AITC
先端ＩＴ活用推進コンソーシアム

ProjectLA

バックエンドの技術解説

RDFを使った三つ組みデータの格納

2013/03/14

クラウド・テクノロジー研究部会 リーダー

荒本道隆

（アドソル日進株式会社）

何故、RDFか？

- 断片的なデータを相互につなぎたい
  - RDFは主語・述語・目的語の三つ組構造で表現
  - 目的語と主語に同じ値を設定して、それぞれをつなぐ
- 属性を事前に決定できない
  - RDFはスキーマレスなので、柔軟に対応できる
  - RDFは多様な情報を関係性で表現できる
- 大量のデータを蓄積・分析したい
  - RDFは構造が単純なので、コンピュータ・パワーが活きる

バックエンドの概要

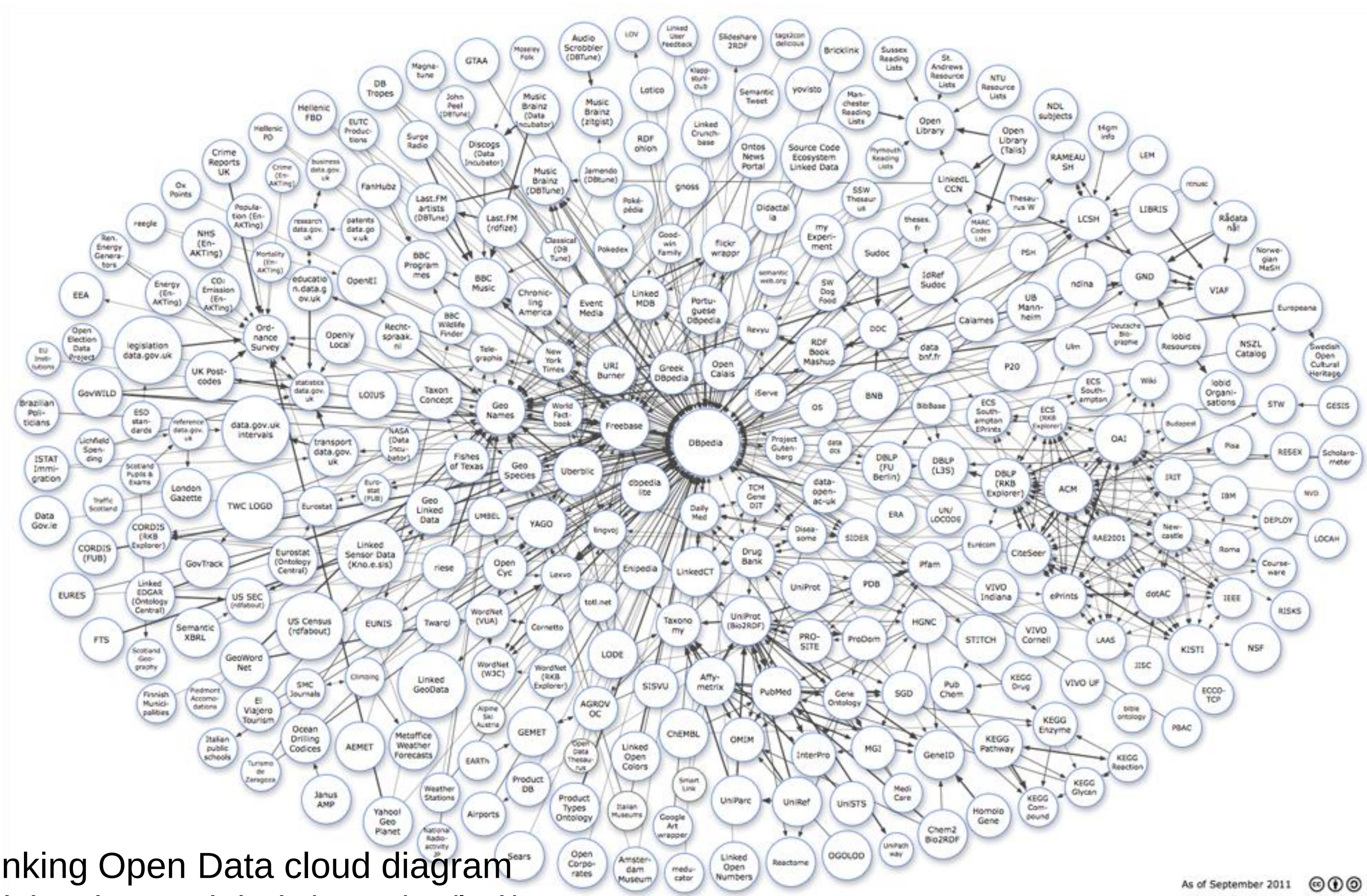
- アプリに、JSON+HTTPを使ったAPIを提供
- 格納はRDF形式
- 検索はSPARQL（RDFクエリ言語）を使用

RDFとは

- RDF：Resource Description Framework
  - 2008年にW3C勧告となった
  - 代表的な使用例
    - RSS（RDF site summary）0.9, RSS1.0
    - **DBpedia**
- 主語、述語、目的語の3つの要素で表現
  - 主語（Subject）：URI
    - http://aitc.jp/project/2013/projectLA/person/1
  - 述語（Predicate）：URI
    - http://aitc.jp/project/2013/projectLA/person#name
  - 目的語（Object）：値
    - 「先端太郎」

