Monaca と
ニフクラ mobile backend で学ぶ
オンラインデータ共有アプリ開発講座

富士通クラウドテクノロジーズ株式会社 著

アシアル株式会社 監修

目次

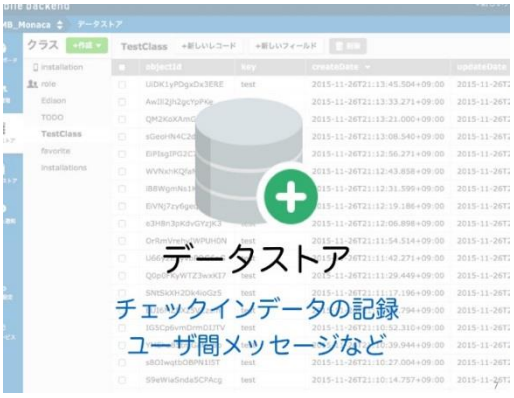
はじめに アプリ開発の強い味方 mBaaS

皆さんはスマートフォンアプリ開発に興味がありますか。アプリにはソーシャルネットワーク、メッセージ、地図、ゲーム、ニュースなど数多くのジャンルがあります。それらほとんどのアプリはネットワークを使っています。WiFi を使ってスマートフォンのデータをインターネット上にアップロードしたり、逆にダウンロードしたりています。そうしたデータの送受信を担う存在はサーバーと呼ばれています。
スマートフォンアプリを開発するにあたってサーバーは必須の存在と言えるでしょう。しかしアプリの開発だけでも大変なのに、サーバーまで用意しないといけないというのはとても厄介です。そこでアプリでよく使われる機能をまとめて提供する mBaaS が注目されています。
mBaaS は mobile Backend as a Service の略で、スマートフォンアプリ開発においてよく使われる機能を提供するサービスになります。最近ではスマートフォン以外の分野でも使われるので m を外して BaaS と呼ぶこともあります。Backend とはアプリの後ろ側（サーバー）を意味しています。つまり mBaaS は先ほど述べていたサーバーの代わりになります。例えばスマートフォンで撮影した写真を保存したり、友達にコメントやプッシュ通知を送ったりすることがサーバー不要で実現します。しかも mBaaS と呼ばれるサービスの多くは無料から利用できるようになっています。当社もニフクラ mobile backend という名称で mBaaS を提供しています。

ニフクラ mobile backend の主な機能について

ニフクラ mobile backend の機能から本チュートリアルで利用するデータストア・位置情報検索について説明します。

データストア － 文字、数字、位置情報を保存



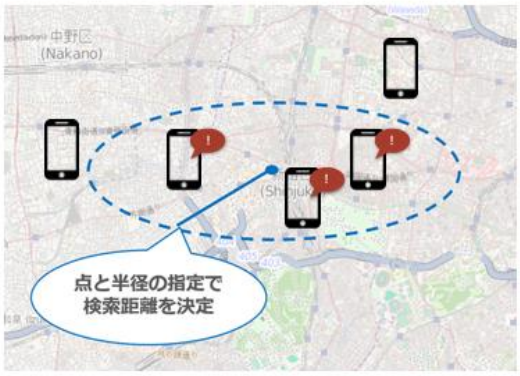
データベースとは、データを保存しておいて、プログラムから自由に取り出したり計算したりする技術のことです。ここで言うデータとは文字（テキスト）、数字、日付、位置情報などになります。こういったデータを集め、まとめて管理できるのがデータベースの特徴になります。

たとえば、電話帳アプリでは保存した友達の名前、電話番号、ニックネーム、メールアドレスなどが一覧で見られるでしょう。それがまさにデータベースです。名前で並び替えたり、検索したりといったことが簡単に実現します。電話帳アプリはデータベースがスマートフォンの中にありますが、他の人のスマートフォンとデータを共有するアプリを作るのであればサーバー上にデータを保存する必要があります。
そういった場面で使うのが、ニフクラ mobile backend の「データストア」です。サーバーにデータを保存することで、データを他の人と共有できるようになります。データの公開範囲については個別に設定できますので、あるデータはオープンに、あるデータは特定の友人やグループだけに公開といった指定もできます。
一般的なデータベースは利用前に構造の設計が必要ですが、データストアの場合は不要です。自由に構造の変更、指定ができるようになっています。

地図を使った検索に － 位置情報検索

「データストア」に含まれる便利な機能の一つに、「位置情報検索」があります。ある一点（例えばみなさんの教室など）を中心として半径何メートル以内にあるデータを検索するといった操作ができます。

スマートフォンには GPS が入っており、位置情報を簡単に利用できます。取得した位置情報をデータストアに保存しておけば、最近行った場所や、友人のいる場所を簡単に調べられるようになるでしょう。



