

# xml\_fdw

XMLファイルをPostgreSQLの情報源として扱う  
外部データラッパの試作

NTTソフトウェア 原田登志([@nuko\\_yokohama](https://twitter.com/nuko_yokohama))

# はじめに

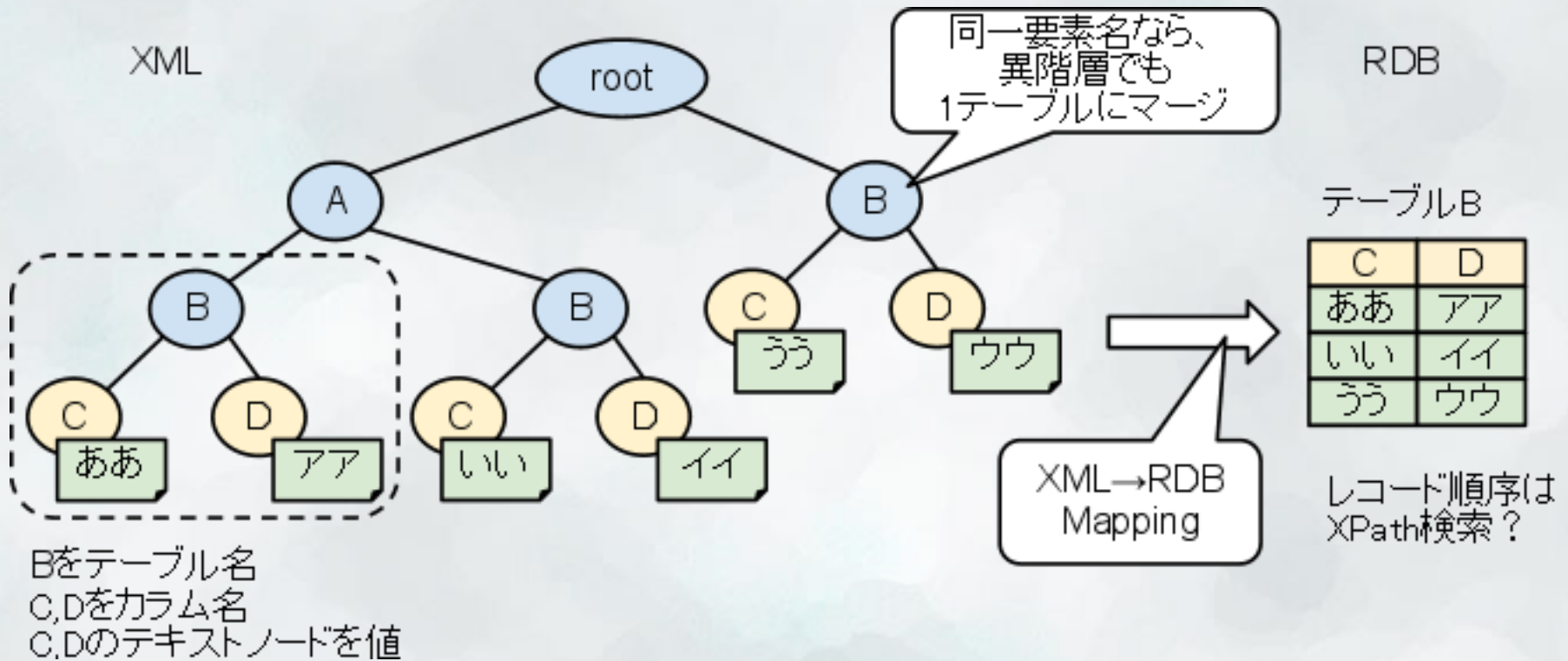
- PostgreSQL 9.1から導入されたForeign Data Wrapper (FDW)
  - 世間ではまだ実装例が少ない。
  - <http://pgxn.org/tag/fdw/>
- こういうものは早いもの勝ち。
  - 非RDBモデルのFDWもそろそろ出始めた。
    - CouchDB, Amazon S3, …
  - いずれはXMLモデルのFDWも出てくるはず？
  - 今、存在していないのなら試しに作ってみよう。