RDFで複雑なデータを表現

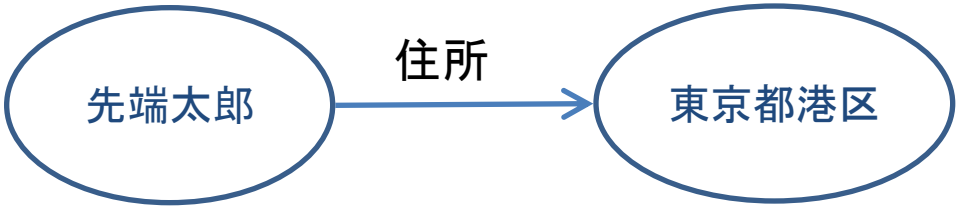
- 目的語（Object）と主語（Subject）に同じ値を入れる



The Linking Open Data cloud diagram
http://richard.cyganiak.de/2007/10/lod/

モデルをRDF化する上での注意

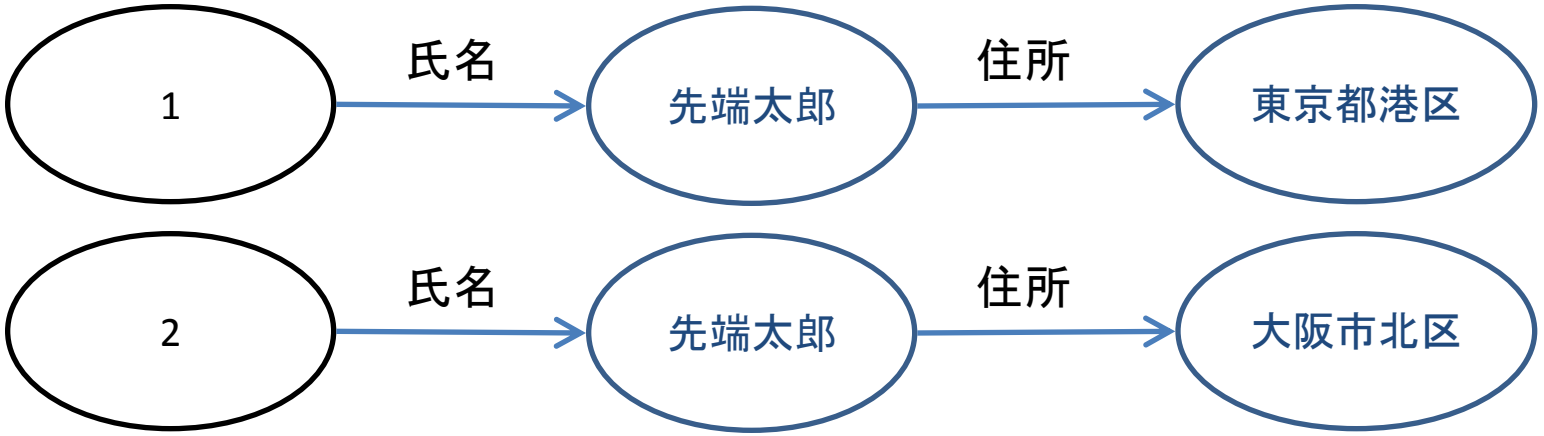
- この部分をRDFにする場合



- 識別できるようにIDを追加

IDを振っただけだと

- これをそのままRDFにすると問題がある



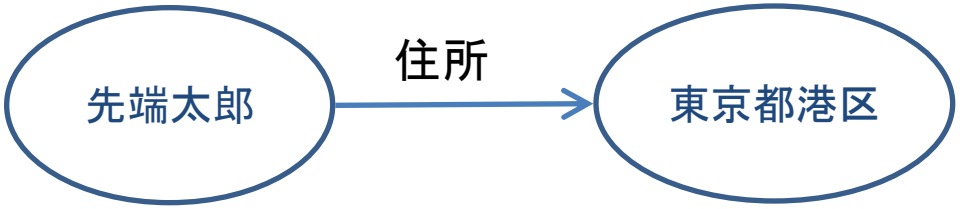
- データがこうなる

主語	述語	目的語
1	氏名	先端太郎
先端太郎	住所	東京都港区

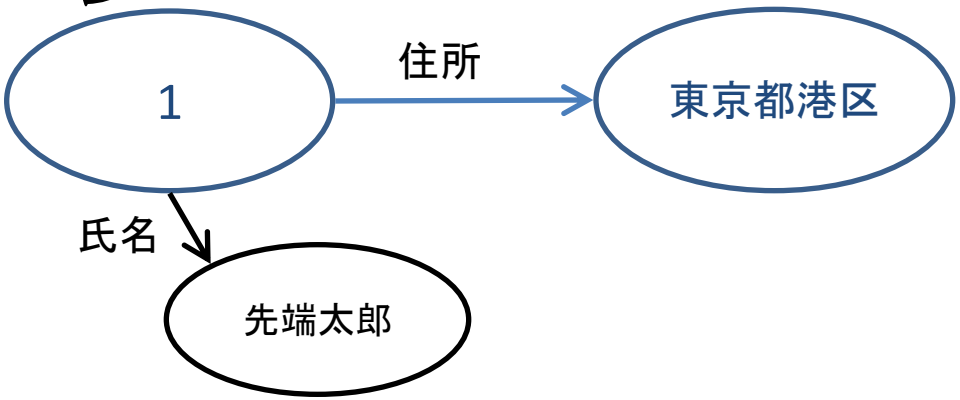
  - 同姓同名が居ると、区別がつかない

区別可能にするには

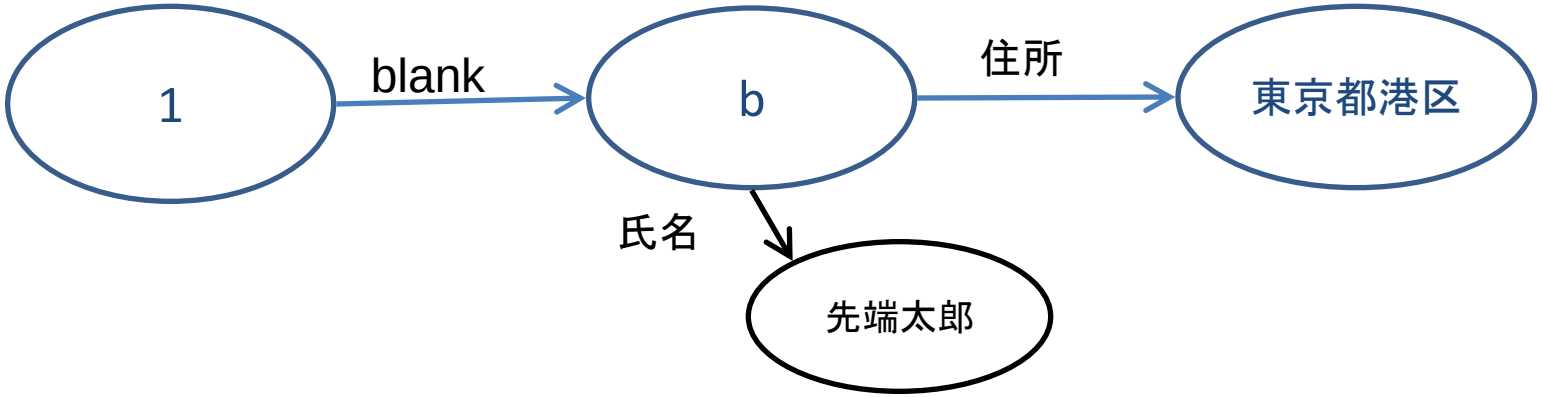
- 元



- IDを追加するパターン

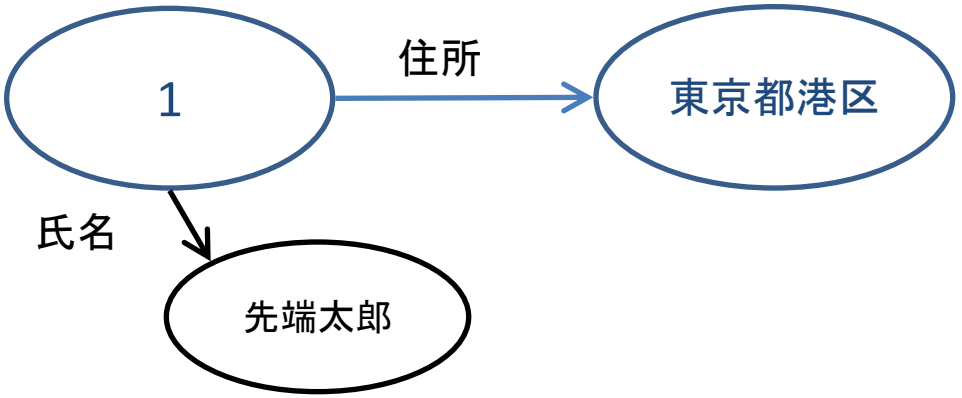


- IDと空白ノードを追加するパターン



IDを追加するパターン

- モデル

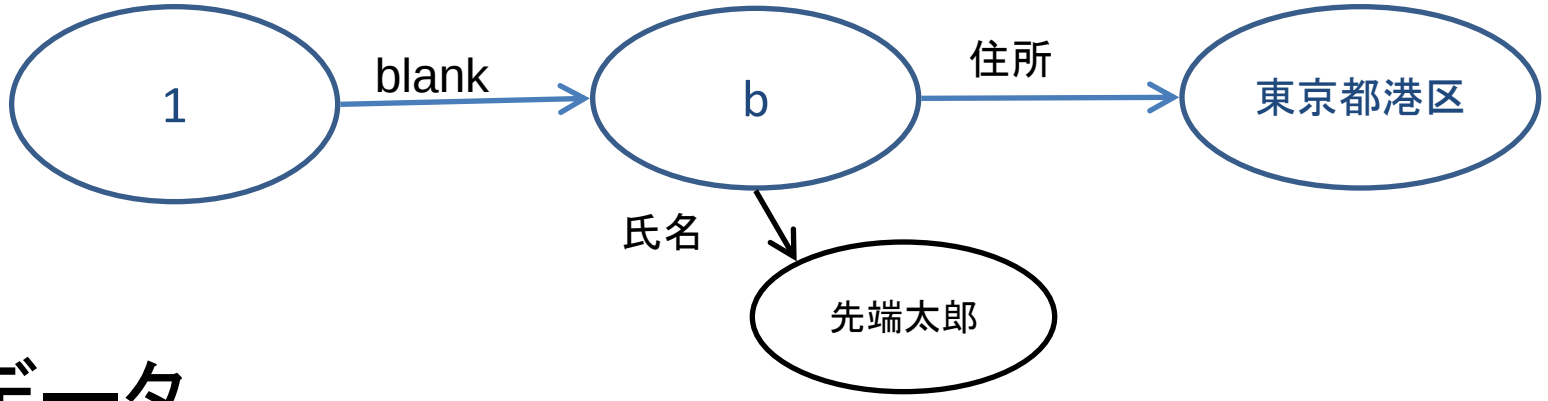


- データ

主語	述語	目的語
1	氏名	先端太郎
1	住所	東京都港区

IDと空白ノードを追加するパターン

- モデル



- データ

主語	述語	目的語
1	Node	b1
b1	氏名	先端太郎
b1	住所	東京都港区

RDFとNoSQLの比較

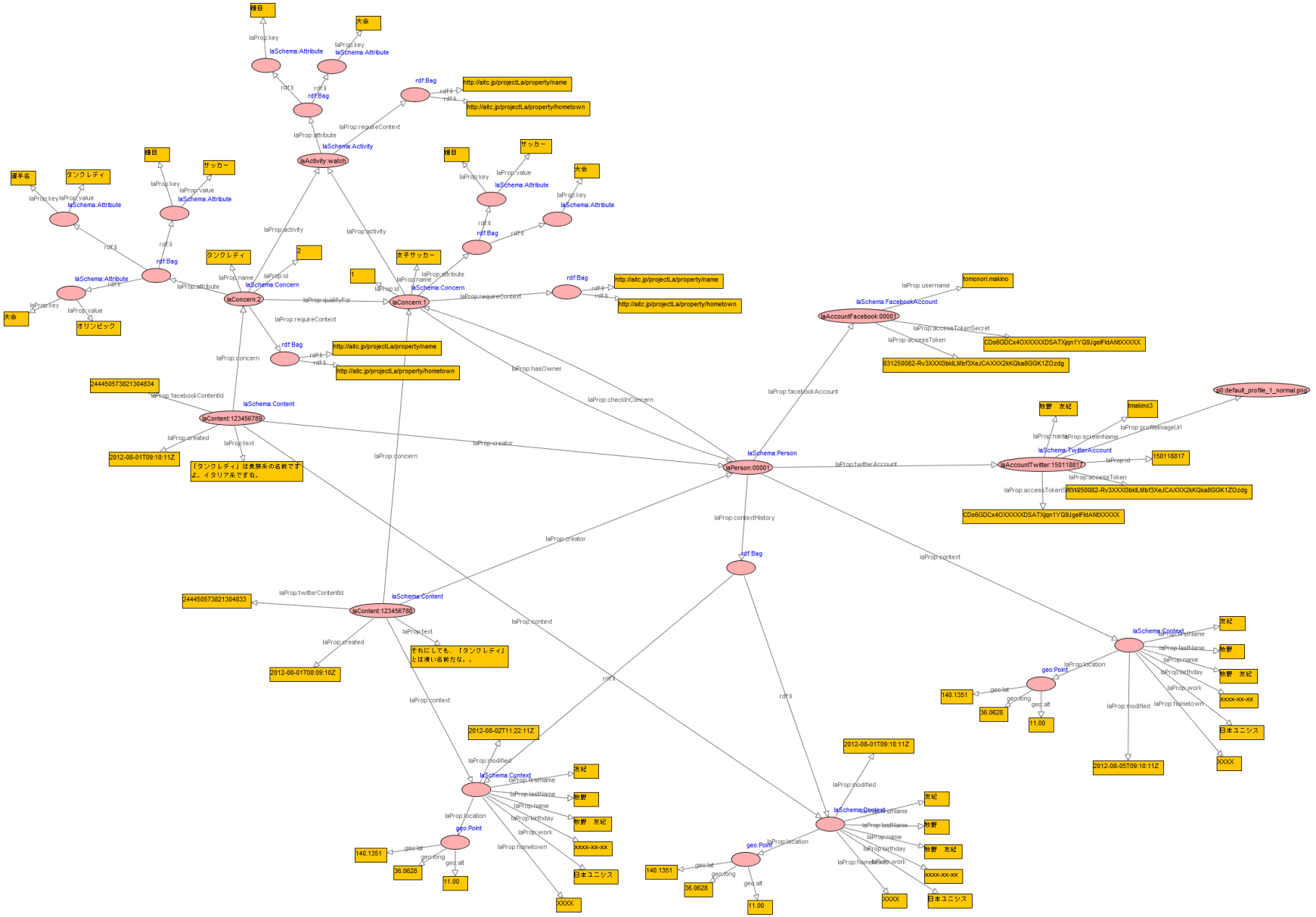
- RDFとCassandraの比較
