記事
Mihoko Iijima · 2021年5月7日 10m read

GPS（GPX）データを InterSystems IRIS に高速に取り込む方法を体験できる実行環境テンプレート

開発者の皆さん、こんにちは！

この記事では、Java から IRIS へ接続する方法の中から **XEP**（ゼップ）を利用して、GPS（GPX）データを高速に取り込むサンプルをご紹介します。

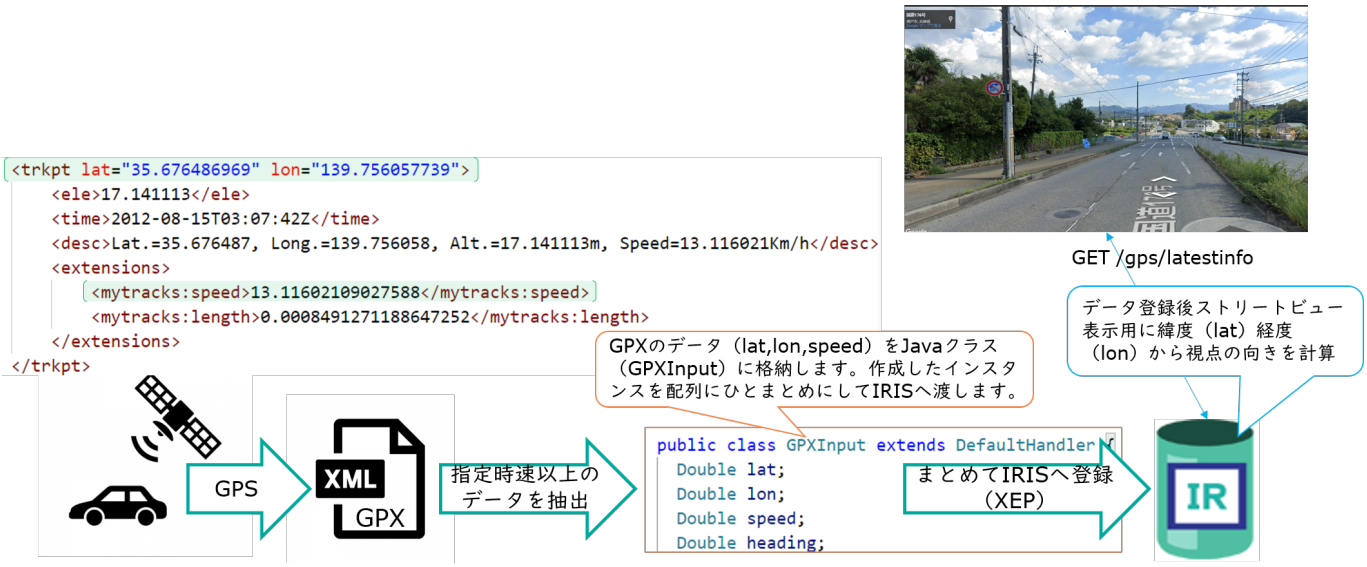
Java の実行環境や IRIS がお手元になくても大丈夫です！コンテナのビルド＋開始で体験できる「実行環境テンプレート」をご用意しました。

ソースコードは コミュニティの Git で公開 していますので、docker、docker-compose 、git がインストールされた環境であれば、すぐにお試しいただけます（Javaの実行環境はコンテナでご提供するので準備不要です）。

操作方法は、Gitの README に記載しています。ぜひお試しください。

この記事の中では、コード解説を追加しています。ぜひ、最後までお付き合いください！

1)処理概要



GPX データを IRIS へ渡す迄の流れにリアクティブプログラミングが行える RxJava2 のライブラリを使用しています。

> メモ：サンプルコードの中では、GPS データを直接受信するのではなく、Google マイマップやサンプル GPS データから GPX ファイルに変換したデータを入力に使用しています。

実行時、複数（数十～千）の GPS データが含まれる GPX ファイルを引数に指定し情報を入力します。

XMLからリストを作成し、作成したリストを RxJava2 の Flowable に渡し、データのフィルタリングを行い、IRIS へデータを渡しています。

> サンプルの GPX ファイルは、Google マイマップから作成しています。マイマップから作成した GPS には、残念ながら速度が含まれません。
>
> 弊社パートナーのビズベース様より、速度が含まれる GPS データをご提供いただきました（ファイル：Test-DriveData13.gpx.xml ）。ビズベース様提供データ Test-DriveData13.gpx.xml を入力した場合は、0km/h より速いデータだけを処理するように記載しています。

