第24回 Lucene/Solr勉強会
オンライン／初心者編２
Apache Lucene/Solr入門

8/17/2021 @kojisays

株式会社ロンウイット

はじめに（再掲）

- 約2年ぶりの開催。
    - 内容が高度化、準備が大変。間隔が空くとますますハードルが上がる。コロナ禍で勉強会もオンラインが一般的に。オンラインで短めのものを、これまでよりも頻度高めで。
- 今後数回に分けて初心者向けの内容を用意。
    - 関口から初心者の方々へ。今回のみならず毎年？
- その後は事例や新機能の発表など。
    - 勉強会＝双方向で。

ネタをお持ちの方等、アンケートでぜひお知らせください

前回受講アンケートより

- 入室音、定員、時間、質問方法等オンライン勉強会開催そのものについて。
- 内容詳細や資料の（事前）公開について。
- Apache Lucene と Solr の違いや製品そのものの話。
- tf-idfやBM25などのスコア、検索精度改善、形態素解析とN-gramの話。

前回の内容（復習）

- 情報検索（全文検索）とその実現方法
- 転置インデックス
- リレーショナルデータベースとの比較
    - スコア計算
- 日本語の単語分割
- 検索エンジンの構成要素

再掲（復習）

転置インデックスの検索例 1

検索質問

q=サザエ

転置インデックス

の	1, 2, 3
は	1, 2, 3
カツオ	1, 3
サザエ	1, 2
ワカメ	2, 3
姉	2
弟	1
妹	3

→ 文書1と2を返す

元の文書

1	カツオはサザエの弟
2	サザエはワカメの姉
3	ワカメはカツオの妹

← 元の文書が必要な場合は別途明示して
インデックス内の別の場所に保存する（後述）

再掲（復習）

日本語の単語分割方法

- 形態素解析
  - 辞書に基づいた単語分割。
    - 検索結果は検索誤りが低く、検索漏れが発生しがち。

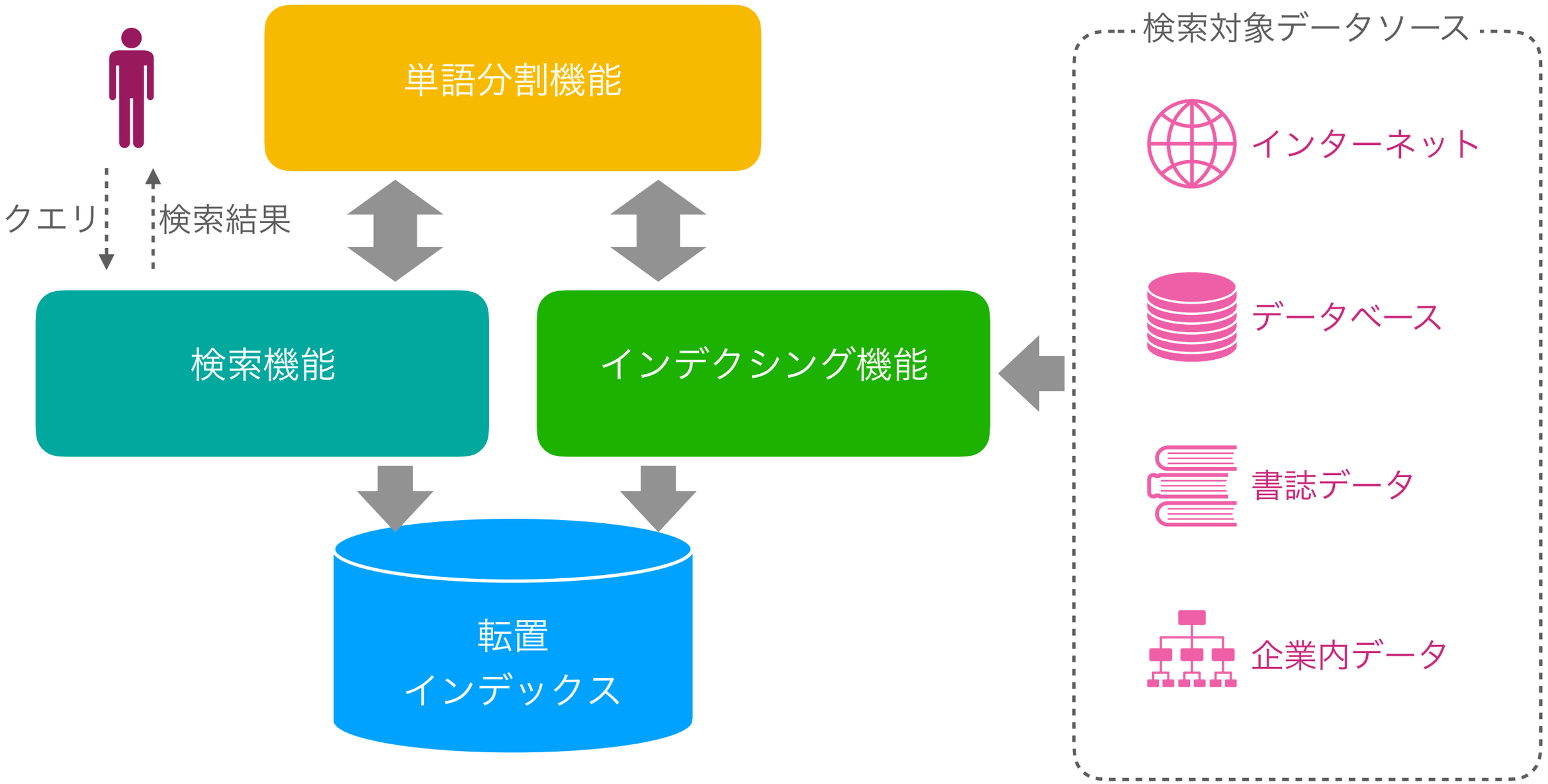
緊急事態宣言
↓
緊急 / 事態 / 宣言

- 文字N-gram
  - 機械的にN文字単位に分割。
    - 検索結果は検索漏れが低く、検索誤りが発生しがち。

緊急事態宣言　(N=2の場合)
↓
緊急 / 急事 / 事態 / 態宣 / 宣言

再掲（復習）

検索エンジンの構成要素



本日の内容

- Apache Lucene/Solr とは？

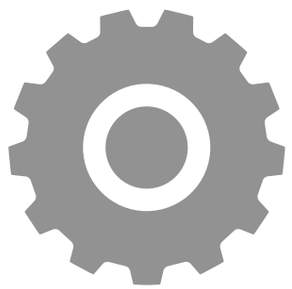
- 検索エンジンの各構成要素のLucene/Solrによる実現方法（対応するクラスやコンポーネント）

