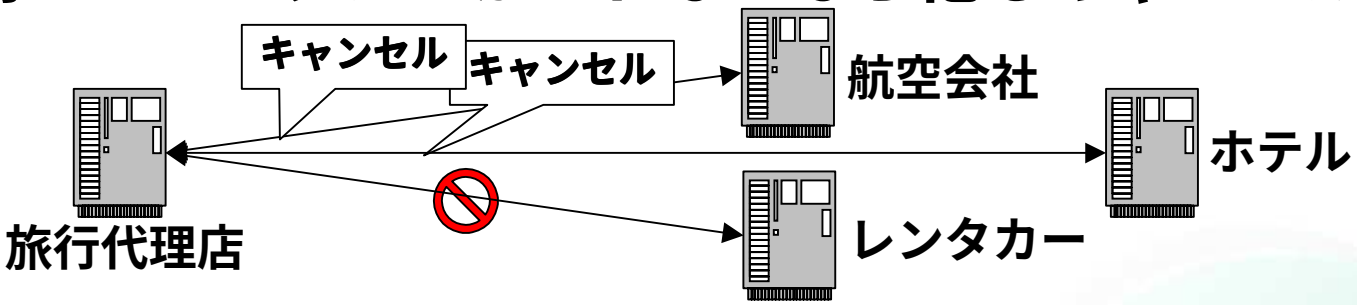
- BPEL4WS (Business Process Execution Language for Web Services)
  - Webサービス提供者の情報
  - WebサービスがXMLメッセージ
  - 処理の流れ
  - 処理が失敗したときの動作

  などを記述する

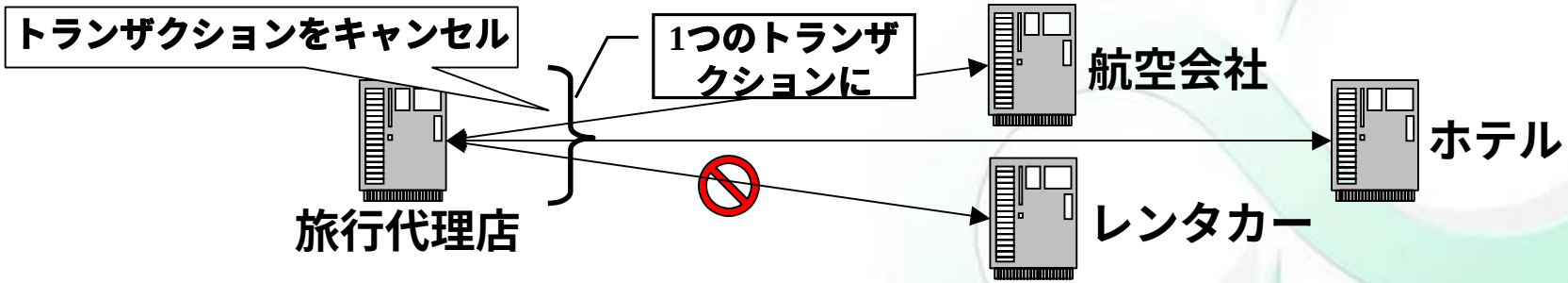
他者から提供されるwebサービスとの連携

分散している他のサービスとの振る舞いを調整するには

例：「レンタカーがとれないなら他もキャンセル」



→トランザクションを利用して，取り消し処理をトランザクション制御基盤に任せる



分散環境でのトランザクション

システムが疎結合

- トランザクション終了までの時間が長い
- 密結合システムのような厳格なトランザクション制御（例えば他のwebサービスのリソースのロック）は無理

Webサービス用のトランザクション

例えばロックの代りに代償トランザクション

仕様：

- WS-Coordination, WS-Transaction
- BTP (Business Transaction Protocol)

他者から提供されるwebサービスとの連携

Webサービスの発見

- ヒューマンリーダブルな情報から人間が発見
  公開仕様，IT関係のニュース，UDDI，...
- レジストリサービスから機械に発見させる
  UDDI

UDDIの利用

UDDIで登録できる内容

- 企業情報（企業名，分類情報(地理，業種など)，企業コード，説明文など)
- サービス情報（サービス名，分類コード，説明文など)
- 技術情報（サービスの所在地(URLなど)，サービスが準拠する仕様，メッセージングプロトコル，トランスポートプロトコル，データ形式など)

下線の項目：機械が検索しやすい形(IDや特定のフォーマットの文字列など)で登録される

演習 3

演習2で考えたサービスをUDDIに登録する．

1．このとき登録する項目を列挙する

2．検索する立場で，UDDIに足りないと考える項目を列挙する

UDDIの利用

現時点で有効なUDDI利用シナリオ

- アクセスポイント(URL)などの情報を使って
  ネットワークの物理的な変更に対応する

- ブラウザを使って人間が適切なサービスを
  検索しクライアントに情報を埋め込む

UDDIの利用

まず必要性を検討

- クライアント実行時に検索が必要か
  (接続先が移転する，サービス提供者が入れ替わる，etc)
- クライアントコンパイル時・デプロイ時に検索が必要か
  (適切なサービス提供者が複数いて選択可能である，etc)
- クライアントの分析・設計時に検索が必要か
  (利用できそうなサービスがUDDIに登録されている，etc)

プライベートUDDI

だれに公開するか，だれが公開した情報を利用するか，を限定

- **サービス提供者を審査してサービスの品質を保証**
- **特定のグループだけに公開することで，サービス利用者が適切なサービスを見つけやすくする**

その他の技術 (下位レイヤの技術)

下位レイヤでのセキュリティ

- 通信路の暗号化
- 通信先サーバ/クライアントの確認&認証

トランザクション管理，送達保証

- トランザクションマネージャの利用
- キューマネージャの利用

まとめ

- XML
  - アーキテクチャ
  - 設計実装上のポイント
    - XMLを操作するコードの書き方
    - どこまでXML形式で持ちまわるか
    - XMLのDB格納方式
    - XML電文設計
    - セキュリティ
- XML webサービス
  - アーキテクチャ
  - 設計実装上のポイント
    - Webサービスインタフェースの分析
    - Webサービスのインテグレーション
    - 他者から提供されるwebサービスとの連携
    - UDDIの利用

参考文献

Webページ

- http://www.infoteria.com/jp/xmlnote/
  基本仕様，応用技術の仕様書へのポータル
- http://www.google.co.jp/
  ここ経由で，たとえば developer works の解説資料へ...
- http://xml.fujitsu.com
  UDDI関連の最新動向など
- http://xml.apache.org
  APIマニュアル，仕様へのリンク．http://jakarta.apache.org も
- http://www.atmarkit.co.jp/
  必要なトピックがみつかりやすい

参考文献

書籍

- 改訂版 標準XML完全解説 上/下
  (ISBN4-7741-1186-4/ISBN4-7741-1302-6)
  XMLの各仕様の詳説