組み込み型 DB とキーバリュー型 DB 間の画像及び位置情報の連携に関する初歩的研究

島根大学　総合理工学部　数理・情報システム学科

計算機科学講座　田中研究室

s073009　射場　解

目次

第１章　序論 4
第２章　組み込み型データベースとキーバリュー型データベース 5
2.1　組み込み型データベース 5
2.2　キーバリュー型データベース 5
2.3　開発環境 6
第３章　システムの設計 7
3.1　システムの概要 7
3.2　SQLite に画像を保存 8
3.2.1　画像及びコメント入力画面に遷移 8
3.2.2　保存する画像を一覧から選択する 10
3.2.3　画像を SQLite に保存 11
3.3　SQLite から Datastore へ画像と住所を保存 11
3.3.1　SQLite から画像を送信 11
3.3.2　SQLite から住所の文字列を送信 12
3.3.3　サーブレット側で画像を受信し Datastore に保存 13
3.3.4　サーブレット側で住所の文字列を受信し Datastore に保存 15
3.4　Datastore から住所を取得 16
3.4.1　店舗詳細の住所をサーブレットに送信 16
3.4.2　Datastore に同じ住所が保存されているか検索 17
3.5　地図を表示 18
第４章　実行結果 19
4.1　SQLite に画像を保存 19
4.2　画像及び住所を SQLite から Datastore に保存 19
4.3　Datastore から住所を検索　し地図をサーバ上に表示 20
第５章　終論 21
謝辞 22

参考文献 23
付録.. 24

第１章　序論

Androidは、2007年に登場し現在ではスマートフォン用OSとして最も普及しており、日本、アメリカなどでトップシェアとなっている[1]。組み込み型データベースとは、スマートフォンでは Android をはじめとした、組み込み機器に用いられているデータベース技術である。一方、最近はアプリケーションやデータなどをネットワーク上に存在させ、必要な時にクライアントにダウンロードして利用するクラウドコンピューティングが普及している。キーバリュー型データベースとは、クラウドコンピューティングを支える重要なデータベース技術である。そのため、組み込み型データベース及びキーバリュー型データベースを研究することは有意義であると考えた。

本研究では代表的なメディアである画像及び住所などの文字列を組み込み型データベースであるSQLiteとキーバリュー型データベースであるDatastore間で連携させるシステムの開発を行う。 具体的にはまずシステムを起動してAndroid端末を操作し、画像をSQLiteに保存する。そして、保存した画像とあらかじめSQLiteに保存しておいた住所を合わせてSQLiteからDatastoreに保存する。 そして、Datastoreに保存された住所を検索して取得する。最後に、地図の画像をサーバ上に描画し Android 端末から表示できるシステムの開発を行う

第２章　組み込み型データベースとキーバリュー型データベース

2.1　組み込み型データベース

組み込み型データベースとは、組み込みシステムで利用されているデータベースである。今回の研究では、組み込み型データベースとして Android をはじめとしたスマートフォンで用いられているデータベースである SQLite を使用する。
SQLite の特徴としては以下のようなものが挙げられる。[2]

- トランザクション処理が可能
  複数の処理を一つにまとめて処理することで、処理前後のデータの整合性を保証する。
- データ型に対する扱いが柔軟
  データベースに誤った型を挿入しようとした時に適切な型に解釈する。
- SQLite のデータはひとつのファイルにまとめられている
  バックアップが容易になり、ファイルをコピーするだけでデータベースを複製することが可能。

2.2　キーバリュー型データベース

キーバリュー型データベースとは、データを Key(キー)と Value(値)の組み合わせで保存するデータベースである。今回の研究では、GoogleAppEngine[3]で提供されている Datastore を使用する。Datastore の特徴としては以下のようなものが挙げられる。[4]

- ほぼ無制限といえるスケーラビリティ
  膨大な量のデータに対して、同時に膨大なアクセスがあった場合でも、読み書きにかかる時間はわずか数十ミリ秒に抑えることができる。
- 可用性の高さ
  読み書きされたデータはコピーされ、複数のデータセンターに保存される。そのため、データセンター規模で障害が発生しても、すばやく復旧することが可能になる。

2.3　開発環境

まず、システムの構築は JAVA 言語で行うため、Java Development Kit (JDK) 6.0[5]のダウンロード及びインストールを行った。次に、JAVA 言語に対応した統合開発環境である、eclipse3.7(Indigo)[6]のダウンロード及びインストールを行った。次に、eclipse で Android の開発を行うためのツールである Android Development Tools (ADT)[7]プラグインのインストールを eclipse のメニューから行った。最後に、Android アプリケーションの開発キットである Android SDK[8]をダウンロードし、インストールを行った。

また、システムの動作を確認するため、Android 端末のエミュレータを eclipse から作成した。なお本研究では、エミュレータのバージョンとして Android4.0.3 を使用した。

第３章　システムの設計

3.1　システムの概要

本研究で作成したシステムの概要は図 3.1 のようになる。図 3.1 はシステムを構成する画面のフローチャートである。ここで①から⑤までの矢印は、画面間の行き来を表している。

まず、システムを実行すると図の左上の店舗一覧画面が表示される。店舗一覧から、店舗名を選択すると、矢印①を辿って、店舗詳細画面が表示される。

次に、店舗詳細画面にて「コメントと画像を追加」を選択すると、画像及びコメントの保存画面に移動する。(図の矢印②) さらに、画像及びコメントの保存画面から「画像」ボタンを選択すると、端末の SDcard の画像を全て取得して一覧表示させる画面に移動する。(図の矢印③) そして一覧から画像を選択すると、矢印③から前の画面に戻り、画面上部の枠の部分に選択した画像を反映させる。そして「保存と同期」を選択すると、枠に表示されている画像とテキストボックスに入力されているコメントを SQLite に保存する。

次に、店舗詳細画面にて「サーバーと同期」を選択すると、SQLite から Datastore に画像及び住所が保存する。(図の矢印④)

最後に、店舗詳細画面にて「地図を表示」を選択すると、店舗詳細画面で表示されている店舗の住所を Datastore から検索し、地図の画像をサーバ上に描画し Android 端末で表示する。(図の矢印⑤)

なお、システムの設計には日経ソフトウェア2011年9月号掲載の記事の13ページから41ページ[2]及び、先行研究[9]を参考とした。

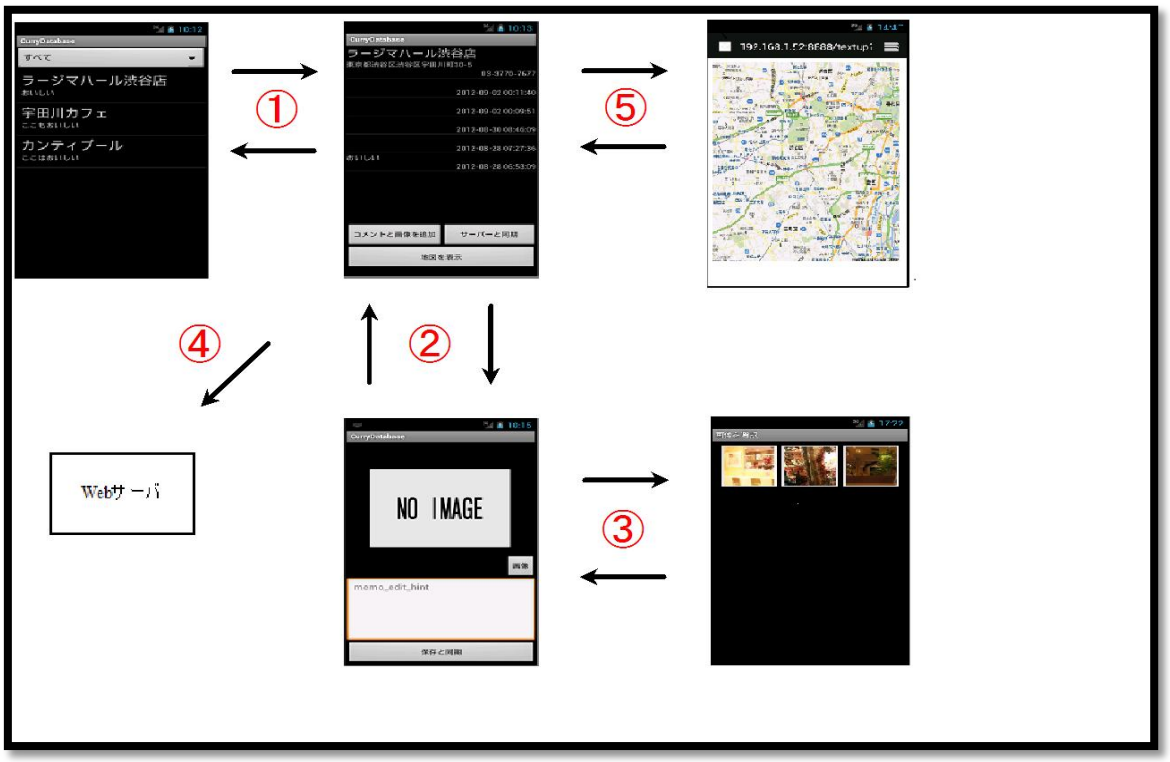

図 3.1　システムを構成する画面のフローチャート

3.2　SQLite に画像を保存

システムを操作するには、画面を表示して画面間を行き来する必要がある。まずシステムを構成する画面は、アクティビティと呼ばれる。また、画面間の行き来やデータの受け渡しを行うための仕組みは、インテントと呼ばれる。Android アプリケーションは、基本的にアクティビティとインテントの組み合わせによって構築されている。