# Foreign Data Wrapperとは？

- Foreign Data Wrapper (FDW)とはSQL規格のSQL/MED (SQL Management of External Data)の一部。
- SQL/MEDとはDBMSの外にある様々なファイルや非リレーショナルデータベースに対してSQL文によって管理したり、アクセスを可能にしようというもの。
  - PostgreSQL以外のRDBMS (Oracle, MySQL, etc...)
  - RDBMS以外のDBMS
  - ファイル (テキストファイル, XML, etc...)
  - Web API (Twitter, Google API, etc...)
- PostgreSQL 9.1ではその中の一部である、外部テーブル (Foreign Table)をサポートしている。

# xml\_fdwの概要

- xml\_fdwはPostgreSQLサーバ上に置かれたXMLファイルを情報源として任意の中間要素と、その配下にある任意の末端要素を、テーブル・カラムにマッピングし、その結果をPostgreSQLテーブルとして見せる外部データラッパ。

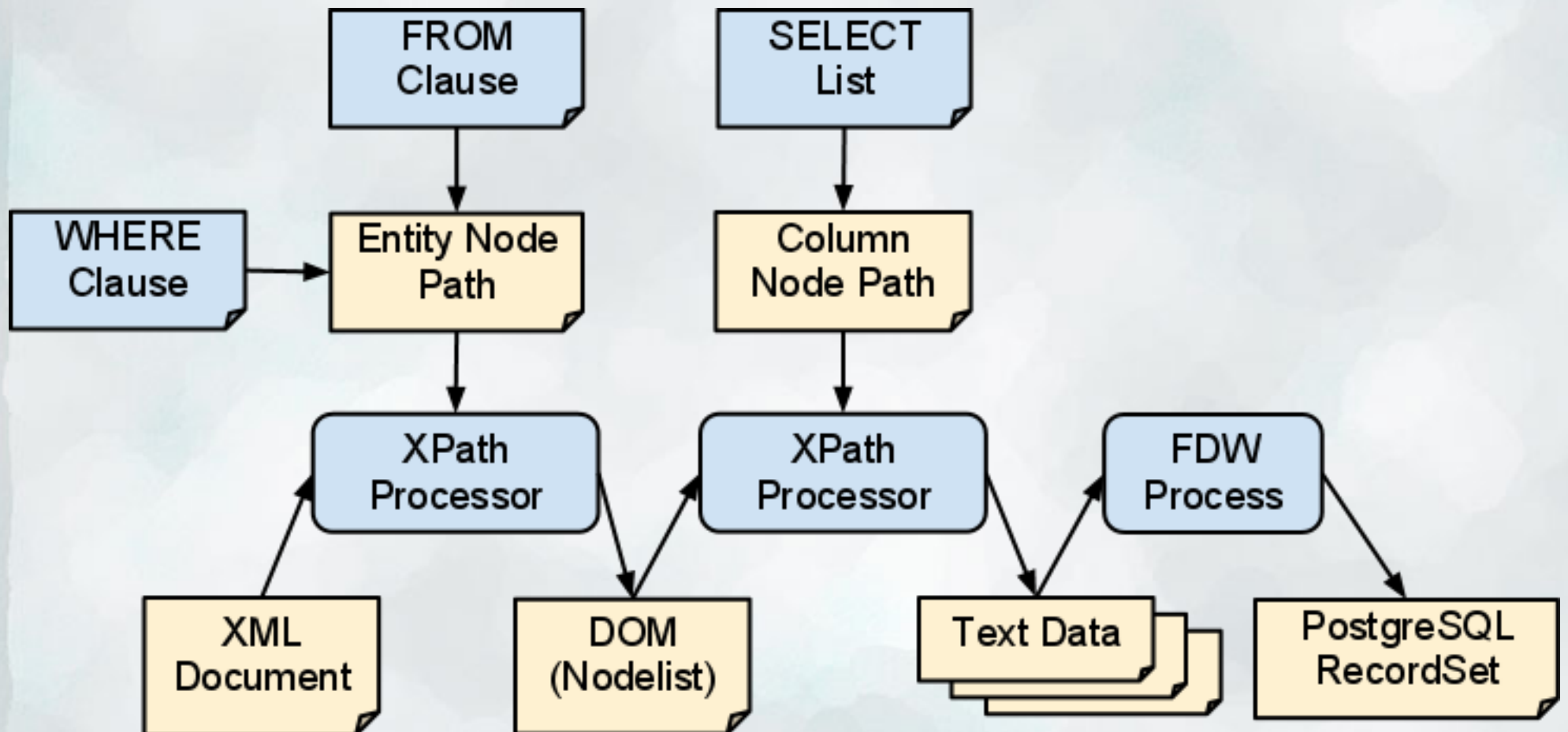


# xml\_fdwの作成目的

- このxml\_fdwの作成目的は主に自分の勉強のためです。
  - PostgreSQL fdwの勉強
  - libxml2によるXMLアクセスの復習
  - C言語の復習
- 現段階では、まだ全然、実用的なものにはなっていません。
- まずはシンプルにXMLファイルをターゲットにしますが、後々にXMLデータベースへも適用できれば・・・。

# xml\_fdwの処理イメージ

- 個々の処理をFDWのどのI/Fに割り当ててるのかを考える。
- XMLのパーズ、XPath処理はlibxml2のAPIを用いる。



# XML-RDBマッピングの概要

- XML文書＝データベース
  - 接続情報はXML文書ファイルのパス
  - 将来的にはデータソースのURLを指定して取得できると汎用性が増すか？