1. Apache Lucene ⇒ Lucene を使ったプログラム構築がいかに面倒くさいか。
   （ただ、一度でもプログラムを書いてみるのをお勧め）
2. Apache Solr ⇒ Solr がそれをいかに簡単に変えたか。

Apache Lucene/Solr

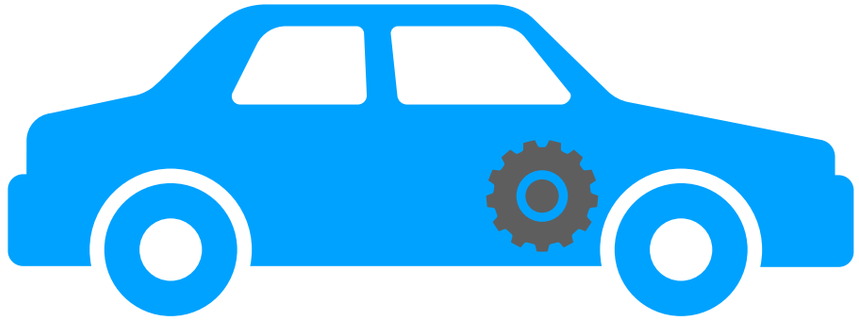
- Apache Lucene
  - Javaで書かれた全文検索ライブラリ

車のエンジン

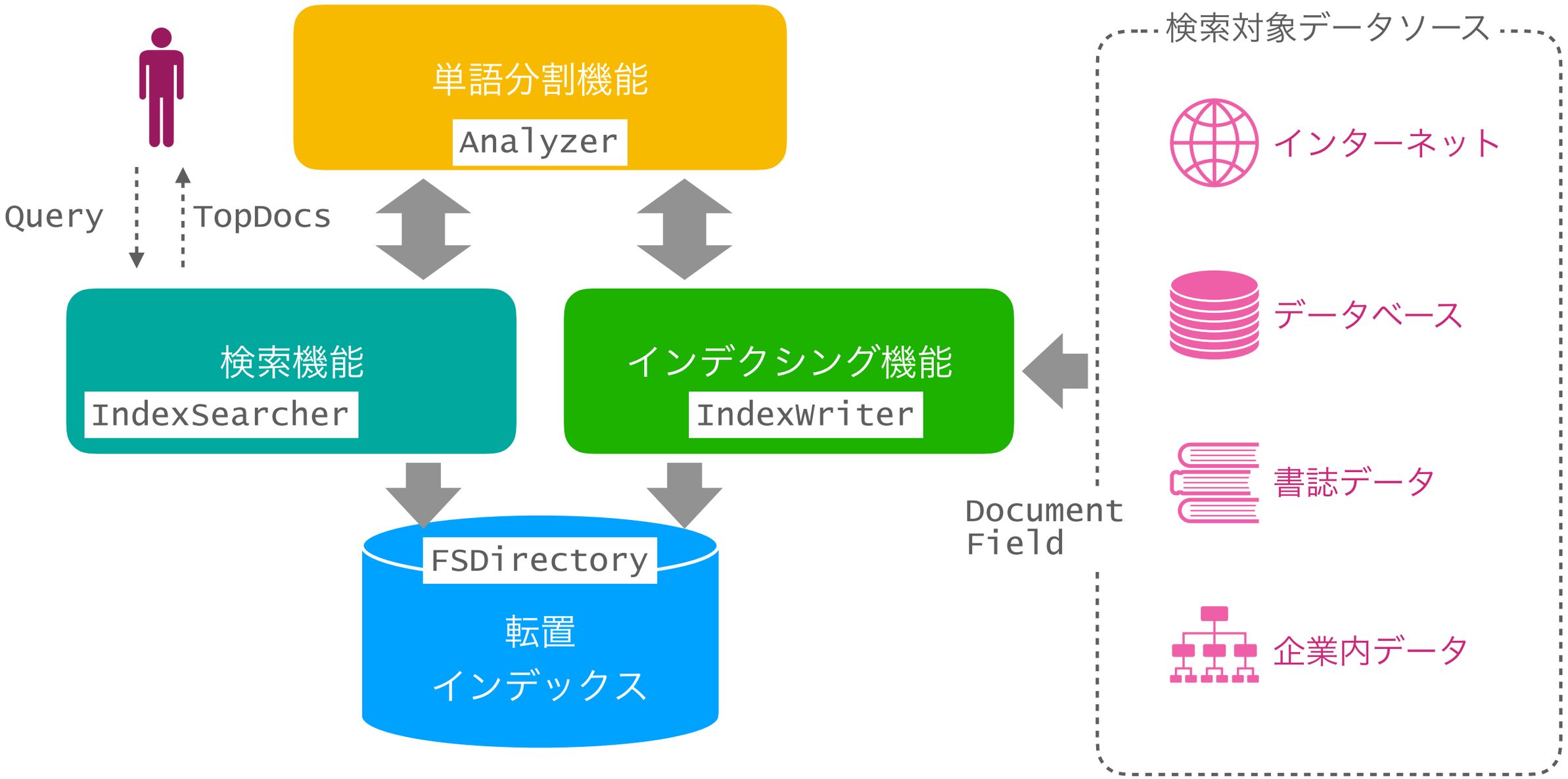


- Apache Solr
  - Javaで書かれた全文検索サーバー
  - Apache Luceneを利用

完成車（ナビ付き）



Luceneの主要なクラス



FSDirectory

```
Path index = FileSystems.getDefault().getPath("index");

// 転置インデックスの作成（なければ作成される）
Directory directory = FSDirectory.open(index);  // ①


:     // インデクシングや検索のコード（後述）でdirectoryを参照


directory.close(); // ②
```

Analyzer

```
// 日本語形態素解析器
Analyzer analyzer = new JapaneseAnalyzer();  // ①

:       // 以降のインデクシングや検索のコードで analyzer を利用

/*
 * その他の主な Analyzer
 *
 * - WhitespaceAnalyzer
 * - StandardAnalyzer
 * - PerFieldAnalyzerWrapper
 * - (NGramTokenizer)
 *
 */
```

IndexWriter, Document & Field

```
// インデクシング準備
IndexWriterConfig config = new IndexWriterConfig(analyzer);  // ①
IndexWriter writer = new IndexWriter(directory, config);

// 検索対象文書の作成
Document doc = new Document(); // ②
doc.add(new TextField("title", "入江聖奈（ボクシング）",
                      Field.Store.YES)); // ③
doc.add(new TextField("body", "東京五輪金メダリスト。カエル大好き。",
                      Field.Store.YES)); // ④
doc.add(new IntPoint("age", 21));  // storeされない。別途指定必要。 // ⑤
doc.add(new StringField("division", "feather")); // ⑥