3.2.1　画像及びコメント入力画面に遷移

SQLite に画像を保存するには、まず図 3.2.1 の店舗詳細画面で「画像及びコメントを追加」を選択する。「画像及びコメントを追加」を選択すると、リスト 3.2.1 の onClick メソッドが実行される。onClick メソッドとは、ボタンクリック時に実行されるメソッドであり、オーバーライドしてボタンクリック時の処理を記述することができる。

onClick メソッド内では、まず Intent クラスのオブジェクトを生成する。ここで、リスト内の MemoInputActivity とは、遷移先である、画像及びコメント入力画面を表している。そして、Intent クラスの putExtra メソッドを使用して、店舗を判別するための shopId を渡す。最後に、startActivity メソッドを使用して画像及びコメント入力画面に遷移する。

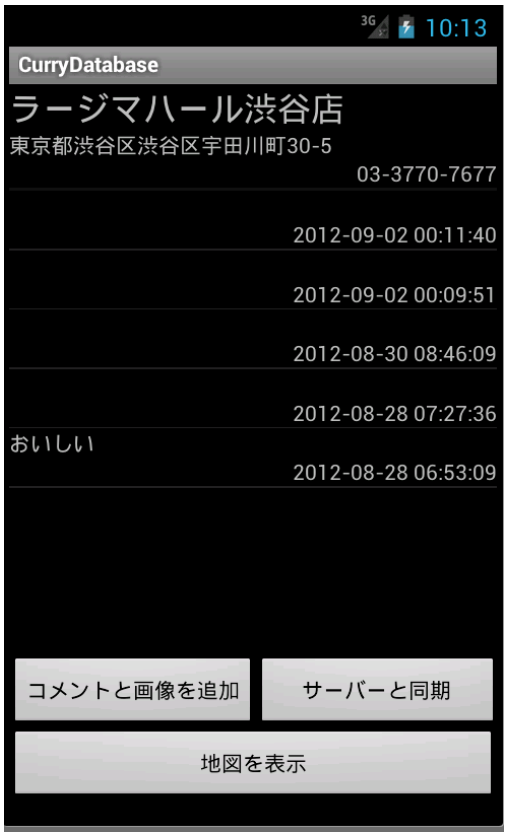

図 3.2.1　店舗詳細画面

```
// 「コメントと画像を追加」を選択時に実行される
class MemoEditClickListener implements OnClickListener {
@Override
        public void onClick(View v) {
                // 画像及びコメント入力画面を呼び出す
                Intent intent = new Intent(self, MemoInputActivity.class);
                //店舗を識別する ID を渡す
                intent.putExtra("shopId", mShopId);
                //画面遷移を実行する
                startActivity(intent);
        }
}
```

リスト 3.2.1　「コメントと画像を追加」を選択時に実行されるプログラム

3.2.2　保存する画像を一覧から選択する

図 3.2.2 左の画像及びコメント入力画面で、「画像」を選択すると図 3.2.2 右の画像一覧画面に遷移する。ここで画像及びコメント入力画面の NOIMAGE の部分は、画像一覧画面から選択した画像を反映させるための枠である。

「画像」を選択した際には、リスト 3.2.2 の onClick メソッドが実行される。ここで startActivityForResult メソッドとは、startActivity メソッドの画面遷移に加えて、遷移先からの返り値を取得することができるメソッドである。今回は、画像一覧画面から選択した画像を返り値として取得している。そして、取得した画像を枠の部分に表示させている。

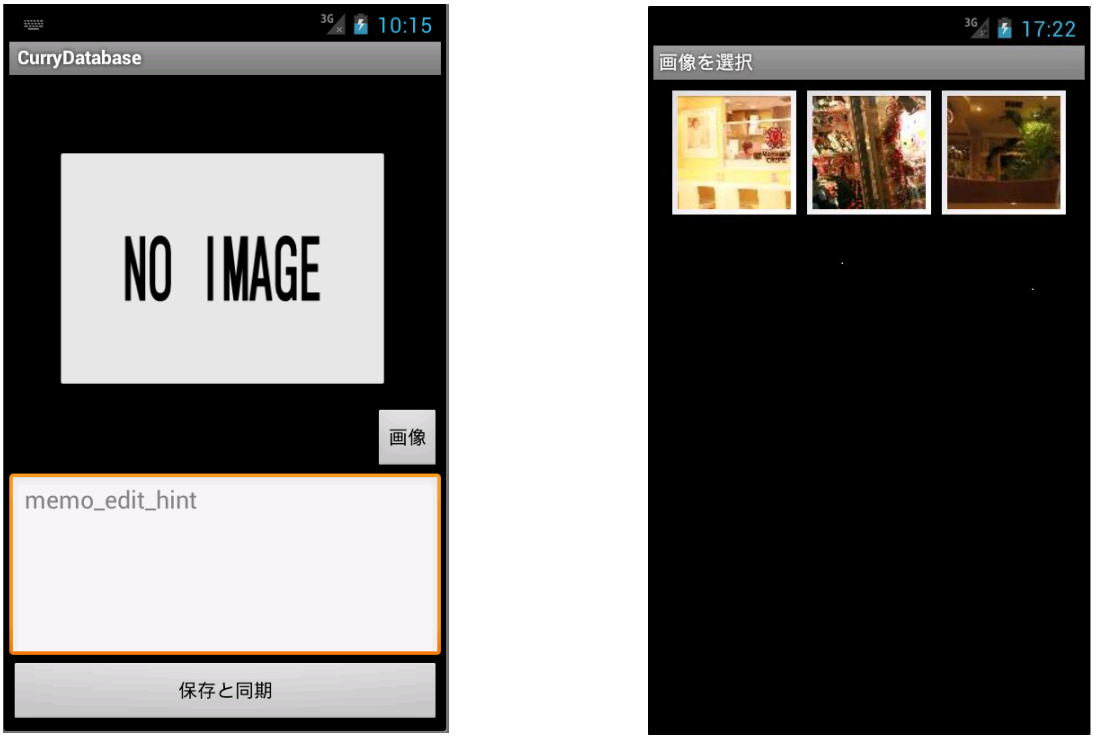

図 3.2.2　画像及びコメント入力画面（左）及び画像一覧画面（右）

```
@Override
public void onClick(View v) {
//Intent のオブジェクトを生成
Intent intent = new Intent(self, GridActivity.class);
//画像を取得するために画像一覧画面にリクエストを行う
startActivityForResult(intent,REQUEST_IMAGE);
}
```

リスト 3.2.2　「画像」を選択時に実行されるプログラム

3.2.3　画像を SQLite に保存

「保存と同期」を選択すると、リスト 3.2.3 のメソッドを実行して、画像及びコメントを SQLite に保存する。ここで、SQLite から Datastore へ画像を送信する際に画像の位置を示すパスが必要になる。そのため、取得した画像から画像のパスを取得し、画像及びコメントと合わせて SQLite に保存する。ContentValues の put メソッドを使用して、キーとなる名前と値のペアを作り、SQLiteDatabase クラスの insert メソッドを使用して、SQLite に保存を行う。

```
public static void insert(SQLiteDatabase db, String memo,  byte[]  blob,  String
imagePath) {

      ContentValues values = new ContentValues();
      values.put(COL_SHOP_ID, shopId);
      values.put(COL_MEMO, memo);
      values.put(IMAGEURI, imagePath);
      values.put(IMAGE, blob);

      db.insert(TABLE_NAME, null, values);
      }
```

リスト 3.2.3　画像及び画像のパスを SQLite に保存するプログラム

3.3　SQLite から Datastore へ画像と住所を保存

SQLite から Datastore へ画像及び住所の文字列を保存するためには、SQLite から画像及び住所の文字列を送信し、サーブレット側で受信して、さらにサーブレット側で Datastore に保存する必要がある。

3.3.1　SQLite から画像を送信

まず、画像をサーブレット側へ送信するには、Apache Mime4J[10] 及び httpmime[11]の二つのライブラリが必要となる。そのため、これらのライブラリのダウンロードを行った。さらに、AndroidManifest.xml に図 3.3.1 の記述を追加した。

画像を送信する際は、容量が大きいのでリスト 3.3.1 のように Multipart 形式で送信する。また、画像ファイルの存在する場所を表す Path を使用して、画像を参照して送信を行

っている。

```
<uses-library android:name="org.apache.http.entity"/>
<uses-library android:name="org.apache.http.james.mime4j"/>
```

図 3.3.1　AndroidManifest.xml に追加するコード

```
DefaultHttpClient client = new DefaultHttpClient();
HttpPost post = new HttpPost(URL);
MultipartEntity entity = new MultipartEntity();

File file = new File(Path);
entity.addPart("IMAGE", new FileBody(file.getAbsoluteFile()));
post.setEntity(entity);
HttpResponse response = client.execute(post);
```