  - 主語（Subject）←KEY
  - 述語（Predicate）←Column
    - **どちらもスキーマレス**
    - **どちらも主語（Key）＋述語（Column）でユニーク**
  - 目的語（Object）←Value

  - 主語の名前空間←Column Family
    - 名前空間を変えることで、異なるデータを混在
      - http://aitc.jp/project/2013/projectLA/person/1
      - http://aitc.jp/project/2013/projectLA/content/1

RDFの実装：Jena

- Apacheのプロジェクトの１つ
  - http://jena.apache.org/
- Java製フレームワーク
  - Semantic Webアプリケーションを構築
  - RDFデータを読み、書き、処理するためのAPI
    - ファイルシステム
    - RDB（Oracle, MS-SQLServer, DB2, PostgreSQL, MySQL, Derby, H2, HSQLDB）
  - RDFとOWLを使ったルールベースの推論エンジン
    - OWL：Web Ontology Language
  - SPARQLのクエリーエンジン
  - RDFデータを公開するためのサーバ

ProjectLAのRDFモデル（の一部）



抽象化したAPIを作成

- やり取りするフォーマット
  - JSON形式
    - JSON⇔RDFの変換ライブラリを作成
    - この変換ライブラリの出来がキーになりそう
- インターフェイスはREST（HTTP）
  - 細かい部分はSenchaのデフォをそのまま採用
    - 悩む手間が省ける
    - APIの種類はそんなに多くないはず
    - 検索は様々なフィルタ条件が必要