- 中間ノード＝テーブル
  - SQLのFROM句をここにマッピングする。
  - WHERE句をXPath-Conditionとして記述。
    - WHERE句情報を元に、XPathを自動生成出来ると理想だが、FDW内でWHERE句情報をどう取るのか要調査・・・(未実装)
- リーフノード(のテキストノード)＝カラム
  - SQLのSELECTリストをここにマッピングする。
- ORDER BY, GROUP BYなどはPostgreSQLに任せる。

# xml\_fdwによる外部サーバと外部テーブルの登録イメージ

- (fdw用関数登録、fdw登録は割愛。)
- 外部サーバ作成例

```
CREATE SERVER xml_doc
  FOREIGN DATA WRAPPER
xml_fdw
  OPTIONS (path '/tmp/foo.
xml');
```

← XML文書のパス

- 外部テーブル作成例

```
CREATE FOREIGN TABLE b (
  c text,
  d text)
  SERVER xml_doc
  OPTIONS (table '//b',
          columns '//c, //d')
```

← PostgreSQLに見せる  
テーブル名

← PostgreSQLに見せる  
カラム名

← テーブル検索用のXPath

← カラム検索用のXPath  
カラム数分、カンマで区切る。

# 実装手順

- まずは、file\_fdwのフォルダごとコピー
- ファイル名、ファイル内の関数名などをちまちまりネーム
  - file→xml
  - これだけでhandler登録までの関数は大体OK.
- validatorはOPTIONに合わせて修正要。
  - mysql\_fdwが良いサンプルになる。
- コールバック関数はもちろん修正が必要。
  - これもfile\_fdwよりmysql\_fdwのほうが参考にしやすいかも。
  - 特にタプルの生成と値の設定方法など。
- 次ページ以降に各コールバック関数の設計概要を示す。