// インデックスに登録
writer.addDocument(doc); // ⑦
writer.commit();
//writer.forceMerge();
writer.close();
```

IndexSearcher & Query

```
// IndexSearcher の準備
DirectoryReader reader = DirectoryReader.open(directory);
IndexSearcher searcher = new IndexSearcher(reader); // ①


// Query の作成（単純な単語検索の例）
Query query = new TermQuery(new Term("body", "カエル")); // ②

// 範囲検索の例
//Query query = IntPoint.newRangeQuery("age", 15, 25); // ③


// 検索の実行（次ページに続く）
TopDocs topDocs = searcher.search(query, 10); // ④
```

TopDocs

```
// ヒットしたトップの文書を取得（Field.Store.YESとしたFieldのみ可能）
Document d1 = searcher.doc(topDocs.scoreDocs()[0].doc); // ①
System.out.println(d1.get("title")); // ②
System.out.println(d1.get("body")); // ③


/*
 * その他、検索結果の取得いろいろ
 * - topDocs.totalHits();     // 総ヒット件数
 * - topdocs.scoreDocs()[0].doc;     // トップヒット文書番号
 * - topdocs.scoreDocs()[0].score;  // トップヒット文書スコア
 */

reader.close(); // ④
```

BooleanQuery

```
// ANDクエリを組み立てる（サザエ AND 弟）
BooleanQuery.Builder builder = new BooleanQuery.Builder(); // ①
builder.add(new TermQuery(new Term("f", "サザエ")),
    BooleanClause.Occur.MUST);  // ②
builder.add(new TermQuery(new Term("f", "弟")),
    BooleanClause.Occur.MUST);  // ③
Query query = builder.build();  // ④

/*
 * 2-gramフィールドの場合、"サザエ"の分割を考慮して
 * 以下のようにプログラムを書かなければならない。          ⑤
 *
 * PhraseQuery phraseQuery =
 *    new PhraseQuery(0, "f", "サザ", "ザエ");
 *
 */
```