データを格納するAPI－１

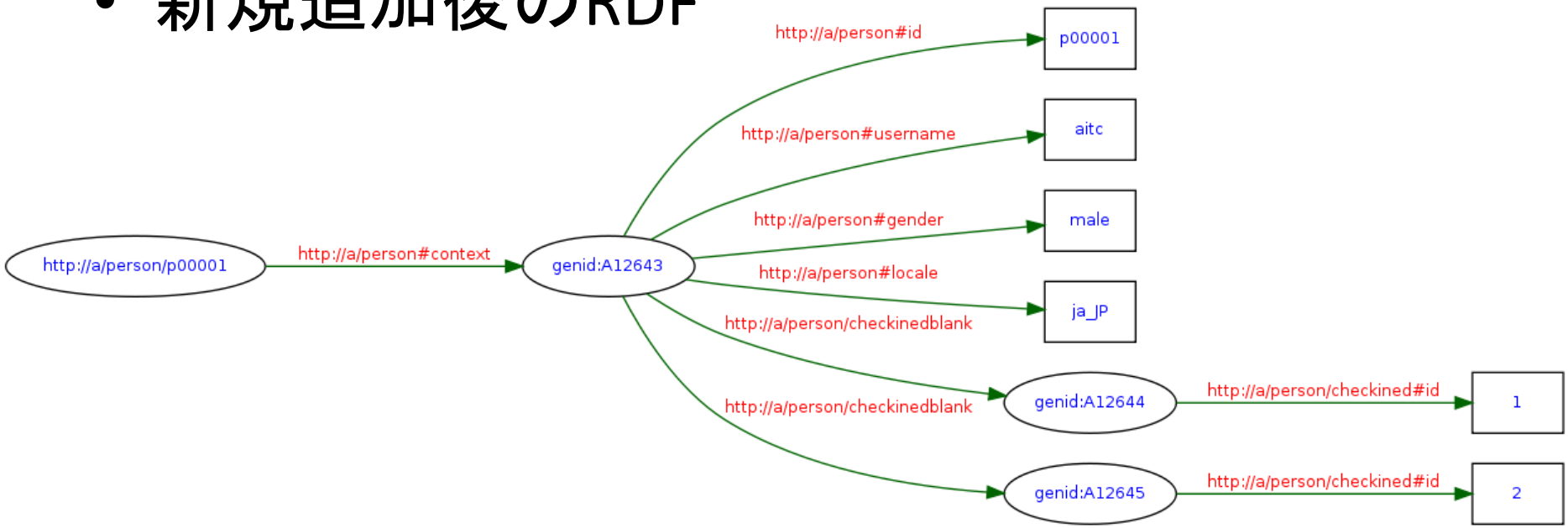
- 格納
  - JSONをPOSTする
    - 内部でJSON→RDFに変換
  - 登録と更新の区別
    - Idが無い：新規登録
    - Idがある：更新

データを格納するAPI－２

- JSON

```
{
    "person": {
        "username": "aitc",  "gender": "male",  "locale": "ja_JP",
        "checkined": [
            {"id": "1"}, {"id": "2"},
        ],
    }
}
```