リスト 3.3.1　画像を送信するプログラム

3.3.2　SQLite から住所の文字列を送信

住所の文字列をサーブレット側へ送信する際には、List クラスの add メソッドを使用して、list に文字列を格納する。そして、list を UTF-8 にエンコードしてサーブレットに送信する。UTF-8 とは、日本語などのそのままでは送信できない文字列を、バイト列に変換して送信できるようにするための仕様の一つである。他の仕様として、UTF-16 などがある。また BasicNameValuePair メソッドは、名前と値をペアで定義し、送信先でキーとなる名前を指定することで値を取り出すことができるメソッドである。今回の例では、ADDRESS という名前をキーとして指定することで送信先で値を取り出すことが出来る。

```
DefaultHttpClient client2 = new DefaultHttpClient();
//画像とは異なるサーブレットに送信する
HttpPost post2 = new HttpPost(URL2);
List<NameValuePair> list = new ArrayList<NameValuePair>();
//list に住所を格納
list.add(new BasicNameValuePair("ADDRESS",address));
list.add(new BasicNameValuePair("SHOPID",String.valueOf(shopId)));
//住所のエンコーディングを行う
post2.setEntity(new UrlEncodedFormEntity (list,"UTF-8"));
HttpResponse response = client2.execute(post2);
```

リスト 3.3.2　SQLite から住所の文字列を送信するプログラム

3.3.3　サーブレット側で画像を受信し Datastore に保存

画像データを受信するサーブレットの doPost メソッドで、画像データの読み込み及び Datastore への保存を行う。まず、ServletFileUpload クラスの parseRequest メソッドで Multipart 形式で送信されたデータを list 型で取得する。isFormField()メソッドは、画像などのファイルデータなら false を返し、文字列などのフォームデータなら true を返すメソッドである。今回は画像データなので false が返されるため、if 文が実行されることになる。また、getName メソッドでは画像名を取得している。そして、getInputStream メソッドで、画像の入力バイトストリームを取得し、同じサイズのバイト配列を定義して read メソッドで格納している。そして、バイト列の画像データを BLOB 型に変換して Datstore に保存する。なお、画像の保存には JDO(Java Data Objects)を使用している。JDO とは、データベースにオブジェクトを永続化させるための仕組みである。[12]

```
public void doPost(HttpServletRequest req, HttpServletResponse resp)
                        throws IOException {
        memoryFileItemFactory factory = new memoryFileItemFactory();
        ServletFileUpload upload = new ServletFileUpload(factory);
        try {
                List<FileItem> list = upload.parseRequest(req);
                for (FileItem item : list) {
                        if (!(item.isFormField())) {
                                //画像名を取得
                                fileName = item.getName();
                                if (fileName != null && !"".equals(fileName)) {
                                        //画像のサイズを取得
                                        int size = (int) item.getSize();
                                        byte[] data = new byte[size];
                                        //画像の入力バイトストリームを取得
                                        InputStream in = item.getInputStream();
                                        in.read(data);
                                        //BLOB 型で Datastore に保存
                                        Blob blobImage = new Blob(data);
                                        Date date = new Date();
                //JDO を使用して Datastore に画像を保存
                imagedat img = new imagedat(imgkey,
                                                blobImage
                                                );
                PersistenceManager pm = PMF.get().getPersistenceManager();
                try {
                        pm.makePersistent(img);
                } catch (Exception e) {
                        e.printStackTrace();
                } finally {
                        pm.close();
                }
```

リスト 3.3.3　画像を受信し Datastore に保存するプログラム

3.3.4　サーブレット側で住所の文字列を受信しDatastoreに保存

住所の文字列を受信するサーブレットの doPost メソッドで、文字列のデコード及び Datastore への保存を行う。まず、setCharacterEncoding メソッドを使用して文字列のデコードを行う。デコードを行う際には、住所を送信した時と同じエンコード方式を使用しなければならない。送信時と受信時で異なるエンコード方式を使用した場合、文字化けが起きるためである。そして、getParameter メソッドを使用してキーとなる名前を指定することで、名前とペアで格納された値（住所の文字列）を受け取り、String 型変数 address に格納する。そして、Entity クラスのオブジェクトを生成し、setProperty メソッドを使用して、住所の文字列を ADDRESS プロパティに保存する。Datastore に保存する際には、住所の文字列は String 型で保存する。なお、Datastore では String 型は 500 文字までの制限があり、それ以上の長さの文字列を保存する際には TEXT 型を使用する。

```
public void doPost(HttpServletRequest req, HttpServletResponse resp)
                        throws IOException {
          //送信時と同じエンコーディング方式を使用する
           req.setCharacterEncoding("UTF-8");
          //住所の文字列を取得
           String address=req.getParameter("ADDRESS");

           Entity entity=new Entity(key);
          //Datastore の ADDERESS プロパティに住所を設定する
           entity.setProperty("ADDRESS", address);
           DatastoreService ds=
                   DatastoreServiceFactory.getDatastoreService();
           ds.put(entity);
}
```

リスト 3.3.4　住所の文字列を受信しDatastoreに保存するプログラム

3.4 Datastore から住所を取得

3.4.1　店舗詳細の住所をサーブレットに送信

「地図を表示」を選択すると、まずリスト 3.4.1 の onClick メソッドが実行される。Cursor クラスを使用して、店舗詳細画面の住所の文字列を取得し、その住所の文字列をクエリとして URL に設定している。サーブレットに送信する際は、Uri クラスの parse メソッドを使用して遷移先の URL を指定し、startActivity メソッドでサーブレットに遷移している。

```
        //「地図を表示」押下のイベントリスナ
        class MapClickListener implements OnClickListener {

        @Override
        public void onClick(View v) {
                SQLiteDatabase db = mDbHelper.getReadableDatabase();
                //Cursor クラスを使用して店舗詳細画面の住所を取得
                Cursor c = ShopDao.getByShopId(db, mShopId);
                c.moveToFirst();
                String address1 = getCursorString(c, ShopDao.COL_ADDRESS1);
                String address3 = getCursorString(c, ShopDao.COL_ADDRESS3);
                c.close();
                String address=address1  + address3;

                Intent intent = new Intent(
                Intent.ACTION_VIEW,
                //取得した住所をクエリに設定しサーブレットに遷移
                Uri.parse("http://192.168.1.52:8888/textup?"+address+""));
                startActivity(intent);
                }
        }
```

リスト 3.4.1　「地図を表示」を選択時に店舗詳細の住所を送信するプログラム

3.4.2　Datastore に同じ住所が保存されているか検索

サーブレットに送信された住所の文字列は、リスト 3.4.2 の getQueryString メソッドで取得する。getQueryString メソッドは、URL の？以降のクエリ情報を取得するためのメソッドである。

そして、Datastore を操作するための Datastoreservice クラスのオブジェクト及び、Datastore からの検索を行うための Query クラスのオブジェクトを生成する。そして while ループ内で、getProperty メソッドを用いて Datastore に保存された ADDRESS プロパティの住所を１つずつ検索し、getQueryString メソッドで取得した住所と比較を行う。もし Datastore に同じ住所があれば、jsp ファイルにフォワードして、地図を表示する処理を行う。フォワードとは、リクエスト情報を引き継いでページ遷移を行うことができる仕組みであり、今回は住所の情報を jsp に渡している。また、jsp とは、HTML 内に java のソースコードを埋め込むように記述できる言語である。本研究では、servlet よりも画面表示に適していると考えたため jsp を地図表示に使用した。

```
//「地図を表示」を押したときの店舗詳細画面の住所を取得する
String ad= req.getQueryString();
String add=URLDecoder.decode(ad,"utf-8");
//データストアから住所を受け取る処理を行う
DatastoreService datastoreService
    = DatastoreServiceFactory.getDatastoreService();
Query query = new Query("ADDRESS");
PreparedQuery preparedQuery = datastoreService.prepare(query);
Iterator<Entity> empEntity = preparedQuery.asIterator();
String add2=null;
//Datastore に保存された住所と同じ住所があれば、
  その住所の地図を表示する
while (empEntity.hasNext()){
        Entity entity=empEntity.next();
        if(entity.getProperty("ADDRESS").equals(add)){
        add2=(String) entity.getProperty("ADDRESS");
        }
}
req.setAttribute("msg", add2);
```

```
//住所で地図を表示する JSP ページにフォワードする
String disp = "/mapview.jsp";
RequestDispatcher dispatch = req.getRequestDispatcher(disp);
dispatch.forward(req, resp);
```

リスト 3.4.2　店舗詳細の住所が Datastore に存在するか検索するプログラム

3.5　地図を表示

地図の表示には GoogleStaticMapsAPI[13]を使用する。GoogleStaticMasAPI とは、Google が提供している、URL パラメータを指定することで地図の画像を取得できる API である。本研究では、jsp ファイルの img タグに図 3.5 のように URL を指定した。