1-1) Java から IRIS へ接続する方法

4 手法あります（サンプルでは、この中の 1 つを利用しています）。

1. SQLでアクセスする場合に便利な **JDBC** の利用
2. 大量データを高速に登録したい場合に最適な **XEP**
3. キーバリュー形式でデータを設定／取得したい場合の **Native API**
4. **Hibernate** を利用する方法

サンプルは、**2) 大量データを高速に登録したい場合に最適な XEP（ゼップ）** を利用しています。

1-2) XEP について

XEP は、Javaで作成したオブジェクトを IRIS に永続化する際に使用する接続方法です（事前に IRIS 側でクラス／テーブルの作成は不要）。

データ登録までの流れは以下の通りです。

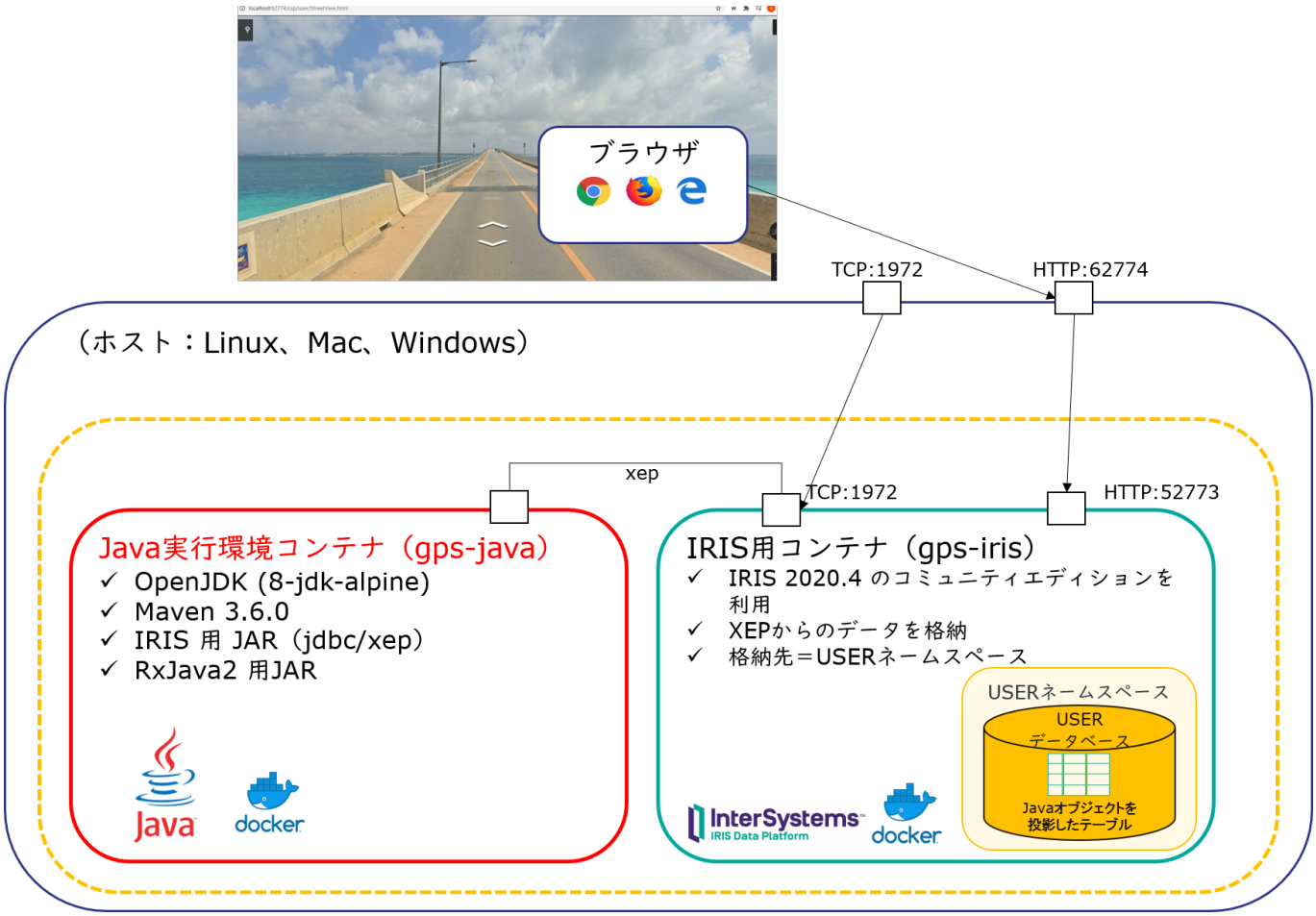
1. IRIS に永続化したいオブジェクトを Java クラスで用意
2. XEP を利用して IRIS に接続し、Java クラスのスキーマ情報を IRIS へ投影（Java クラスを分析し、スキーマをインポート）
3. 永続化したい Java オブジェクを 1 つ以上作成し、Java の配列にセット
4. IRIS へ渡して永続化（store() メソッドを利用して登録）