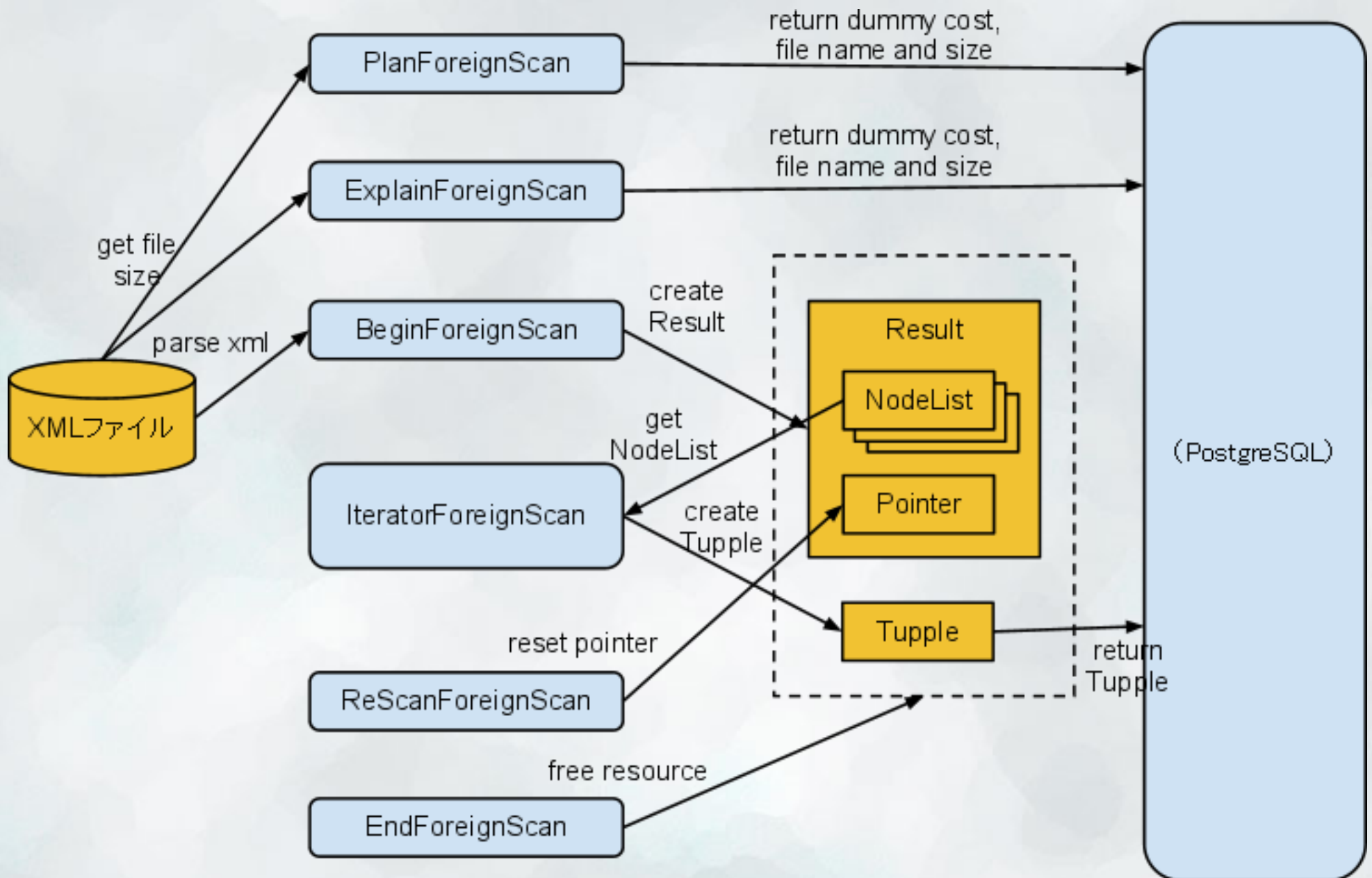
# 実装するコールバック関数の種類

- PostgreSQLのForeign Data Wrapperを実装するためには、以下の種類のコールバック関数を実装する必要がある。
  - PlanForeignScan
    - 外部テーブルスキヤンの実行計画作成
  - ExplainForeignScan
    - 詳細な実行計画作成
  - BeginForeignScan
    - 外部テーブルスキヤンの実行を開始
  - IterateForeignScan
    - 外部ソースから一行を取り出して返却
  - ReScanForeignScan
    - 先頭からスキヤンを再開
  - EndForeignScan
    - スキヤンを終了しリソースを解放
- 「[50.2. 外部データラッパのコールバックルーチン](#)」を参照

# xml\_fdwでのコールバック関数実装概要

- xml\_fdwでは各コールバック関数で以下の処理を実装した。
  - PlanForeignScan
    - ダミーのCostを設定とXML文書のパスとサイズを設定
  - ExplainForeignScan
    - (PlanForeignScanと同じ)
  - BeginForeignScan
    - XML文書のパース、レコード単位のノードリスト情報取得
    - XPath関数によってドキュメントからノードリストを取得
  - IterateForeignScan
    - ノードリストからカラム単位の情報を取得
    - XPath関数によってノードからテキストノードを取得
  - ReScanForeignScan
    - 内部構造体のポインタを先頭に設定
  - EndForeignScan
    - リソースの解放