まず、ｃenter には地図の画像の中心座標を設定する。本研究では、住所の文字列を設定している。また、zoom には地図のズームレベルを設定する。０から２１までの数値を指定し、数値が大きいほど地図が拡大されて表示される。また、size には地図のサイズを決定する。縦横最大 640 ピクセルまで指定することができる。また、markers にはマーカーの色や座標を指定する。最後に、sensor には現在地情報を使用するか否かを設定する。本研究では現在地情報は使用しない為、false に設定した。

```
http://maps.google.com/maps/api/staticmap?center=<%=msg2%>
&zoom=13&size=640x640&markers=color:blue|<%=msg2%>&sensor=false
```

図 3.5

第４章　実行結果

4.1　SQLite に画像を保存

画像を SQLite に保存する際の実行結果は図 4.1 のようになった。図 4.1 から、まず画像ボタンを選択して、画像一覧画面に移動する。そして画像一覧画面から画像を選択し、選択した画像が枠の部分に反映されていることが確認できる。この後に保存と同期ボタンを選択することで、画像及びコメントを SQLite に保存することができる。

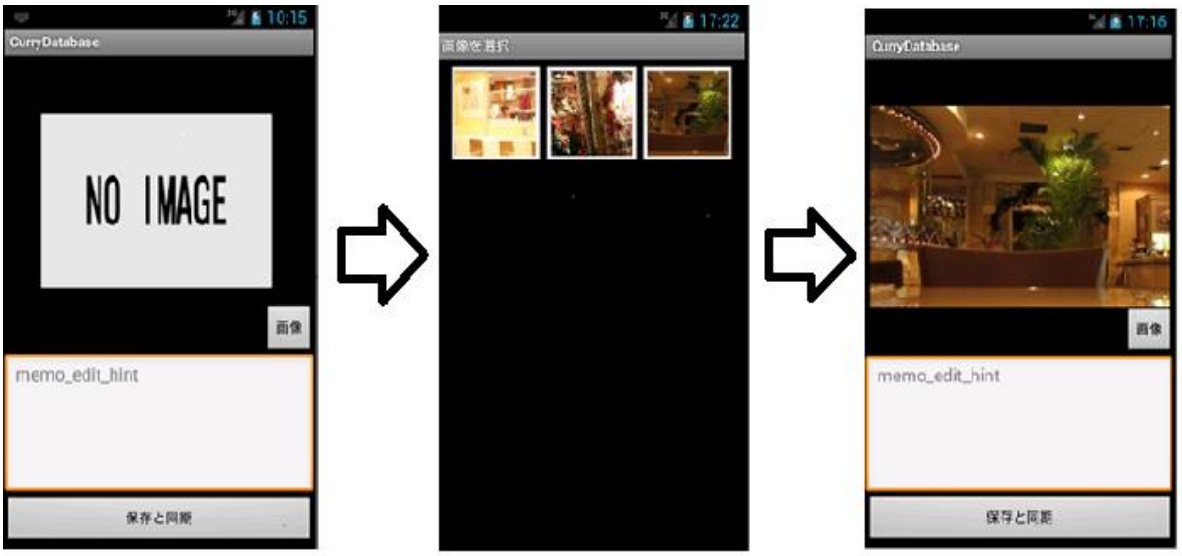

図 4.1　SQLite に画像を保存した実行例

4.2　画像及び住所を SQLite から Datastore に保存

datastore に住所及び画像を保存した結果は図 4.2 のようになった。図 4.2 は Datastore に保存されたエンティティ及びプロパティを確認できる Datastore の管理画面である。
エンティティとは、Datastore に保存されるオブジェクトのことである。[14] またプロパティとは、整数、浮動小数点値、文字列、日付、バイナリデータのいずれかのデータ型を持つ名前付きの値のことである。[14]
まず、上の表の ADDRESS プロパティに住所が保存されていることが確認できる。 上の表では主キーとして、店舗 ID を使用している。
また、下の表の image プロパティに、画像をバイト列で表した BLOB 文字列が保存されてい

ることが確認できる。下の表では主キーとして、取得した画像名を使用している。

ID/Name	ADDRESS
name=1	東京都渋谷区渋谷区宇田川町30-5
name=2	東京都渋谷区渋谷区宇田川町33-1
name=3	東京都渋谷区渋谷区桜丘町16-6

図 4.2　ADDRESS プロパティの一覧

ID/Name	image
name=large.jpg	7856 bytes, SHA-1 = f336a372f702e7bdedfe140c7d8d2d9df078ab72
name=test.png	6667 bytes, SHA-1 = 457339b4e17717795d2fa26a0f69e151eece1e01

図 4.2　image プロパティの一覧

4.3　Datastore から住所を検索 し地図をサーバ上に表示

地図を表示を選択した時の実行結果は下図のようになった。右図から、店舗詳細画面の住所の位置を青いマーカで示した地図が表示されたことが確認できる。

図 4.3　「地図を表示」を選択した時の実行結果

第５章　終論

本研究では、SQLite に画像を保存し、あらかじめ登録しておいた住所と合わせてdatastore に保存し、Datastore から検索した住所を用いて地図を表示するシステムを実装した。

しかし、店舗リスト及び住所などはあらかじめ SQLite に保存されていることを前提としているため、地図表示機能に関しては実用的な機能であるとは言い難い。また、端末内の画像もあらかじめ eclipse 上から登録しておく必要があるため手間がかかり、実用的ではない。そのため、実際に現地行ったときに、端末で撮影した写真を一覧に反映する機能を実装すれば、手間がなくなり有意義であると考えられる。また、店舗の情報を共有するという意味で、店舗に対するユーザ評価を共有する機能や、住所が変更された際に更新する機能の実装も有意義であると考えられ、今後の課題であると考える。

謝辞

本稿執筆にあたり、終始熱心なご指導を頂いた田中章司郎教授に心より御礼申し上げます。また、研究室の皆様からも多大な恩恵を頂きましたこと、深く感謝致します。

なお、本論文、本研究で作成したプログラム及び、データ、並びに関連する発表資料などの全ての知的財産権を本研究の指導教官の田中章司郎教授に譲渡致します。

参考文献

[1]http://www.thinkwithgoogle.com/mobileplanet/ja/downloads/
[2]若命全洋、小川信一、“日経ソフトウェア 2011 年 9 月号”、日経ソフトウェア、2011
[3]http://code.google.com/intl/ja/appengine
[4]http://itpro.nikkeibp.co.jp/article/COLUMN/20110107/355908/
[5]http://java.sun.com/javase/ja/6/download.html
[6]http://www.eclipse.org/downloads/
[7]https://dl-ssl.google.com/android/eclipse/
[8]http://developer.android.com/sdk/index.html
[9]組込型データベースとキーバリュー型データベースの画像連携に関する初歩的研究
―どこでも画像アップロードの試作―　（吉留明樹　２０１１）
[10]http://hc.apache.org/downloads.cgi
[11]http://james.apache.org/download.cgi#Apache_Mime4J
[12]https://developers.google.com/appengine/docs/java/gettingstarted/usingdatastore?hl=ja
[13]https://developers.google.com/maps/documentation/staticmaps/?hl=ja
[14] https://developers.google.com/appengine/docs/python/datastore/entities?hl=ja

付録

付録１：店舗一覧画面を表示するプログラム ShopActivity.java

```
package net.vvakame.currydb;


import java.util.ArrayList;
import java.util.List;


import net.vvakame.currydb.db.CurryDbHelper;
import net.vvakame.currydb.db.MemoDao;
import net.vvakame.currydb.db.ShopDao;
import android.app.ListActivity;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import android.widget.Spinner;


/**
 * 店舗一覧画面の Activity
 *
 * @author vvakame
 */
public class ShopActivity extends ListActivity {


		final Context self = this;
```

```
CurryDbHelper mDbHelper;

String mAddress2;

@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.shop_list);

        mDbHelper = new CurryDbHelper(this);

        // 区による絞り込みの Spinner の準備
        Cursor c = ShopDao.getAddres2List(mDbHelper.getReadableDatabase());
        List<String> addressList = cursorToStrings(c, ShopDao.COL_ADDRESS2);
        c.close();
        addressList.add(0, getString(R.string.all));

        // アダプタに値を詰め替える
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                        android.R.layout.simple_spinner_item);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        for (String address : addressList) {
                adapter.add(address);
        }

        // スピナーにアダプタとイベントハンドラの設定を行う
        Spinner spinner = (Spinner) findViewById(R.id.spinner);
        spinner.setAdapter(adapter);
        spinner.setOnItemSelectedListener(new Address2SelectedListener());

        // Shop 一覧表示
        ensureShopList(null);
}

static List<String> cursorToStrings(Cursor c, String column) {
        List<String> list = new ArrayList<String>();
```

```
                // データ件数 0 なら空のリストを返す
                if (c.getCount() == 0) {
                        return list;
                }
                // 指定したカラムがカーソルのどこかを探す
                final int idx = c.getColumnIndex(column);
                // カーソルの頭出し
                c.moveToFirst();
                // 次の要素がある限り次々と値を読み出す
                do {
                        list.add(c.getString(idx));
                } while (c.moveToNext());
                return list;
        }

        void ensureShopList(String address2) {
                // 店舗一覧の取得
                SQLiteDatabase db = mDbHelper.getReadableDatabase();
                Cursor c;
                if (address2 == null) {
                        c = ShopDao.getAllWithNewestMemo(db);
                } else {
                        c = ShopDao.getAllWithNewestMemoByAddress2(db, address2);
                }

                // Cursor の close を Activity(のライフサイクル)に任せる
                startManagingCursor(c);

                // ListView に検索結果を表示する
                ListView list = getListView();
                SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
                                R.layout.shop_list_row, c, new String[] { ShopDao.COL_NAME,
                                                MemoDao.COL_MEMO }, new int[] { R.id.shop,
R.id.memo });
                list.setAdapter(adapter);
                list.setOnItemClickListener(new ShopItemClickListener());
```

```
}

// スピナーで区を選択した時のイベントリスナ
class Address2SelectedListener implements OnItemSelectedListener {

	@Override
	public void onItemSelected(AdapterView<?> paramAdapterView,
				View paramView, int paramInt, long paramLong) {

		//address2 に選択した区を代入
		String address2 = (String) paramAdapterView.getAdapter().getItem(
					paramInt);
		//スピナーですべてを選択した時
		if (getString(R.string.all).equals(address2)) {
			address2 = null;
		}
		mAddress2 = address2;
		//Shop 一覧表示
		ensureShopList(address2);
	}

	@Override
	public void onNothingSelected(AdapterView<?> paramAdapterView) {

		mAddress2 = null;
		ensureShopList(null);
	}
}

// 店舗リストをタップした時のイベントリスナ
class ShopItemClickListener implements OnItemClickListener {

	@Override
	public void onItemClick(AdapterView<?> parent, View view, int position,
				long id) {
```

```
                                Intent intent = new Intent(self, MemoActivity.class);
                                intent.putExtra("shopId", id);
                                startActivity(intent);
                        }
                }

                @Override
                protected void onDestroy() {
                        mDbHelper.close();
                        super.onDestroy();
                }
}
```