XEPについて詳しくは、ドキュメントをご参照ください。

2) 実行環境テンプレートの使用方法

テンプレートはコンテナを利用しています。 Docker、docker-compose、git が利用できる環境でお試しください。

使用するコンテナのイメージ



データ登録後、ストリートビューで表示を確認できます。

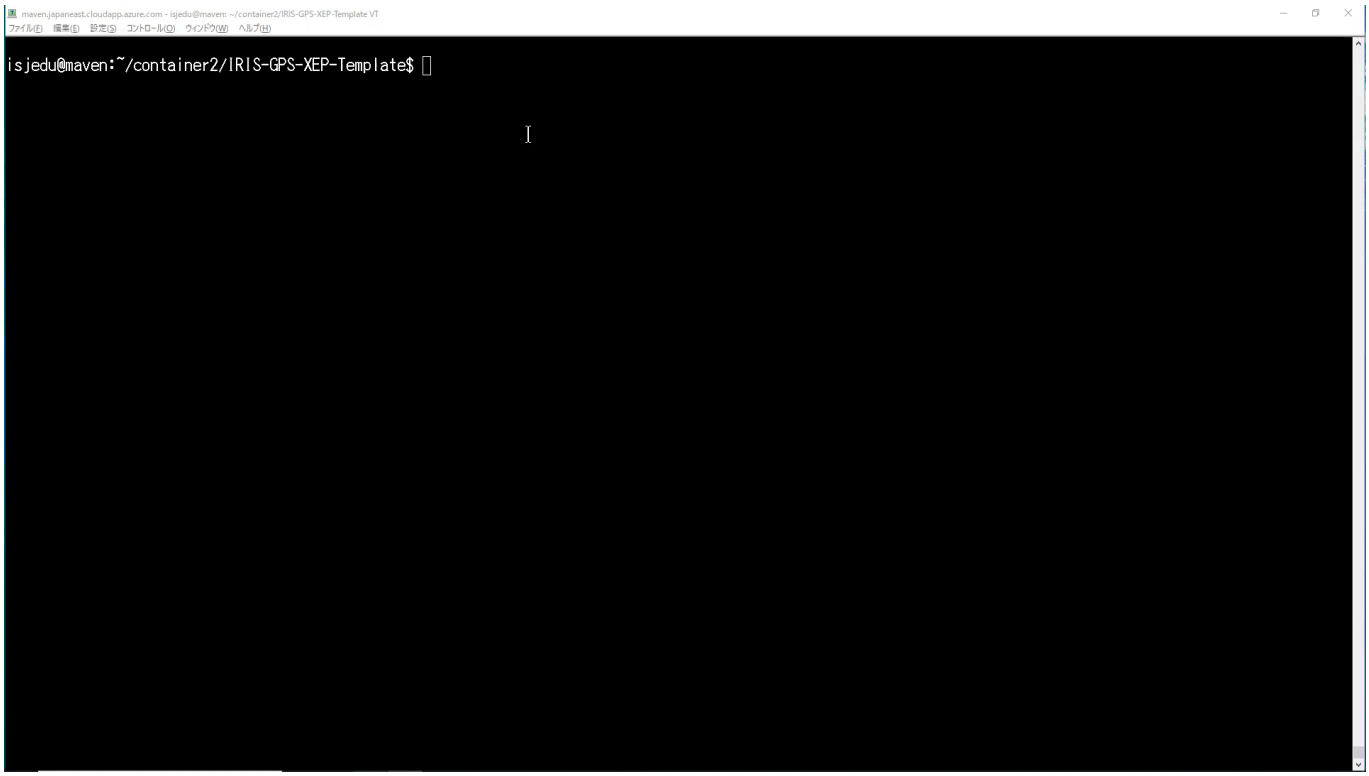
URL: http://お使いのホスト:62774/csp/user/StreetView.html