- 新規追加後のRDF



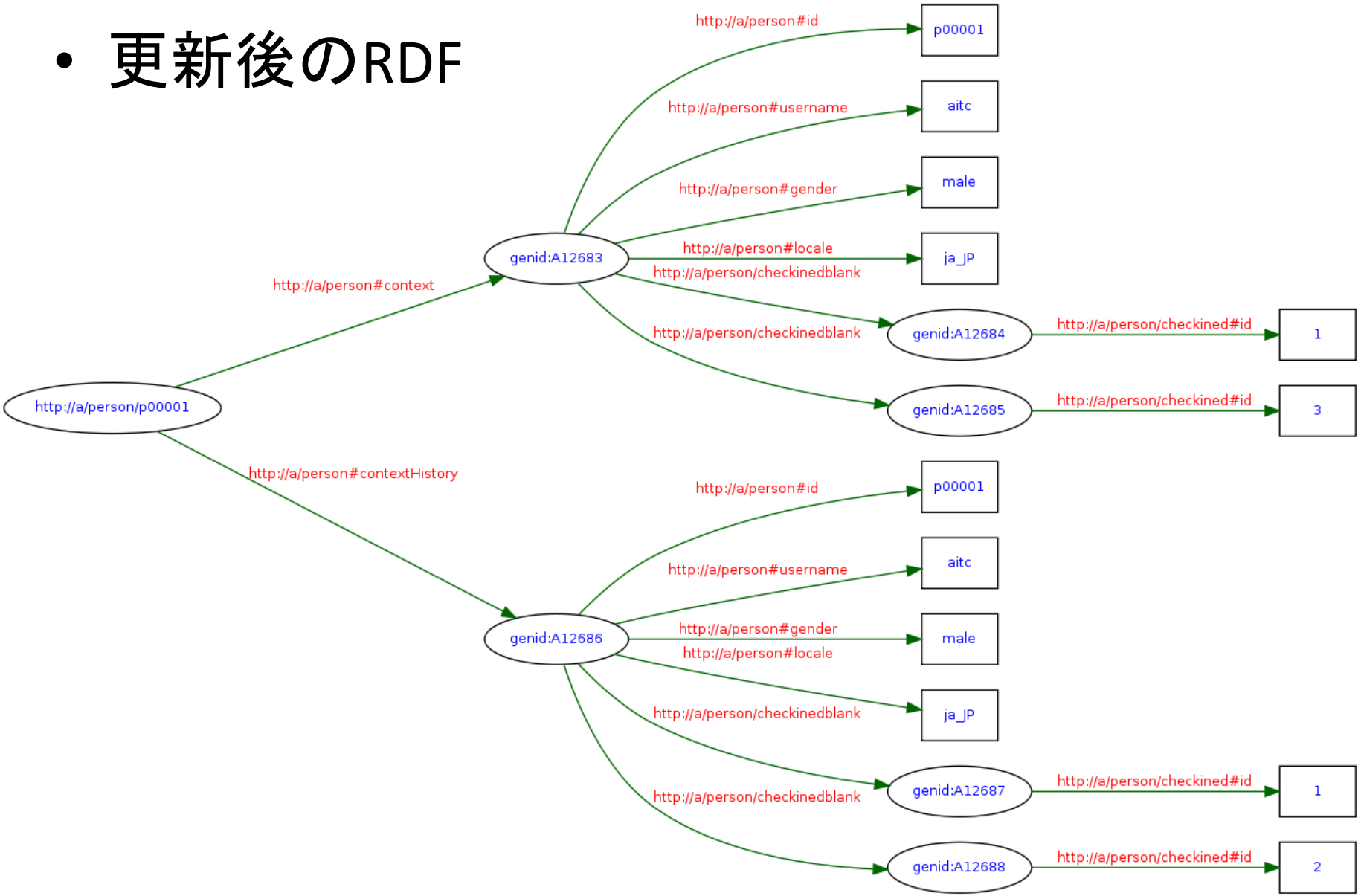
データを格納するAPI－３

- ユーザー情報は単なる更新ではない
  - 投稿した時点のユーザー状態を履歴として残す
    - contextをhistoryContextに書き換えるためのSPARQL

```
DELETE { <http://a/person/p00001> <http://a/person#context> ?o }
INSERT  { <http://a/person/p00001> <http://a/person#contextHistory> ?o }
WHERE  { <http://a/person/p00001> <http://a/person#context> ?o }
```