付録２：店舗詳細画面を表示するプログラム MemoActivity.java

```
package net.vvakame.currydb;

//import net.vvakame.currydb.ShopActivity.SyncClickListener;
import net.vvakame.currydb.db.CurryDbHelper;
import net.vvakame.currydb.db.MemoDao;
import net.vvakame.currydb.db.ShopDao;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.app.ListActivity;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemLongClickListener;
import android.widget.SimpleCursorAdapter;
import android.widget.TextView;

/**
 * 選択した店舗のメモ一覧画面の Activity
 *
 * @author vvakame
 */
public class MemoActivity extends ListActivity {

          final Context self = this;
```

```
CurryDbHelper mDbHelper;

long mShopId;

@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.memo_list);

        // 前の画面よりお店の ID を受け取る
        mShopId = getIntent().getLongExtra("shopId", -1);
        if (mShopId == -1) {
                finish();
                return;
        }

        mDbHelper = new CurryDbHelper(this);
        SQLiteDatabase db = mDbHelper.getReadableDatabase();

        // 店舗情報を表示
        {
                Cursor c = ShopDao.getByShopId(db, mShopId);
                c.moveToFirst();
                String shopName = getCursorString(c, ShopDao.COL_NAME);
                String address1 = getCursorString(c, ShopDao.COL_ADDRESS1);
                String address3 = getCursorString(c, ShopDao.COL_ADDRESS3);
                String tel = getCursorString(c, ShopDao.COL_TEL);
                c.close();
                String address=address1 + address3;

                setTextViewToText(R.id.shop_name, shopName);
                setTextViewToText(R.id.address, address);
                setTextViewToText(R.id.tel, tel);
        }

        // メモ一覧の表示
        ensureMemoList();

        // イベントリスナの設定
        getListView().setOnItemLongClickListener(new MemoDeleteClickListener());
        findViewById(R.id.insert).setOnClickListener(
                        new MemoEditClickListener());

        // 「サーバーと同期」ボタンにイベントハンドラを設定
        findViewById(R.id.sync).setOnClickListener(new SyncClickListener());

        //「地図を表示」ボタンにイベントハンドラを設定
        findViewById(R.id.map).setOnClickListener(new MapClickListener());
}

static String getCursorString(Cursor c, String colName) {
        return c.getString(c.getColumnIndex(colName));
}
```

```
	void setTextViewToText(int id, String text) {
		TextView textView = (TextView) findViewById(id);
		textView.setText(text);
	}

	void ensureMemoList() {
		SQLiteDatabase db = mDbHelper.getReadableDatabase();
		Cursor c = MemoDao.getByShopId(db, mShopId);
		startManagingCursor(c);

		SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
				R.layout.memo_list_row, c, new String[] { MemoDao.COL_MEMO,
					MemoDao.COL_AT }, new int[] { R.id.memo,
R.id.at });
		setListAdapter(adapter);
	}

	// 「コメントと画像を追加」押下のイベントリスナ
	class MemoEditClickListener implements OnClickListener {

		@Override
		public void onClick(View v) {
			// 画像及びコメント入力画面を呼び出す
			Intent intent = new Intent(self, MemoInputActivity.class);
			intent.putExtra("shopId", mShopId);
			startActivity(intent);
		}
	}

	// メモ一覧のアイテムロングタップのイベントリスナ
	class MemoDeleteClickListener implements OnItemLongClickListener {

		@Override
		public boolean onItemLongClick(AdapterView<?> paramAdapterView,
				View paramView, int paramInt, final long paramLong) {
			// paramLong は MEMO テーブルの _id の値

			// 消していいですか？ はい/いいえ 選択ダイアログの表示
			Builder builder = new AlertDialog.Builder(self);
			builder.setTitle(R.string.confirm_memo_delete_title);
			builder.setMessage(R.string.confirm_memo_delete);
			builder.setPositiveButton(R.string.yes,
					new DialogInterface.OnClickListener() {

						@Override
						public void onClick(
								DialogInterface
paramDialogInterface,
								int paramInt) {

							// はい なら消す
							SQLiteDatabase db =
mDbHelper.getWritableDatabase();
							MemoDao.delete(db, paramLong);
```

```
                                                                ensureMemoList();
                                                        }
                                                });
                                builder.setNegativeButton(R.string.no,
                                                new DialogInterface.OnClickListener() {

                                                        @Override
                                                        public void onClick(
                                                                        DialogInterface
paramDialogInterface,
                                                                        int paramInt) {
                                                                // いいえ ならスルー(ダイアログ
閉じる)
                                                        }
                                                });
                                builder.setCancelable(true); // 戻るボタンでキャンセルを可能にする
                                builder.create().show();

                                return true;
                        }
                }


                //「サーバと同期」押下のイベントリスナ
                class SyncClickListener implements OnClickListener {

                        @Override
                        public void onClick(View v) {
                                AppengineSync.fullSync(self,mShopId);
                                ensureMemoList();
                        }
                }

                //「地図を表示」押下のイベントリスナ
                class MapClickListener implements OnClickListener {

                        @Override
                        public void onClick(View v) {
                                SQLiteDatabase db = mDbHelper.getReadableDatabase();
                                Cursor c = ShopDao.getByShopId(db, mShopId);
                                c.moveToFirst();
                                String address1 = getCursorString(c, ShopDao.COL_ADDRESS1);
                                String address3 = getCursorString(c, ShopDao.COL_ADDRESS3);
                                c.close();
                                String address=address1  + address3;

                                Intent intent = new Intent(
                                                Intent.ACTION_VIEW,
                                                Uri.parse("http://192.168.1.52:8888/textup?"+address+""));
                                        startActivity(intent);
                        }
                }

                @Override
```

```
        protected void onDestroy() {
                mDbHelper.close();
                super.onDestroy();
        }
}
```