> ストリートビューの正確な表示には、GoogleのAPIキーが必要となります。
>
> APIキーを入手されたら、Git clone 後、./IRIS/web 以下にある apikey.txt に API キーを保存してから IRIS 用コンテナのビルドを行ってください（ビルド時に
> 、apikey.txt
> の中身を反映しながらコンテナを作成しています。記載がない場合はキー無しの状態でお試しいただけます）。

コンテナ開始までの手順詳細については、以下リンク先をご参照ください。

- 2-1) ダウンロード (git clone)
- 2-2) ストリートビューの表示をお試しいただくための準備
- 2-3) Java実行環境用コンテナを使う場合
- 3) Java の実行をホストで行う場合
  - 3-1) Linuxの場合
  - 3-2) Windows の場合

2-3) の手順例



サンプルコードについて

実行に使用している Start.java では、引数に渡される GPX ファイル
から、リストを作成しています（リストの作成には GPXInput.java を使用）。

リストには、GPXInput クラスのインスタンスを登録しています。

GPXInput クラスのインスタンスには、GPX
ファイルから取得した、緯度（lat）、経度（lon）、速度（speed）を設定しています。

> Google マイマップで線を引いた GPS データは、速度が入ってこないため、速度データがある場合のみ speed にデータを設定しています。

```
<trkpt lat="35.676486969" lon="139.756057739">
    <ele>17.141113</ele>
    <time>2012-08-15T03:07:42Z</time>
    <desc>Lat.=35.676487, Long.=139.756058, Alt.=17.141113m, Speed=13.116021Km/h</desc>
    <extensions>
        <mytracks:speed>13.11602109027588</mytracks:speed>
         <mytracks:length>0.0008491271188647252</mytracks:length>
    </extensions>
</trkpt>
```

（以下、GPXInput.java から一部抜粋）

```
public class GPXInput extends DefaultHandler {
  Double lat;
  Double lon;
  Double speed;
  Double heading;
  //XML??????????List???
  public static List<GPXInput> datalist = new ArrayList<GPXInput>();
```

GPXInput クラスのインスタンスが入ったリスト作成後、Start.java の中で、IRISに接続するために必要なインスタンスを作成しています（以下、抜粋して記載しています。IRISへの接続情報は、 Start.java の 12～16 行目をご参照ください）。

```
        // XML??GPXInput????????????????
        List<GPXInput> datalist=GPXInput.GPXToList(Paths.get(args[0]).toFile().toStri
ng());
        System.out.println("???????" + datalist.size());

        // EventPersister?????IRIS??????????
        EventPersister xepPersister = PersisterFactory.createPersister();
        xepPersister.connect(host,port,namespace,username,password);

        // ?????????????????????????
        String classname="JavaXEPSample.GPXInput";
        xepPersister.deleteExtent(classname);

        // ????????????????
        xepPersister.importSchema(classname);
```

最後の行の ***xepPersister.importSchema(classname);*** では、Java クラスのインスタンスを永続化するため、Java クラス名（ここでは、JavaXEPSample.GPXInput）を引数に指定し、スキーマをインポートしています。

残りの処理は、作成したリストを IRIS へ渡し、永続化するだけです。

今回のサンプルでは、本物の GPS データが流れてくるわけではないのですが、そうなっても対応できるようにリアクティブプログラミングが行える RxJava2 のライブラリを使用しています。

サンプルでは、EmitData.java の中で記述しています。

EmitData.java では、最初に Java クラスのインスタンス永続化などに使用する「イベント」を作成しています。

イベントの作成は、 EventPersister から作成します（例文の xepPersister は、EventPersister のインスタンスで、EmitData.java の Start() 関数の第1引数で受け取っています）。

```
Event xepEvent = xepPersister.getEvent(Java????);  //??????????????????????
```