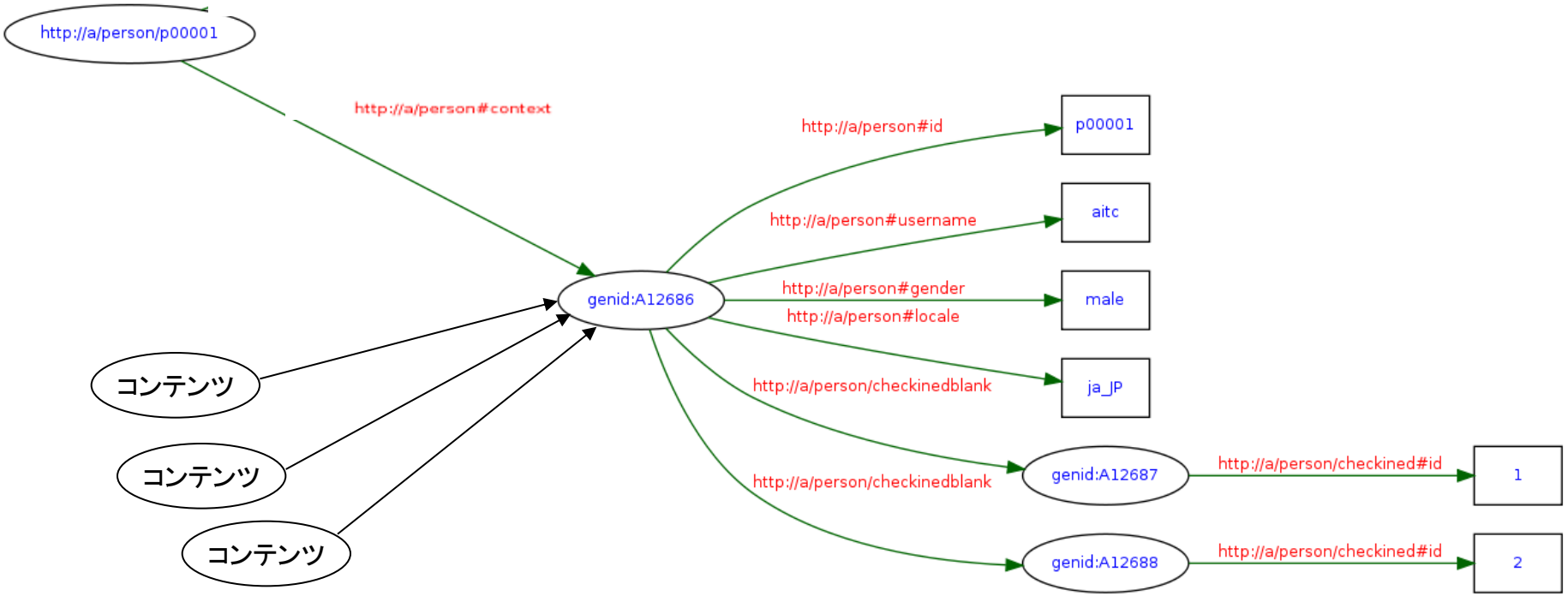
データを格納するAPI－４

- 更新後のRDF



データを格納するAPI－５

- コンテンツとの関連



データを取得するAPI－１

- 取得
  - GETするとJSONを返す
    - SPARQLでXMLを取得し、JSONに変換する
  - 取得時には、様々な付加情報を追加

データを取得するAPI－２

- 登録時

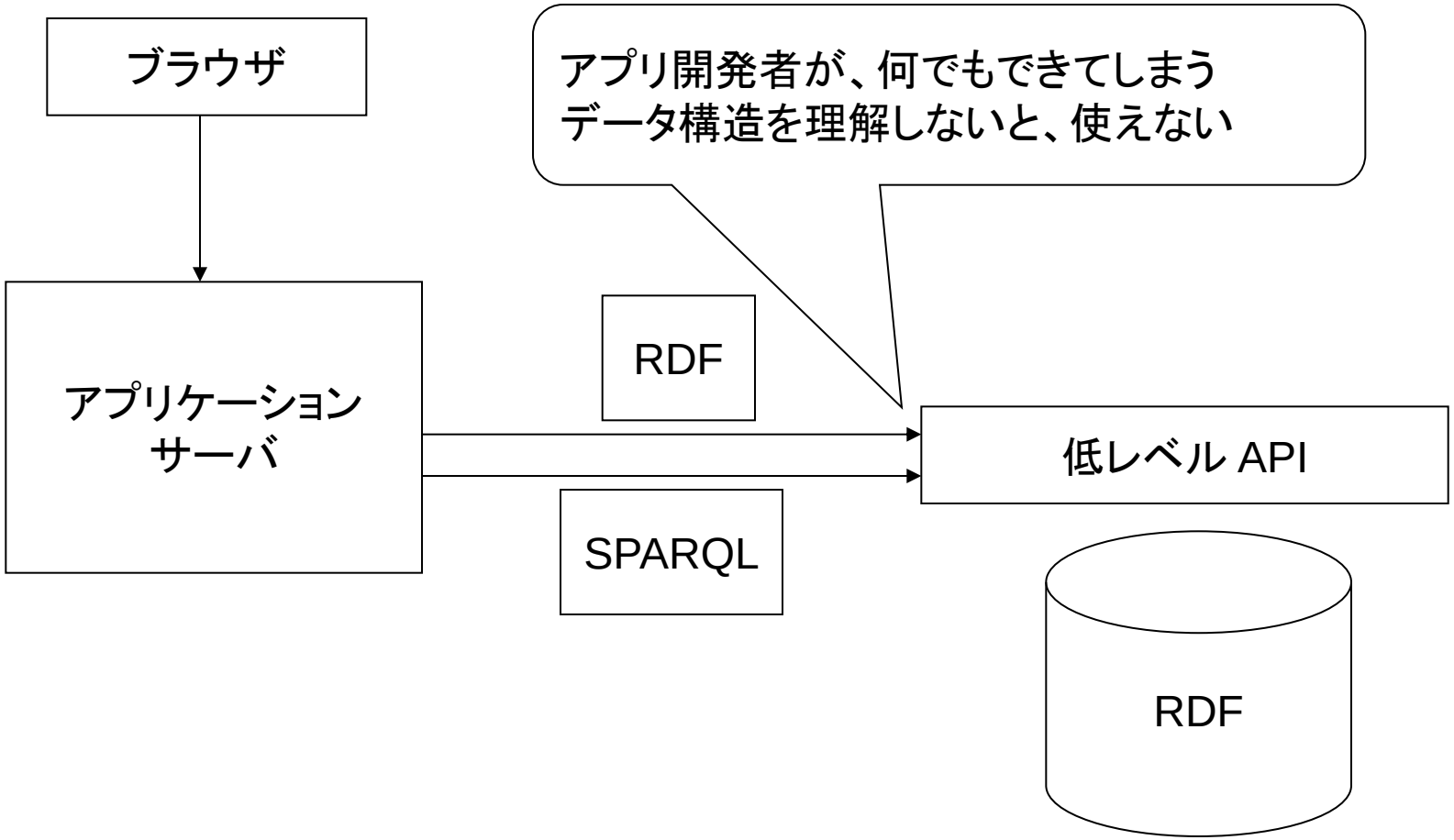
```
{
    "person": {
        "username": "aitc",  "gender": "male",  "locale": "ja_JP",
        "checkIned": [