付録３：画像及びコメント入力画面を表示するプログラム MemoInputActivity.java

```
package net.vvakame.currydb;


import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.InputStream;


import net.vvakame.currydb.db.CurryDbHelper;
import net.vvakame.currydb.db.MemoDao;
import android.app.Activity;
import android.content.ContentResolver;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Bitmap;
import android.graphics.Bitmap.CompressFormat;
import android.graphics.BitmapFactory;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Toast;


/**
 * メモ入力画面の Activity
 *
```

```
 * @author vvakame
 */
public class MemoInputActivity extends Activity implements OnClickListener {

	final Context self = this;
	private static final int  REQUEST_IMAGE = 0;
	long mShopId;
	private Button button1;
	private Button button3;
	private Bitmap bm1;
	private Uri imageUri;
	static String imagePath;
	//起動時に呼び出される
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.memo_input);

		mShopId = getIntent().getLongExtra("shopId", -1);
		if (mShopId == -1) {
			finish();
			return;
		}
		//「保存と同期」ボタンを押した際のクリックリスナ
		button1=(Button)findViewById(R.id.add);
		button1.setOnClickListener(this);

		//「画像」ボタンを押した際の処理のクリックリスナ
		button3=(Button)findViewById(R.id.Button03);
		button3.setOnClickListener(this);

	}

	@Override
	public void onClick(View v) {
		if(v==button1){//コメントと画像を追加を押した時の処理
```

```
                    CurryDbHelper dbHelper = new CurryDbHelper(self);
                    SQLiteDatabase db = dbHelper.getWritableDatabase();

                    EditText editText = (EditText) findViewById(R.id.memo_edit);
                    String memo = editText.getText().toString();

                    //Bitmap を bytearray に変換
                    ByteArrayOutputStream baoStream = new ByteArrayOutputStream();
                    bm1.compress(CompressFormat.PNG, 90, baoStream );

                    try{
                              baoStream.flush();
                              byte[] blob = baoStream.toByteArray();
                              baoStream.close();
                              //画像のファイルパスを取得
                              imagePath=getPath(self,imageUri);
                              //コメント、画像、及び画像の Uri を SQLite に保存
                              MemoDao.insert(db, mShopId, memo,blob,imagePath);
                    }catch(Exception e){
                              e.printStackTrace();
                    }
                    dbHelper.close();

                    finish();

          }else{//画像ボタンを押した時の処理

                    //画像を取得するためのリクエストコードを定義
                    Intent intent = new Intent(self, GridActivity.class);
                    startActivityForResult(intent,REQUEST_IMAGE);
          }
}

//一覧表示された画像をクリックした時に呼び出される
@Override
protected void onActivityResult(int requestCode
```

```
                    ,int resultCode,Intent intent) {


                    if(requestCode==REQUEST_IMAGE&&resultCode==RESULT_OK){
                              //画像の Uri を取得
                        imageUri = (Uri) intent.getData();
            InputStream in = null;
            try {
                in = getContentResolver().openInputStream(imageUri);

                //読み込みの際のオプションを設定
                                        BitmapFactory.Options options = new BitmapFactory.Options();
                                        // 任意のサイズに変換して読み込み
                                        options.inSampleSize = 1;


                                    bm1 = BitmapFactory
                        .decodeStream(in, null, options);


                                        //NO_IMAGE の部分に画像を表示する
                                        ImageView
changeimageView=(ImageView)findViewById(R.id.ImageView01);
                                        changeimageView.setImageBitmap(bm1);
                                }catch (FileNotFoundException e) {
                                }
                    }



                    button1=(Button)findViewById(R.id.add);
                    button1.setOnClickListener(this);


                    button3=(Button)findViewById(R.id.Button03);
                    button3.setOnClickListener(this);
          }


          //画像のファイルパスを取得
          public static String getPath(Context context, Uri uri) {
              ContentResolver contentResolver = context.getContentResolver();
```

```
                String[] columns = { MediaStore.Images.Media.DATA };
                Cursor cursor = contentResolver.query(uri, columns, null, null, null);
                cursor.moveToFirst();
                String path = cursor.getString(0);
                cursor.close();
                return path;
            }
}
```

付録４：取得した画像を一覧表示させるプログラム GridActivity.java

```
package net.vvakame.currydb;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.Serializable;

import android.app.Activity;
import android.content.ContentResolver;
import android.content.Intent;
import android.database.Cursor;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.widget.ImageView;
import android.widget.Toast;

public class GridActivity extends Activity  {
  private static final int  REQUEST_GALLERY = 0;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.imageview);

    setResult(Activity.RESULT_CANCELED);

    // ギャラリー呼び出し
    Intent intent = new Intent();
    intent.setType("image/*");
// 画像を１つ選択して返す
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(intent,  REQUEST_GALLERY);

 }

 //画像を一つ選択してギャラリーから戻ってきたときに呼び出される
 @Override
 protected void onActivityResult(int requestCode, int resultCode, Intent data) {
            if(requestCode ==  REQUEST_GALLERY && resultCode == RESULT_OK) {
                        try {

                                    InputStream in = getContentResolver().openInputStream(data.getData());
                                    Bitmap img = BitmapFactory.decodeStream(in);
                                    in.close();

                                    //インテントへのパラメータの付加
                                    data.putExtra("key", img);
                                    //MemoInputAcctivityに結果コードを通知
                                    setResult(Activity.RESULT_OK,data);
                                    //アクティビティの終了
                                    finish();

                        } catch (FileNotFoundException e) {
                                    e.printStackTrace();
                        } catch (IOException e) {
                                    e.printStackTrace();
                        }catch (NullPointerException e) {
```

```
                                e.printStackTrace();
                        }
                }
        }
```

付録５：画像及び住所を Datastore に送信するプログラム AppEngineSync.java

```
package net.vvakame.currydb;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import net.vvakame.currydb.db.CurryDbHelper;
import net.vvakame.currydb.db.MemoDao;
import net.vvakame.currydb.db.ShopDao;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
//import org.apache.http.NameValuePair;
//import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;

import org.apache.http.entity.mime.HttpMultipartMode;
import org.apache.http.entity.mime.MultipartEntity;
import org.apache.http.entity.mime.content.ContentBody;
import org.apache.http.entity.mime.content.FileBody;
import org.apache.http.entity.mime.content.StringBody;
```

```
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
import org.w3c.dom.Text;

import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;
import net.vvakame.currydb.MemoInputActivity;

public class AppengineSync {




	public static final String URL = "http://192.168.1.52:8888/imageup";
	public static final String URL2 = "http://192.168.1.52:8888/textup";
	public static void fullSync(Context context,long mShopID){
		CurryDbHelper dbHelper = new CurryDbHelper(context);
		SQLiteDatabase db = dbHelper.getWritableDatabase();

		// 全件 POST
		{

			//shopID に該当する店舗のレコードを MEMO テーブルから引き出す
			Cursor c = MemoDao.getByShopId(db,mShopID);
			if (c.moveToFirst()) {

				int id = c.getColumnIndex("_id");
				int shopId = c.getColumnIndex(MemoDao.COL_SHOP_ID);
				int imagePath=c.getColumnIndex(MemoDao.IMAGEURI);
				int blobnum=c.getColumnIndex(MemoDao.IMAGE);
```

```
                                        do {

                                                //店舗 ID から住所を取得
                                                Cursor cu = ShopDao.getByShopId(db, mShopID);
                                                cu.moveToFirst();
                                                String      address1      =      getCursorString(cu,
ShopDao.COL_ADDRESS1);
                                                String      address3      =      getCursorString(cu,
ShopDao.COL_ADDRESS3);
                                                cu.close();

                                                long memoId = c.getLong(id);
                                                String address=address1+address3;

                                                if(c.getString(imagePath)!=null){
                                                post(memoId, c.getLong(shopId), c.getString(imagePath)
                                                                        ,c.getBlob(blobnum),address);
                                                post2(memoId,                           c.getLong(shopId),
c.getString(imagePath)
                                                                        ,c.getBlob(blobnum),address);
                                                }
                                        } while (c.moveToNext());
                                }
                                c.close();
                        }




                        dbHelper.close();
                }

                //データストアに画像を post する
                static boolean post(long memoId, long shopId
                                ,String Path, byte[] blob, String address){
```

```
            DefaultHttpClient client = new DefaultHttpClient();
            HttpPost post = new HttpPost(URL);
            MultipartEntity entity = new MultipartEntity();

            try {
                    File file = new File(Path);
                entity.addPart("IMAGE", new FileBody(file.getAbsoluteFile()));
                    post.setEntity(entity);
                    HttpResponse response = client.execute(post);
                    return 200 == response.getStatusLine().getStatusCode();

            } catch (IOException e) {
                    Log.w("Curry", e.getLocalizedMessage(), e);

            } catch (NullPointerException e) {
                    e.printStackTrace();



            }
            return false;
    }

    //データストアに住所を post する
    static boolean post2(long memoId, long shopId
                    ,String Path, byte[] blob, String address){

            try {
            DefaultHttpClient client2 = new DefaultHttpClient();
            HttpPost post2 = new HttpPost(URL2);
            List<NameValuePair> list = new ArrayList<NameValuePair>();
            list.add(new BasicNameValuePair("ADDRESS",address));
            list.add(new BasicNameValuePair("SHOPID",String.valueOf(shopId)));
            post2.setEntity(new UrlEncodedFormEntity (list,"UTF-8"));
            HttpResponse response = client2.execute(post2);
            return 200 == response.getStatusLine().getStatusCode();
```

```
                } catch (IOException e) {
                        Log.w("Curry", e.getLocalizedMessage(), e);

                } catch (NullPointerException e) {
                        e.printStackTrace();



                }
                return false;
        }


        static String getCursorString(Cursor c, String colName) {
                return c.getString(c.getColumnIndex(colName));
        }
}
```