この後の永続化の処理ですが、GPS
データ受信の度に1件ずつ永続化するか、全件まとめて永続化するか、タイミングを選択できます。

本物の GPS
データを受信している場合は、通知（subscripbe()の第1引数）のタイミングが良いと思いますが、サンプルでは GPX ファイルからデータを一括で取得しているのもあるので、データ送付の完了通知（subscripbe()の第3引数）のタイミングで一括登録しています（XEP は 1件だけ／一括 のどちらでも登録できます）。

インスタンスの永続化を命令している行は、以下の1行です。

```
xepEvent.store(storedata);
```

ということで、処理全体は以下の通りです。

```
public class EmitData {
    public static void Start(EventPersister xepPersister,List<GPXInput> datalist,Stri
ng classname) throws InterruptedException{
        // EventPersister??Event???
        Event xepEvent = xepPersister.getEvent(classname);

        //??????????????List
        List<GPXInput> storelist = new ArrayList<GPXInput>();

        Flowable.fromIterable(datalist)
        .subscribeOn(Schedulers.io())
        .filter(obj->(obj.speed == null) || ((obj.speed !=null)&&(obj.speed>0)))
        .subscribe(
            // ??????
            // ** 1???IRIS??????????????????????????????????????????? **
            obj -> {
                storelist.add(obj);
                System.out.println("onNext?" + obj.lat + "-" + obj.lon + "-" + obj.sp
eed);
        },

        // ?2?????????
        error -> System.out.println("???=" + error),

        //?3???????
        () -> {
            System.out.println("????????? " + storelist.size());

            GPXInput[] storedata = storelist.toArray(new GPXInput[storelist.size()]);
            xepEvent.store(storedata);
            System.out.println("???????!");
        });
```

```
        Thread.sleep(2000);
    }
}
```

ここまでが、Java で行っている流れです。

サンプルでは、この他に IRIS に登録した GPS （緯度と経度）を REST で取得し、ストリートビューで確認できるようにもしています（IRIS 内に REST ディスパッチクラス を用意しています）。

IRIS で作成する REST サーバについてご興味ある方は、ぜひこちらの記事もご参照ください。

GPX に含まれる経度と緯度の情報だけでも、ストリートビューは表示できますが、視点の向きを安定させるため Java 実行後に IRIS に用意したルーチン（enrich.mac）を呼び出し、情報を追加しています。

（以下、一部抜粋）

```
calcdegree(x1,y1,x2,y2)
    set rd=($ZPI/180)
    set temp1=$zsin((x2-x1)*rd) if temp1=0 {return ""}
    set temp2=$zcos(y1*rd) * $ztan(y2*rd)- $zsin(y1*rd)*$zcos((x2-x1)*rd)
    set degree=90-($zarctan(temp2/temp1)*180/$ZPI)
    if (x2-x1)>=0 {
        set degree=90-($zarctan(temp2/temp1)*180/$ZPI)
    }
    else {
        set degree=270-($zarctan(temp2/temp1)*180/$ZPI)
    }
    return degree
```

サンプルの GPX ファイル は、桜島、江の島、戸隠へ向かう道、有馬温泉へ向かう道、哲学の道、サグラダファミリア付近、ピラミッド付近、など用意しています。こんな時期ですので、ちょっとした　　お散歩感覚 でストリートビューの表示をお楽しみいただけたらと思います。

最後までお付き合いいただきありがとうございました！

＜参考にさせていただいたページ＞

ストリートビューの表示：https://www.asobou.co.jp/blog/web/streetview

XMLの読み込み：https://engineer-club.jp/java-xml-read

#Java #オブジェクトデータモデル #コンテナ化 #InterSystems IRIS #InterSystems IRIS for Health

ソースURL:
https://jp.community.intersystems.com/post/gps%EF%BC%88gpx%EF%BC%89%E3%83%87%E3%83%BC%E3%82%BF%E3%82%92-intersystems-iris-%E3%81%AB%E9%AB%98%E9%80%9F%E3%81%AB%E5%8F%96%E3%82%8A%E8%BE%BC%E3%82%80%E6%96%B9%E6%B3%95%E3%82%92%E4%BD%93%E9%A8%93%E3%81%A7%E3%81%8D%E3%82%8B%E5%AE%9F%E8%A1%8C%E7%92%B0%E5%A2%83%E3%83%86%E3%83%B3%E3%83%97%E3%83%AC%E3%83%BC%E3%83%88