            {"id": "1"}, {"id": "2"},
        ],
    }
}
```

- 取得時

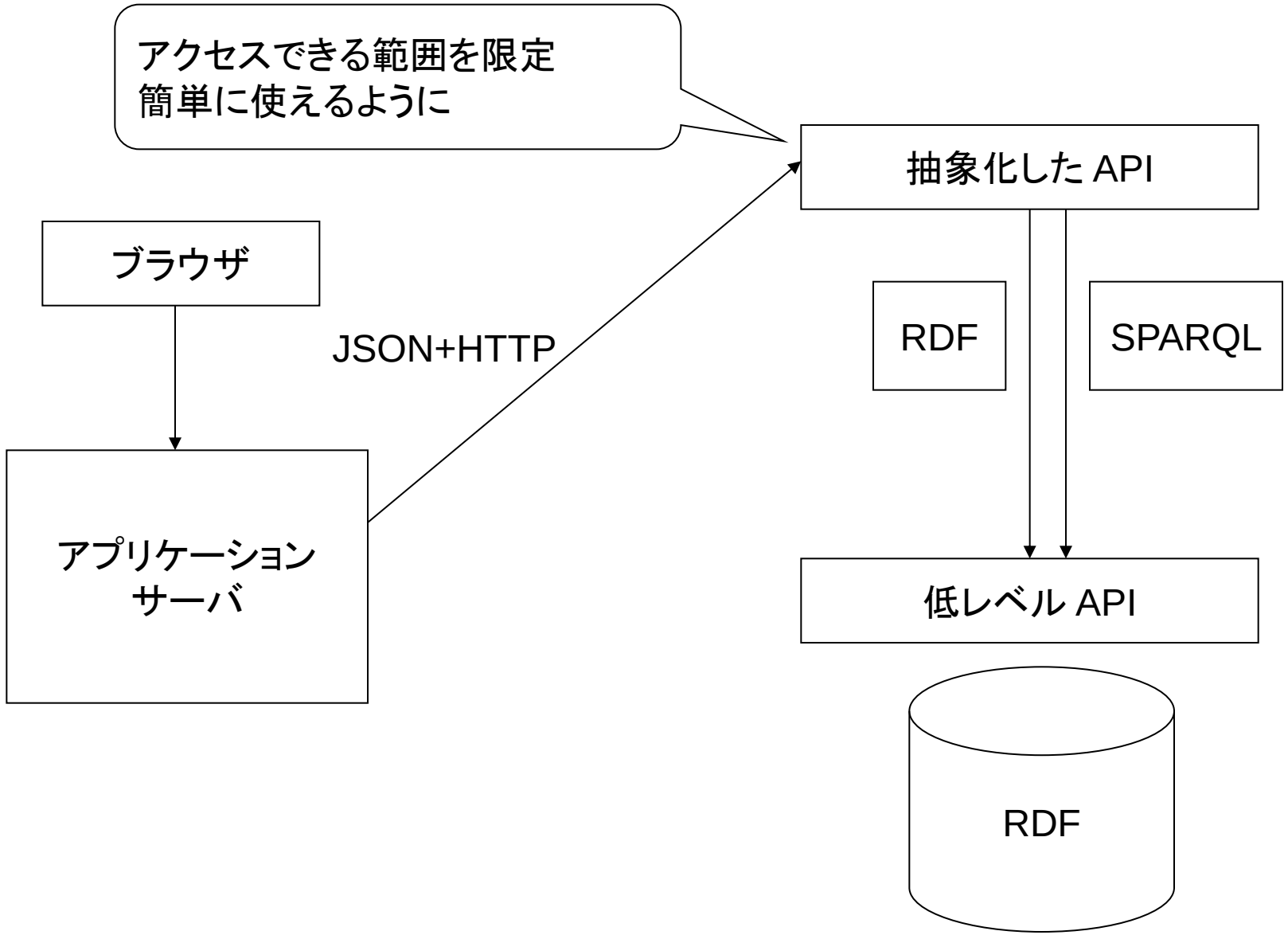
```
{
    "person": {
        "id": "p00001",
        "username": "aitc",  "gender": "male",  "locale": "ja_JP",
        "checkIned": [
                {"id": "1", "title": "3.11 首都圏。帰宅する方法は何？"},
                {"id": "2", "title": "3.11災害状況"},
        ],
    }
}
```