QueryParser

```
// QueryParserを使ってもっと簡単にANDクエリを組み立てる
QueryParser parser = new QueryParser("f", analyzer);// ①
Query query = parser.parse("サザエ AND 弟"); // ②

// query.toString() を確認すると、
// BooleanQuery が組み立てられているのがわかる

/*
 * その他のQueryParserで有効なクエリ式例（N-gramも考慮不要）
 *
 * - body:(サザエ OR フネ) NOT title:タマ
 * - age:[15 TO 25]                  // 数値範囲検索
 * - body:tech*                      // techで始まる全ての単語
 * - body:"my friend"~1              // myとfriendの間に1単語許す
 *
 */
```

フィールドの重みを考慮

```
QueryParser parser = new QueryParser("f", analyzer);

Query query = parser.parse("title:サザエ^10 body:サザエ^2");
```

↓

通常、フィールドの重みはNormによって「いい感じ」に考慮される。（次ページ）

Normの効果

（問題）キーワード X を探しているとき、以下のどちらの文書を上位に表示して欲しいか？

	title	body
doc1	○ ○ ○ ○ ○ ○	○ ○ ○ ○ ○ ○ X ○ ○ ○ ○ X ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
doc2	○ ○ X ○ ○ ○	○ ○

↓

Normにより長いフィールド（単語数が多いフィールド）ほどスコアを低減させる。

$$\mathrm{score}(D,Q) = \sum_{i=1}^{n} \mathrm{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\mathrm{avgdl}}\right)}$$

ドキュメント長

https://ja.wikipedia.org/wiki/Okapi_BM25

IDFの効果

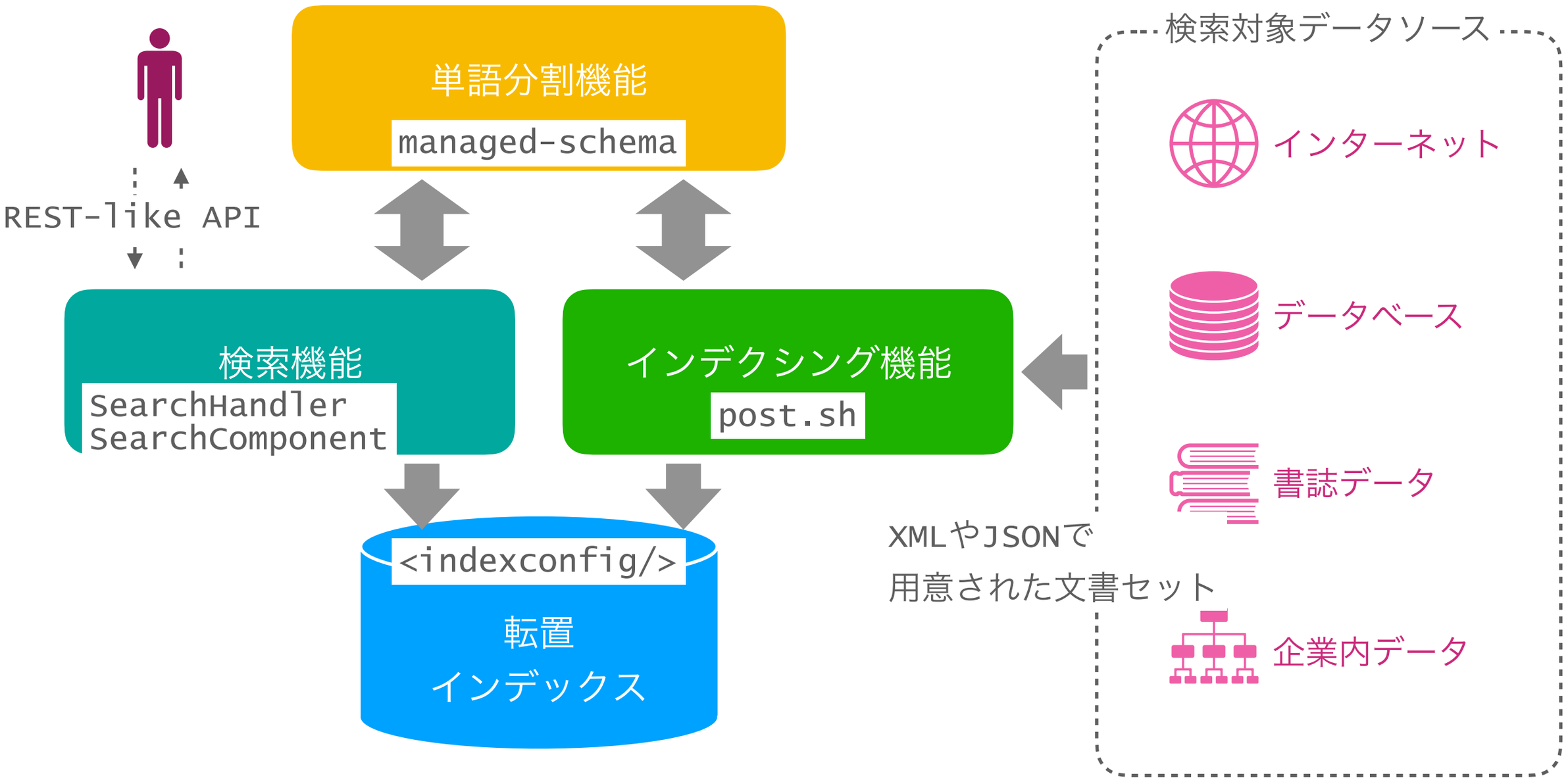
複数単語が現れる検索でIDF*が効いてくる。（例：「**サザエ** OR **弟**」。n=2, q_1=**サザエ**, q_2=**弟**）