付録６：SQLite のデータベーステーブルを作成し、店舗の情報を登録するプログラム

CurryDbHelper.java

```
package net.vvakame.currydb.db;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class CurryDbHelper extends SQLiteOpenHelper {
  static final String DB_NAME = "curry.db";
  static final int DB_VERSION = 1;

  public CurryDbHelper(Context context) {
    super(context, DB_NAME, null, DB_VERSION);
  }

  //db オブジェクトが生成されたときに実行される
  @Override
```

```
public void onCreate(SQLiteDatabase db) {
  // 店舗マスター作成
  db.execSQL("CREATE TABLE SHOP (" +
    "_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "NAME TEXT NOT NULL," +
    "ADDRESS1 TEXT NOT NULL," +
    "ADDRESS2 TEXT NOT NULL," +
    "ADDRESS3 TEXT NOT NULL," +
    "TEL TEXT NOT NULL" +
  ")");
  // メモテーブル作成
  db.execSQL("CREATE TABLE MEMO (" +
    "_id INTEGER PRIMARY KEY AUTOINCREMENT," +
    " SHOP_ID INTEGER NOT NULL," +
    " MEMO TEXT," +
    " IMAGEURI TEXT,"+
    " IMAGE BLOB,"+
    " AT TEXT DEFAULT CURRENT_TIMESTAMP" +
  ")");
   insertShopData(db);
   insertMemoData(db);
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion,
  int newVersion) {
  // DB のスキーマをバージョンアップしたい場合に利用
}

public void insertShopData(SQLiteDatabase db) {
          db.beginTransaction();
          ShopDao.insert(db, 1, "ラージマハール渋谷店","東京都","渋谷区"
            ,"渋谷区宇田川町 30-5","03-3770-7677");
          ShopDao.insert(db, 2, "宇田川カフェ", "東京都", "渋谷区",
            "渋谷区宇田川町 33-1", "03-5784-3134");
          ShopDao.insert(db, 3, "カンティプール", "東京都", "渋谷区",
```

```
                "渋谷区桜丘町 16-6", "03-3770-5358");

            db.setTransactionSuccessful();
            db.endTransaction();
        }

        public void insertMemoData(SQLiteDatabase db) {
            db.beginTransaction();
            MemoDao.insert(db, 1, "おいしい",null,null);
            MemoDao.insert(db, 2, "ここもおいしい",null,null);
            MemoDao.insert(db, 3, "ここはおいしい",null,null);
            db.setTransactionSuccessful();
            db.endTransaction();
        }
}
```

付録７：SQLite にコメント、画像、画像のパスを保存するためのプログラム MemoDao.java

```
package net.vvakame.currydb.db;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;

import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.net.Uri;

public class MemoDao {

        public static final String TABLE_NAME = "MEMO";

        public static final String COL_SHOP_ID = "SHOP_ID";
        public static final String COL_MEMO = "MEMO";
        public static final String COL_AT = "AT";
        public static final String IMAGE = "IMAGE";
```

```
        public static final String IMAGEURI = "IMAGEURI";

        public static Cursor getByShopId(SQLiteDatabase db, long shopId) {
                return db.query(TABLE_NAME, null, COL_SHOP_ID + " = ?",
                                new String[] { String.valueOf(shopId) }, null, null, COL_AT
                                        + " DESC");
        }
      //SQLiteにコメント、画像、画像のパスを保存する
        public static void insert(SQLiteDatabase db, long shopId, String memo,byte[] blob,String imagePath)
{

                ContentValues values = new ContentValues();
                values.put(COL_SHOP_ID, shopId);
                values.put(COL_MEMO, memo);
                values.put(IMAGEURI, imagePath);
                values.put(IMAGE, blob);
                //3つめの引数は一行分のデータセットを表すContextValues?オブジェクト
                db.insert(TABLE_NAME, null, values);
        }

        public static void delete(SQLiteDatabase db, long _id) {
                db.delete(TABLE_NAME, "_id = ?", new String[] { String.valueOf(_id) });
        }

        public static Cursor getAll(SQLiteDatabase db) {
                return db.query(TABLE_NAME, null, null, null, null, null, null);
        }

        public static void forceInsert(SQLiteDatabase db, long id, long shopId,
                        String memo, Date at) {

                delete(db, id);

                SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
                df.setTimeZone(TimeZone.getTimeZone("GMT"));
```

```
                ContentValues values = new ContentValues();
                values.put("_id", id);
                values.put(COL_SHOP_ID, shopId);
                values.put(COL_MEMO, memo);
                values.put(COL_AT, df.format(at));

                db.insert(TABLE_NAME, null, values);
        }

        // 画面表示には利用しないが test で便利なので用意
        static int count(SQLiteDatabase db) {
                Cursor c = db.rawQuery("SELECT count(*) as cnt FROM " + TABLE_NAME,
                                null);
                c.moveToFirst();
                int count = c.getInt(c.getColumnIndex("cnt"));
                c.close();
                return count;
        }
}
```

付録８：店舗情報のデータベースにアクセスするプログラム ShopDao.java

```
package net.vvakame.currydb.db;

import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;

public class ShopDao {

        public static final String TABLE_NAME = "SHOP";

        public static final String COL_NAME = "NAME";
        public static final String COL_ADDRESS1 = "ADDRESS1";
        public static final String COL_ADDRESS2 = "ADDRESS2";
        public static final String COL_ADDRESS3 = "ADDRESS3";
        public static final String COL_TEL = "TEL";
```

```
	//最も最近のコメントを返す
	public static Cursor getAllWithNewestMemo(SQLiteDatabase db) {
		String sql = "SELECT S.*, M.MEMO AS MEMO, MAX(M.AT) FROM SHOP S LEFT OUTER JOIN MEMO M
ON S._id = M.SHOP_ID GROUP BY S._id";
		return db.rawQuery(sql, null);
	}

	public static Cursor getAllWithNewestMemoByAddress2(SQLiteDatabase db,
			String address2) {
		String sql = "SELECT S.*, M.MEMO AS MEMO, MAX(M.AT) FROM SHOP S LEFT OUTER JOIN MEMO M
ON S._id = M.SHOP_ID WHERE S.ADDRESS2 = ? GROUP BY S._id";
		return db.rawQuery(sql, new String[] { address2 });
	}
	//店舗 ID によって店舗の情報を取得
	public static Cursor getByShopId(SQLiteDatabase db, long shopId) {
		return db.query(TABLE_NAME, null, "_id = ?",
				new String[] { String.valueOf(shopId) }, null, null, null);
	}

	public static Cursor getAddres2List(SQLiteDatabase db) {
		return db.query(true, TABLE_NAME, new String[] { COL_ADDRESS2 }, null,
				null, null, null, null, null);
	}




	// Shop の insert は機能上存在しないが test で便利なので用意 package private.
	static void insert(SQLiteDatabase db, int shopId, String name,
			String address1, String address2, String address3, String tel) {

		ContentValues values = new ContentValues();
		values.put("_id", shopId);
		values.put(COL_NAME, name);
		values.put(COL_ADDRESS1, address1);
```

```
                values.put(COL_ADDRESS2, address2);
                values.put(COL_ADDRESS3, address3);
                values.put(COL_TEL, tel);

                db.insert(TABLE_NAME, null, values);
        }

        // 画面表示には利用しないが test で便利なので用意
        static int count(SQLiteDatabase db) {
                Cursor c = db.rawQuery("SELECT count(*) as cnt FROM " + TABLE_NAME,
                                null);
                c.moveToFirst();
                int count = c.getInt(c.getColumnIndex("cnt"));
                c.close();
                return count;
        }
}
```

付録９：Datastore に住所を保存し ADDRESS エンティティを登録するプログラム textUpServlet.java

```
package images;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLDecoder;
import java.net.URLEncoder;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.util.logging.Logger;
```

```
import javax.imageio.ImageIO;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.google.appengine.api.datastore.DatastoreService;
import com.google.appengine.api.datastore.DatastoreServiceFactory;
import com.google.appengine.api.datastore.Entity;
import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.datastore.KeyFactory;
import com.google.appengine.api.datastore.PreparedQuery;
import com.google.appengine.api.datastore.Query;
import com.google.appengine.api.images.Image;
import static com.google.appengine.api.datastore.FetchOptions.Builder.*;

public class textUpServlet extends HttpServlet {
             /**
           *
           */
          private static final long serialVersionUID = 1L;

          public void doPost(HttpServletRequest req, HttpServletResponse resp)
                                        throws IOException {

                        req.setCharacterEncoding("UTF-8");
                        String address=req.getParameter("ADDRESS");
                        String id=req.getParameter("SHOPID");
                    //   address = new String(address.getBytes("ISO-8859-1"), "UTF-8");
                        Date date = new Date();

                        Key key=KeyFactory.createKey("ADDRESS", id);
                        Entity entity=new Entity(key);
                        entity.setProperty("ADDRESS", address);
```

```
                entity.setProperty("DATE", date);
                DatastoreService ds=
                                DatastoreServiceFactory.getDatastoreService();
                ds.put(entity);



}



@SuppressWarnings("deprecation")
public void doGet(HttpServletRequest req, HttpServletResponse resp)
                throws IOException {

        //Android端末から送信された住所を取得する
        String ad= req.getQueryString();
        String add=URLDecoder.decode(ad,"utf-8");
        //データストアから住所を受け取る処理を行う
        DatastoreService datastoreService
    = DatastoreServiceFactory.getDatastoreService();
        Query query = new Query("ADDRESS");



        PreparedQuery preparedQuery = datastoreService.prepare(query);

        Iterator<Entity> empEntity = preparedQuery.asIterator();
        String add2=null;
        while (empEntity.hasNext()){
                Entity entity=empEntity.next();
                if(entity.getProperty("ADDRESS").equals(add)){
                        add2=(String) entity.getProperty("ADDRESS");
                }
        }

        req.setAttribute("msg", add2);

        //受け取った住所で地図を表示するJSPページに遷移する
```

```
                        String disp = "/mapview.jsp";
                          RequestDispatcher dispatch = req.getRequestDispatcher(disp);

                          try {
                                  dispatch.forward(req, resp);
                        } catch (ServletException e) {
                                  e.printStackTrace();
                        }
          }
}
```