システム構成

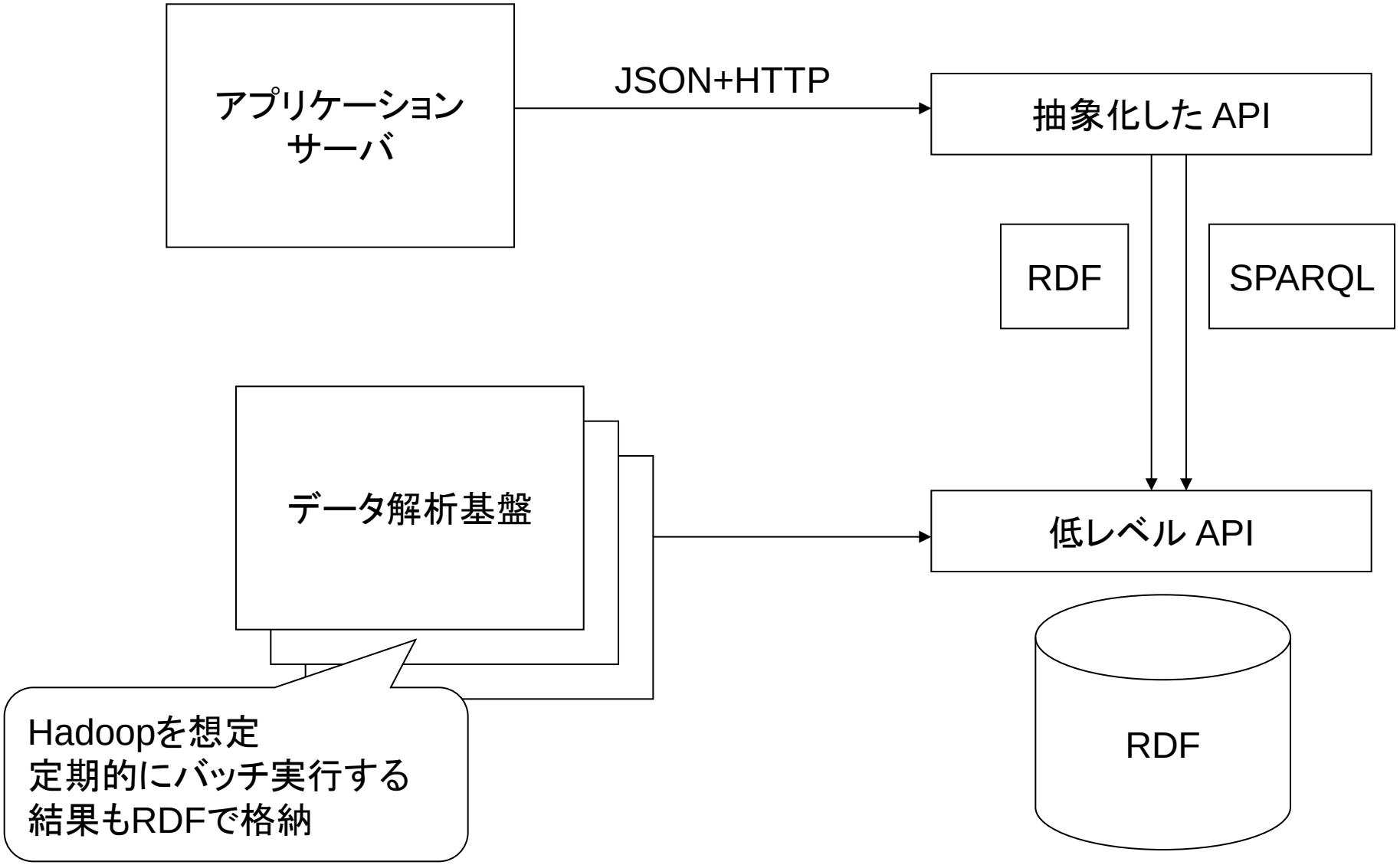
システム構成－初期



システム構成－現在開発中



システム構成－今後の予定



まだ開発中ですが、まとめ

- JSON⇔RDFは、ある程度パターン化できそう
  - O/Rマッピングよりは、かなり大変
- RDFで格納する事で、複雑な関連性を表現
  - 図化することで、理解も容易
- 集計が大変そうなので、クラウド技術が必須