*IDF: Inverse Document Frequency

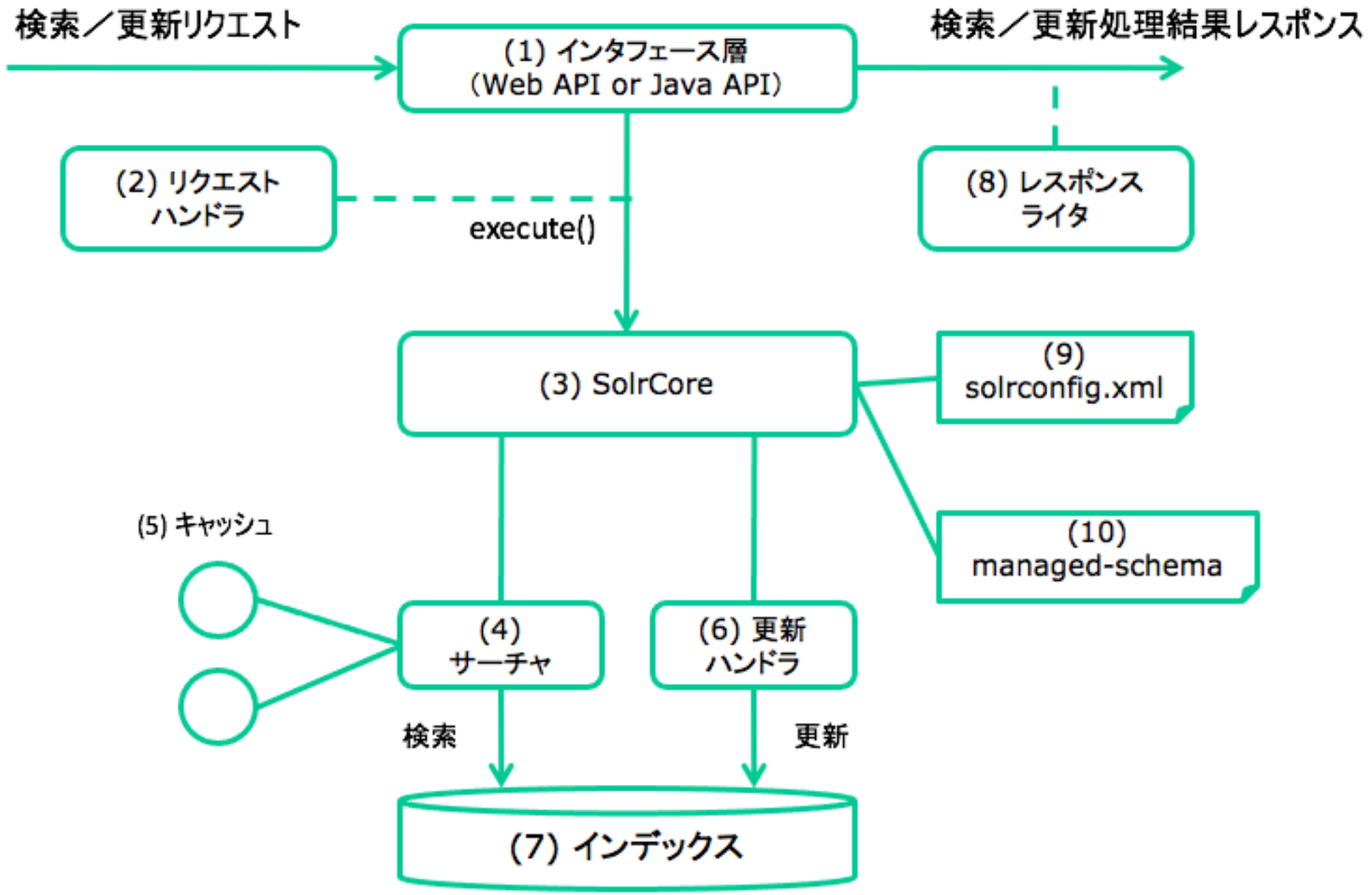
$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{\text{avgdl}}\right)}$$