付録１０：image エンティティを Datastore に登録するプログラム imagedat.java

```
package images;
import java.util.Date;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.IdentityType;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;
import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.datastore.Blob;

//永続化対象のクラスであることを宣言する
@PersistenceCapable(identityType = IdentityType.APPLICATION)
public class imagedat {
          //オブジェクトを一意に判断する為のフィールドであることを宣言する。
    @PrimaryKey
    //永続化対象のフィールドであることを宣言する (@Persistent)また、永続化時に ID が自動採番される。
    @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
    private Key imgkey;
          @Persistent
    private Blob image;
```

```
    public imagedat(Key imgkey,
                                        Blob image
                                        ){
           this.imgkey = imgkey;
         this.image = image;

    }



    public Key getImgkey() { return imgkey; }
    public Blob getContent() { return image; }



    public void setImgkey(Key imgkey) { this.imgkey = imgkey; }
    public void setContent(Blob image) { this.image = image; }



}
```

付録１１：Datastore に画像を保存するプログラム imageUpServlet.java

```
package images;
import images.imagedat;
import images.PMF;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
//import java.io.OutputStream;
```

```
import java.util.Date;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.List;
import javax.jdo.PersistenceManager;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletOutputStream;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemIterator;
import org.apache.commons.fileupload.FileItemStream;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.apache.commons.fileupload.util.Streams;

import com.google.appengine.api.images.Image;
import com.google.appengine.api.images.ImagesService;
import com.google.appengine.api.images.ImagesServiceFactory;
import com.google.appengine.api.images.Transform;
import com.google.appengine.api.images.ImagesService.OutputEncoding;
import com.google.appengine.api.datastore.Key;
import com.google.appengine.api.datastore.KeyFactory;



import com.google.appengine.api.datastore.Blob;

@SuppressWarnings("serial")
public class imageUpServlet extends HttpServlet {
          private static final long serialVersionUID = 1L;
    String guestname = "";
    String email = "";
    String comment = "";
    String filetitle = "";
    Key  imgkey;
    String fileName = "";
```

```
public void doPost(HttpServletRequest req, HttpServletResponse resp)
                        throws IOException {
      memoryFileItemFactory factory = new memoryFileItemFactory();
      ServletFileUpload upload = new ServletFileUpload(factory);
      //resp.setContentType("image/png");
      resp.setContentType("image/jpeg");
      //OutputStream out = resp.getOutputStream();
      ServletOutputStream out = resp.getOutputStream();
      try {
                        List<FileItem> list = upload.parseRequest(req);
                        for (FileItem item : list) {
                                  if (!(item.isFormField())) {
                                            fileName = item.getName();
                                            if (fileName != null && !"".equals(fileName)) {
                                                      int size = (int) item.getSize();
                                                      byte[] data = new byte[size];
                                                      InputStream in = item.getInputStream();
                                                      in.read(data);
                        ImagesService imagesService = ImagesServiceFactory.getImagesService();
                        Image oldImage = ImagesServiceFactory.makeImage(data);
                        Transform resize = ImagesServiceFactory.makeResize(900, 150);
                //Image newImage = imagesService.applyTransform(resize, oldImage, OutputEncoding.PNG);
                Image newImage = imagesService.applyTransform(resize, oldImage, OutputEncoding.JPEG);
                byte[] newImageData = newImage.getImageData();
                out.write(newImageData);
                out.flush();

                                                      //画像名を取得
                Key imgkey = KeyFactory.createKey(imagedat.class.getSimpleName(), fileName);
                Blob blobImage = new Blob(newImageData);
```

```
                    imagedat img = new imagedat(imgkey,blobImage);
                                        PersistenceManager pm = PMF.get().getPersistenceManager();
                                                            try {
                                                                        pm.makePersistent(img);
                                                            } catch (Exception e) {
                                                                        e.printStackTrace();
                                                            } finally {
                                                                        pm.close();
                                                            }


                                                    }
                                        }
                                }
          } catch (FileUploadException e) {
                    e.printStackTrace();
          } finally {
                    if (out != null) {
                                out.close();
                    }
          }
    }

}
```

付録１２：Datastore に画像を保存するためのプログラム memoryFileItemFactory.java

```
/*******************************************************************************
 * Copyright (c) 2001, 2007 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
```

```
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *******************************************************************************/
package images;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemFactory;
public class memoryFileItemFactory implements FileItemFactory {
    public memoryFileItemFactory() {    }
    public FileItem createItem(String fieldName, String contentType, boolean isFormField,
            String fileName) {
        memoryFileItem result = new memoryFileItem(fieldName, contentType, isFormField, fileName);
        return result;
    }
}
```

付録１３：Datastore に画像を保存するためのプログラム memoryFileItem.java

```
/*******************************************************************************
 * Copyright (c) 2001, 2007 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *******************************************************************************/
package images;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.UnsupportedEncodingException;
import java.util.Map;
```

```
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileItemHeaders;
import org.apache.commons.fileupload.FileItemHeadersSupport;
import org.apache.commons.fileupload.ParameterParser;

public class memoryFileItem implements FileItem, FileItemHeadersSupport {
    //private static final long serialVersionUID = -2531086814081783645L;
    private static final long serialVersionUID = 1L;
    //public static final String DEFAULT_CHARSET = "ISO-8859-1";
    public static final String DEFAULT_CHARSET = "UTF-8";
    private String fieldName;
    private String contentType;
    private boolean isFormField;
    private String fileName;
    private long size = -1;
    private byte[] cachedContent = null;
    private FileItemHeaders headers = null;

    public memoryFileItem (String fieldName, String contentType, boolean isFormField, String fileName) {
        this.fieldName = fieldName;
        this.contentType = contentType;
        this.isFormField = isFormField;
        this.fileName = fileName;
    }

    public void delete() {
        cachedContent = null;
        baos = null;
    }

    public byte[] get() {
        if (isInMemory()) {
            if (cachedContent == null) {
                cachedContent = baos.toByteArray();
            }
            return cachedContent;
```

```
        }
        return new byte[0];
    }

    public String getContentType() {
        return contentType;
    }

    public String getFieldName() {
        return fieldName;
    }

    public InputStream getInputStream() throws IOException {
        if (cachedContent == null) {
            cachedContent = baos.toByteArray();
        }
        return new ByteArrayInputStream(cachedContent);
    }

    public String getName() {
        return fileName;
    }

    private ByteArrayOutputStream baos = null;

    public OutputStream getOutputStream() throws IOException {
        if (baos == null) {
            baos = new ByteArrayOutputStream();
        }
        return baos;
    }

    public long getSize() {
        if (size >= 0) {
            return size;
        } else if (cachedContent != null) {
```

```
            return cachedContent.length;
        } else if (baos != null) {
            return baos.toByteArray().length;
        }
        return 0;
    }

    public String getString() {
        byte[] rawdata = get();
        String charset = getCharSet();
        if (charset == null) {
            charset = DEFAULT_CHARSET;
        }
        try {
            return new String(rawdata, charset);
        } catch (UnsupportedEncodingException e) {
            return new String(rawdata);
        }
    }

    @SuppressWarnings("unchecked")
    public String getCharSet() {
        ParameterParser parser = new ParameterParser();
        parser.setLowerCaseNames(true);
        // Parameter parser can handle null input
        Map params = parser.parse(getContentType(), ';');
        return (String) params.get("charset");
    }

    public String getString(String encoding) throws UnsupportedEncodingException {
        return new String(get(), encoding);
    }
    public boolean isFormField() {
        return isFormField;
    }
    public boolean isInMemory() {
```

```
        return true;
    }
    public void setFieldName(String name) {
        this.fieldName = name;
    }
    public void setFormField(boolean state) {
        this.isFormField = state;
    }
    public void write(File file) throws Exception {
        // no process;
    }
    public FileItemHeaders getHeaders() {
        return headers;
    }
    public void setHeaders(FileItemHeaders headers) {
        this.headers = headers;
    }
}
```

付録１４：Datastore に画像を保存するためのプログラム PMF.java

```
/*******************************************************************************
 * Copyright (c) 2001, 2007 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *******************************************************************************/
package images;
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManagerFactory;
public final class PMF {
    private static final PersistenceManagerFactory pmfInstance =
        JDOHelper.getPersistenceManagerFactory("transactions-optional");
```

```
    private PMF() {}
    public static PersistenceManagerFactory get() {
        return pmfInstance;
    }
}
```

付録１５：端末にサーバ上の地図を表示する jsp ファイル mapview.jsp

```
<%@  page import="java.net.*"
 contentType="text/html; charset=utf-8"
 pageEncoding="utf-8"%>
<html>
          <head>
          <%
//        住所を受け取る
          String add = (String)request.getAttribute("msg");
          String msg2 = URLEncoder.encode(add, "utf-8");

          if ( msg2==null )
          {
                    msg2="";
          }
%>
                    <title>map view</title>
          </head>
          <body>
<%--                <h1><%=add%></h1>       --%>
          <img
src="http://maps.google.com/maps/api/staticmap?center=<%=msg2%>&zoom=13&size=640x640&markers=color:blue|<%
=msg2%>&sensor=false"
                     alt="not show map."
                     width="640"
                     height="640">

           </body>
</html>
```