# xml\_fdwでのコールバック関数実装概要



# 各コールバック関数の実装

## PlanForeignScan

- PlanForeignScan
  - 外部テーブルへのスキヤンの実行計画を作成
  - xml\_fdwではダミーのコスト(startup\_cost, total\_cost)を設定するのみ。
    - file\_fdwのコードをほぼそのまま流用。
    - 本来は「XML文書アクセスのコスト」とは何か、から検討要

# 各コールバック関数の実装

## ExplainForeignScan

- ExplainForeignScan
  - 外部テーブルスキャンの追加のEXPLAIN出力を表示
  - xml\_fdwでは以下のように実装する。
    - XML文書のパス(optionで指定したパス)を表示する。
    - (COSTS OFF)が指定されなければ、XML文書ファイルのサイズを表示する。
    - ↑ file\_fdwのパクリ。
    - もうちょっと頑張るならXMLをパースしてXMLの要素数とかの情報を出したほうがいいのかな・・・

# 各コールバック関数の実装

## BeginForeignScan (1/4)

- BeginForeignScan
  - 外部テーブルスキュンの実行を開始
  - xml\_fdwでは以下の処理を行う。
    - GetOption関数でCREATE SERVERで指定した、pathオプションの値を取得する。
      - pathオプションの指定がなければ、ereport() にpath未指定の旨のメッセージを設定して呼び出す。
    - XML文書をlibxml2でパースする。
      - パースに失敗した場合 (pathに指定した文書がない、XMLでない等) はereport() にXMLファイル失敗の旨のメッセージを設定して呼び出す。
    - (つづく)

# 各コールバック関数の実装

## BeginForeignScan (2/4)

- BeginForeignScan

- xml\_fdwでは以下の処理を行う。(つづき)

- GetOption関数でtableオプションとcolumnsオプションの値(=取得用XPath文字列)を取得する。
  - この方式だと、XPath-conditionによる絞り込みや、カラム名とXML内の要素名とのマッピングが可能になる。
  - でも、オプションなしなら、CREATE FOREIGN TABLE定義中のテーブル名とカラムの情報を元にXPathを自動生成できるのが理想？でも、取得方法が分からない・・・。
  - FOREIGN TABLEのrelation oidからシステムカタログをゴリゴリ漁って調べればいいのか・・・？
- (つづく)

# 各コールバック関数の実装

## BeginForeignScan (3/4)

- BeginForeignScan

- xml\_fdwでは以下の処理を行う。(つづき)

- tableオプションに指定したXPathを実行し、レコードセット相当の中間ノード群を取得する。
- Documentに対して、tableオプションに指定したXPathにより中間要素を取得
  - XPathにヒットする要素が複数存在した場合には、その数分のレコードを生成する。
  - XPathでヒットするが存在しない場合、0件 (NOT FOUND) とみなす。
  - 変なXPathが設定されたら動作保証外...
- (つづく)

# 各コールバック関数の実装

## BeginForeignScan (4/4)

- BeginForeignScan

- xml\_fdwでは以下の処理を行う。(つづき)

- 情報引継用の構造体をnode->fdw\_stateに設定する。
- 情報引継用の構造体には以下の情報が管理されている。
  - tableオプションに指定したXPathによって取得されたノード群
  - カレントポインタ
  - tableオプションで取得されたノード数
    - nodes->nodeNr で取得可能だった。
  - columnsオプションで指定されたカラム取得用XPath文字列群
    - カンマセパレータで今回は指定するので、カンマでばらす処理が必要。

# 各コールバック関数の実装

## IterateForeignScan(1/3)

- IterateForeignScan

- 外部ソースから一行を取り出して、それをタプルテーブルスロットに設定して返却。
- xml\_fdwでは以下の処理を行う。
  - node->fdw\_stateから構造体Xを取得。
  - 情報引継用の構造体のカレントポインタをインクリメントして、カレントポインタが指し示すノード群配列の1要素を取り出す。
    - これが1レコード分の情報にあたる。
    - (各レコード分の処理は次ページへつづく)
  - カレントポインタがノード数に達したら、EODとみなしてExecClearTuple(slot); でクリアしたslotを返却する。
    - どうやら、これを渡すと以降、IterateForeignScanが呼び出されなくなるようだ。