	body
doc1	○ ○ ○ ○ ○ ○ **弟** ○
doc2	○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ **弟** ○ ○ ○ ○ ○ ○ ○ ○ ○
doc3	○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ **弟** ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
doc4	○ ○ ○ ○ ○ **サザエ** ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ **弟** ○ ○ ○ ○ ○
	○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ **弟** ○ ○ ○ ○ ○ ○ ○ ○ ○
:	
doc99	○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ **サザエ** ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○
doc100	○ ○ ○ ○ **弟** ○

Solrの主要なコンポーネント



Solrの構成要素



ロンウイット社「Solr 基礎」テキストより引用。

Solrを使い始める準備

1.インストール

A. Solrダウンロードサイトから ZIPをダウンロードして展開

B. DockerでSolrコンテナを実行

2.起動（デフォルトポート番号8983）

```
solr start
```

3.コレクション（コア）の作成

```
solr create_core -c collection1 \
     -d sample_techproducts_configs
```

Solrでのスキーマ設定

```
<field name="age" type="pint"
       indexed="true" stored="true"/> ①

<field name="division" type="string"
       indexed="true" stored="true"/> ②

<field name="title" type="text_ja" ③
       indexed="true" stored="true" multiValued="false"/>

<field name="body" type="text_ja" ④
       indexed="true" stored="true" multiValued="true"/>

<field name="id" type="string"
       indexed="true" stored="true"/> ⑤

<fieldType name="pint" class="solr.IntPointField" ⑥
           docValues="true"/>

<fieldType name="string" class="solr.StrField"/> ⑦
```

Solrでのスキーマ設定（つづき）

```
<fieldType name="text_ja" class="solr.TextField"
           positionIncrementGap="100" omitNorms="false"
           autoGeneratePhraseQueries="true"> ①

  <analyzer type="index"> ②
    <tokenizer class="solr.JapaneseTokenizerFactory"/> ③
    <filter class="solr.JapaneseBaseFormFilterFactory"/> ④
    <filter class="solr.JapaneseKatakanaStemFilterFactory"
            minimumLength="4"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>

  <analyzer type="query"> ⑤
    <tokenizer class="solr.JapaneseTokenizerFactory"
                mode="search"/>
    <filter class="solr.JapaneseBaseFormFilterFactory"/>
    <filter class="solr.JapaneseKatakanaStemFilterFactory"
            minimumLength="4"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>

</fieldType>
```