# ExecClearTuple

- この関数はPostgreSQL本体の関数。
- 以下の場所にあった。
  - src/backend/executor/execTuples.c (447行目)
- 関数コメント
  - This function is used to clear out a slot in the tuple table.
  - 関数名のとおり、タプルスロットをクリアするための関数らしい。
- おそらく、これでクリアしてからタプルスロットに値をセットするのが作法なのだと思う。
- また、mysql\_fdwの使い方を見ると、もう返却する結果がない場合(検索終了時)にはクリアした状態のタプルを渡しているように見える。
  - この解釈が本当に正しいのか？

# 各コールバック関数の実装

## IterateForeignScan(2/3)

- IterateForeignScan (つづき)

- レコード単位の処理

- 取得した1レコード相当の中間要素に対し、columnsオプションから指定された複数のXPathを順次を実行する。
  - 今回はstrtokでXPathを分解・取り出した。
- 属性・要素自体がなければ、そのカラムはNULLと見なす。
- 属性・要素があればその属性値・テキストノードをカラム値とみなす。空要素の場合、空文字列の値として扱う。
- columnsオプションで指定したXPath結果とCREATE FOREIGN TABLEで指定したカラムが不整合の場合には動作保証外
- 該当するカラム要素が複数存在した場合、全要素のテキストノードを連結した値をカラム値とみなす。

# 各コールバック関数の実装

## IterateForeignScan(3/3)

- IterateForeignScan (つづき)
  - レコード単位の処理
    - 取得したカラム値を引数として、BuildTupleFromCStrings()で1カラム分の情報をセットする。
    - 全カラム数分、カラム値をセットしたら、ExecStoreTuple()を発行してタプルを格納する。
    - ExecStoreTupleにセットしたslotを返却する。

# BuildTupleFromCStrings

- この関数はPostgreSQL本体の関数
- 以下の場所にあった。
  - src/backend/executor/execTuples.c (1083行目)
- 関数コメント
  - BuildTupleFromCStrings - build a HeapTuple given user data in C string form. values is an array of C strings, one for each attribute of the return tuple. A NULL string pointer indicates we want to create a NULL field.
  - CString型のユーザデータをこの関数に与えてタプルを生成する関数だと思われる。
  - NULL値をセットする場合、この関数にもNULLをセットする。

# ExecStoreTuple

- この関数はPostgreSQL本体の関数
- 以下の場所にあった。
  - src/backend/executor/execTuples.c (328行目)
- 関数コメント
  - This function is used to store a physical tuple into a specified slot in the tuple table.
  - この関数は、タプルテーブルの指定されたスロットに物理的なタプルを保存するのに用いる。
    - コメントだけだと正直、きちんと理解しきれない・・・
  - 今回はmysql\_fdwのやり方を真似て、BuildTupleFromCStrings()の後にExecStoreTuple()を発行した。

```
tuple = BuildTupleFromCStrings
(TupleDescGetAttInMetadata(
    node->ss.ss_currentRelation-
>rd_att), values);
ExecStoreTuple(tuple, slot,
InvalidBuffer, false);
```

# 各コールバック関数の実装

## ReScanForeignScan

- ReScanForeignScan
  - 先頭からスキャンを再開。
  - xml\_fdwでは、以下のように実装。
    - 情報引継用の構造体のカレントポインタを初期値に戻すのみとする。

# 各コールバック関数の実装

## EndForeignScan

- EndForeignScan
  - スキャンを終了しリソースを解放。
  - xml\_fdwでは、以下のリソースを解放。
    - パースしたXML文書(Document)を解放する。
      - 他にlibxml2で生成した各種リソースなども。
    - 情報引継用の構造体の解放

# 動作検証(1/5)

- 検索対象とするXMLはスライド3に示したものの。
- このXMLファイルを /tmp/foo.xml に置く。
- 外部サーバ定義と外部テーブル定義を以下のように行う。

```
CREATE SERVER foo
  FOREIGN DATA
  WRAPPER xml_fdw
  OPTIONS (path
' /tmp/foo.xml' );
```

```
CREATE FOREIGN TABLE
b (
  c text,
  d text)
  SERVER foo
  OPTIONS (table
' //b',
```

```
columns ' //c
```

# 動作検証(2/5)

## ● SELECT文の実行例

```
SELECT * FROM
```

```
b;
```

c	d
---	---

ああ	アア
いい	イイ
うう	ウウ
ええ	エエ

(4 rows)

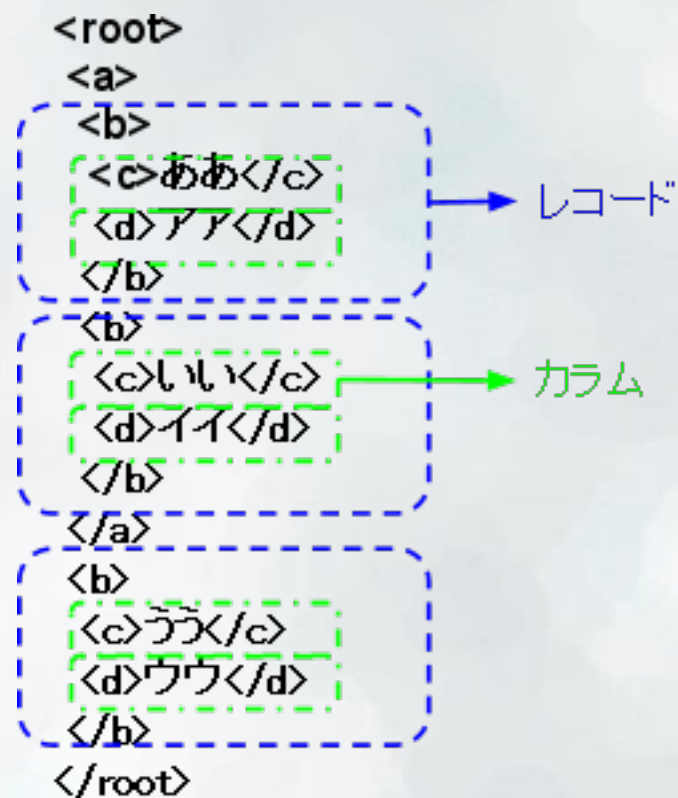
```
SELECT c FROM
```

```
b;
```

```
c
```

ああ
いい
うう
ええ

(4 rows)



# 動作検証(3/5)

## ● SELECT文の実行例

```
SELECT * FROM b WHERE c = 'うう';
```

c	d
うう	ウウ

(1 row)

```
SELECT * FROM b ORDER BY c DESC ;
```

c	d
ええ	エエ
うう	ウウ
いい	イイ
ああ	アア

(4 rows)

# 動作検証(4/5)

## ● EXPLAINの実行例

```
EXPLAIN SELECT * FROM b WHERE c = 'うう';  
QUERY PLAN
```

```
-----  
-----  
Foreign Scan on b (cost=0.00..1.21  
rows=1 width=64)  
  Filter: (c = 'うう'::text)  
  Foreign Xml: /tmp/foo.xml  
  Foreign Xml Size: 199  
(4 rows)
```

```
EXPLAIN ANALYZE SELECT * FROM b WHERE c =  
'うう';
```

```
QUERY PLAN
```

```
-----  
-----  
-----  
Foreign Scan on b (cost=0.00..1.21  
rows=1 width=64) (actual time=0.101..  
0.133 rows=1 loops=1)  
  Filter: (c = 'うう'::text)  
  Foreign Xml: /tmp/foo.xml
```

# 動作検証(5/5)

- UPDATE/TRUNCATE/CREATE INDEXは動作しない。
  - FOREIGN TABLEはPostgreSQL管理上、relationの種別が'b'
  - 'b' のrelationに対する更新やインデックス生成はエラーとなる。

```
UPDATE b SET c = 'んん';  
ERROR: cannot change foreign table "b"  
STATEMENT: UPDATE b SET c = 'んん';
```

```
TRUNCATE TABLE b;  
ERROR: "b" is not a table  
STATEMENT: TRUNCATE TABLE b;
```

```
CREATE INDEX b_c_idx ON b (c);  
ERROR: cannot create index on foreign  
table "b"  
STATEMENT: CREATE INDEX b_c_idx ON b  
(c);
```