Solrでの文書登録

```
post -c collection1 -commit yes mydocs.json
```

```
[
  {
    "id": "tokyo2020-boxing-wm-feather-001",
    "title": "入江聖奈（ボクシング）",
    "body": [ "東京五輪金メダリスト。カエル大好き。" ],
    "age": 21,
    "division": "feather"
  },
  :
]
```

Solrでの検索例

1.全件検索

```
http://localhost:8983/solr/collection1/select?q=*:*
```

2.キーワード検索、範囲検索

```
http://localhost:8983/solr/collection1/select?q=body:カエル
http://localhost:8983/solr/collection1/select?q=age:[15 TO 25]
```

3.絞り込み検索

```
http://localhost:8983/solr/collection1/select?q=*:*
    &fq=age:[15 TO 25]&fq=division:feather
```

4.レスポンスライターを指定

```
http://localhost:8983/solr/collection1/select?q=*:*&wt=php
```

5.ハイライト

```
http://localhost:8983/solr/collection1/select?q=body:カエル
    &hl=on&hl.fl=body&hl.snippets=3&hl.fragsize=150
```

curl コマンドで実行するときはURLエンコードを行う（例："*:*" ⇒ "*%3A*"）

Solrでの検索結果


```
{
  "responseHeader":{ // ①
    "status":0,
    "QTime":5,
    "params":{
      "q":"body:カエル",
      "hl":"on",
      "hl.fl":"body",
      "rows":"1"}
  },
  "response":{"numFound":1,"start":0,"numFoundExact":true,  // ②
    "docs":[
      {
        "id":"tokyo2020-boxing-wm-feather-001",
        "title":"入江聖奈（ボクシング）",
        "body":["東京五輪金メダリスト。カエル大好き。"],
        "_version_":1691179867063713792
      }
    ]
  },
  "highlighting":{  // ③
    "tokyo2020-boxing-wm-feather-001":{
      "body":["東京五輪金メダリスト。<em>カエル</em>大好き。"]
    }
  }
}
```

まとめ

- Apache Lucene はOSSの全文検索ライブラリ。車にたとえればエンジン。エンジンがただで配られていても、車を作ろうという人は少数で、なかなか普及しなかった。
    - 短くていいのでLuceneを使ってプログラムを書こう！（お勧め）
- Apache Solr はLuceneを使って書かれたOSSの全文検索サーバー。車にたとえれば完成車。ただエンジンを使っただけではなく、ナビがついているなど完成度が高く、ユーザーが全世界に一気に広まった。
    - ユーザー数の増加やSolrでの発明がLuceneに逆輸入されるなど、Luceneの価値向上にも貢献。

次回（予定）

- Apache Solr のスキーマ設定
  - 基本テクニック、日本語特有の注意事項、みんな大好きシノニム、・・・
- 日時未定

受講アンケート

- 次回以降の勉強会の参考とするため、ぜひ受講アンケートにご協力お願いします！

- 勉強会で発表できるネタをお持ちの方、本日の内容で質問のある方、次回以降で取り上げて欲しい内容のリクエストなどあれば、アンケートの自由記入欄にお書きください。

https://docs.google.com/forms/d/e/1FAIpQLSceP77mzLRgcm0Oluy4hrwEVEfdJFLf8hl3dwlaHLwv1R4UvQ/viewform?usp=sf_link

開始時間（11:00）まで
ビデオと音声をオフ（ミュート）にして
そのままお待ちください。

第24回 Lucene/Solr勉強会
オンライン／初心者編２