# 開発工数

- xml\_fdwをつくるのに費やした時間。
  - コンセプト ⇒ 3分
  - 設計 ⇒ 3時間
  - 実装(コピーや置換作業含む) ⇒ 4時間
    - 先達のソースを参考にしながら試行錯誤。
    - mysql\_fdwは非常に参考になった。
  - 試験(導通レベル) ⇒ 2時間
  - ソースコードの実効ステップ ⇒ 約400 Step

# 作ってみて思ったこと

- FDW開発用ガイドラインが欲しい！
  - 今回ははfile\_fdwやmysql\_fdwなど、先達の作ったソースをベースに試行錯誤しながら開発。
  - 初心者向けにもう少し詳しく書いて欲しいこと。
    - コールバック関数間の情報引継ぎ方法
    - タプルの生成 & 値の設定方法
- ユーティリティライブラリが欲しい！
  - オプション取得機能などは、汎用化したサポートライブラリみたいなものが欲しいかも…
- 単体デバッグ環境が欲しい！
  - PostgreSQLに組み込む前にコールバック関数単体での動作検証が出来る環境があると嬉しいかも。

# TODO

- ソースの公開
- XPathがヒットしないときにNULL値を設定する処理。
- 属性対応。NodeType判定 +  $\alpha$  の処理を追加。
- 名前空間への対応
  - xpathとセットでnamespace(prefix, url)リストを渡す？
- カラム取得用XPathで複数ヒットしたときの扱い。
  - 配列で返却するというオプションがあってもいいのかも？
- XML文書のデータソース指定の柔軟化。
  - URL指定、XMLデータベースへの拡張など。
- カラム取得用XPath生成に使う情報の定義
  - 現状は、CREATE FOREIGN TABLEのOPTIONでXPath式を直接指定。
  - カラム名・型定義情報から本当は取得したいところ。
- XPathコンテキストのキャッシュ(生成コスト削減)
- WHERE句のXPath-pushdown。
- XML文書に対するコスト計算・EXPLAIN仕様の検討

# 別のxml\_fdw案

- 任意のXQueryやXSLTを記述させて、プロセッサに食わせて、ResultSetを表現する規定のXMLを生成させるやりかたも考えられる。
- optionに指定するのもXQuery式やXSLT文書のみになる。
  - namespace問題もXQuery内やXSLT内で解決できるかも。
  - ただ、WHERE句のpush-downが困難になるかも。
  - XQueryを使うことで複数のXML文書の結合結果をPostgreSQLで扱えるかも。
- 実装は楽になるけど、使う側にXQueryやXSLTの知識を要求することになるのでイマイチか・・・？

# 他のFDW(まだ構想レベル)

- 列指向データベースとの連携ラッパ
  - MonetDBというオープンソースの列指向DBMSがある。
  - このDBMSは集約計算が高速というのが売り。
- PostgreSQLなどのRDBMSでは高速な処理が苦手な、集約計算を、FDW経由でMonetDBに行わせ、結果だけをPostgreSQLで処理できないか。
  - OLTP系の処理はPostgreSQLに任せる。
- 列指向DBMSはOLTP系が苦手なので、適材適所でPostgreSQLとMonetDBを組み合わせて、ポータルとしてPostgreSQLを使うようなイメージ。

# 参考

- xml\_fdwの検討/実装にあたっては以下のリソースを参考にした。
- PostgreSQL 9.1.0文書 第50章外部データラッパの作成
  - <http://www.postgresql.jp/document/current/html/fdwhandler.html>
- contrib/file\_fdw
  - <http://www.postgresql.jp/document/current/html/file-fdw.html>
- mysql\_fdw
  - [http://pgxn.org/dist/mysql\\_fdw/](http://pgxn.org/dist/mysql_fdw/)
- libxml2
  - <http://xmlsoft.org/>