ニフクラ mobile backend では一点を軸とした円を用いた検索の他、二点を指定した四角の中に含まれるデータを検索する仕組みを用意しています。

mobile backend 最初の一歩

ニフクラ mobile backend(以下 mobile backend)を使い始めるまでの手順を説明します。まず会員登録が必要ですが、無料で利用開始できますので安心してください。

mobile backend の利用登録

最初に mobile backend のサイトを表示します。URL は下記の通りです。

https://mbaas.nifcloud.com/

右上の「無料登録」をクリックします。次の画面で「SNS ID（クレジットカード払い）の登録」の「会員登録（無料）」をクリックします。（登録には Facebook, Twitter, Google のいずれかの SNS ID が必要です。お持ちでない方は事前に SNS ID の取得をお願いします。）

クリック後「新規アカウント登録」という画面が出ます。使用する SNS ID に該当するものを選んでクリックします。

各 SNS ID の画面に遷移するので、使用するアカウントを選択します。（使用する SNS ID により次の画面が異なります。）

再び mobile backend の画面に戻り、「メールアドレスの登録」が表示されます。メールアドレスを入力し、「確認メールを送信」をクリックします。（「確認メール送信完了」画面に遷移します。）

入力したメールアドレス宛にメールが届きます。届いたメールを開いて、記載された URL をブラウザで開きます。mobile backend の利用規約が表示されるので内容を確認し、「以上の規約に同意する」にチェックを入れ「アカウント登録」ボタンをクリックします。

これでアカウントの作成は完了です。初回のログインが完了し「アプリの新規作成」画面が表示されます。

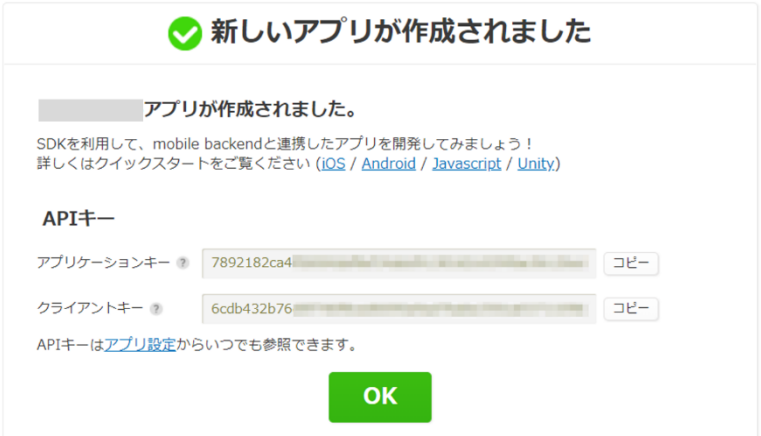
チュートリアルアプリの事前準備

mobile backend を使ったチュートリアルアプリとして「ブロック崩しアプリへのオンラインランキングの導入」と「あしあとマップ」を提供しています。これらのチュートリアルを始める前に下記の事前準備があるのでそちらを行いましょう。

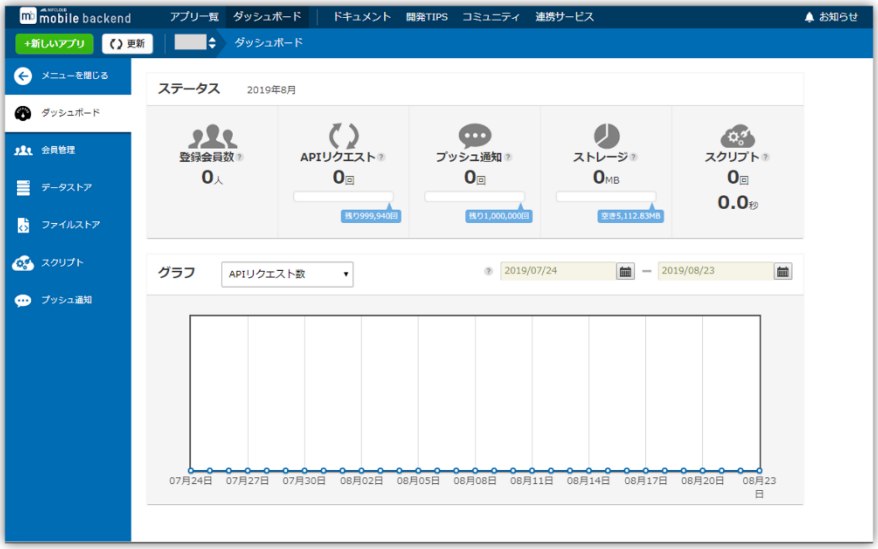
1. mobile backend 上にアプリを作る
2. Monaca にサンプルプロジェクトをインポートする

mobile backend 上にアプリを作る

アカウント作成完了（初回ログイン）後に表示される「アプリの新規作成」画面で、アプリ名のところにブロック崩しであれば「breakout_ranking」、あしあとアプリであれば「ashiato_map」と入力して新規作成をクリックします。
完了するとアプリケーションキーとクライアントキーという 2 つの API キーが発行され表示されます。この 2 つのキーはとても大事なものなので、安易に公開しないようにしてください。（API キーは後程使用します。）
「OK」をクリックして画面を閉じましょう。



アプリの管理画面が表示されます。ここで登録したデータやファイルを確認や、プッシュ通知の作成など行うことができます。



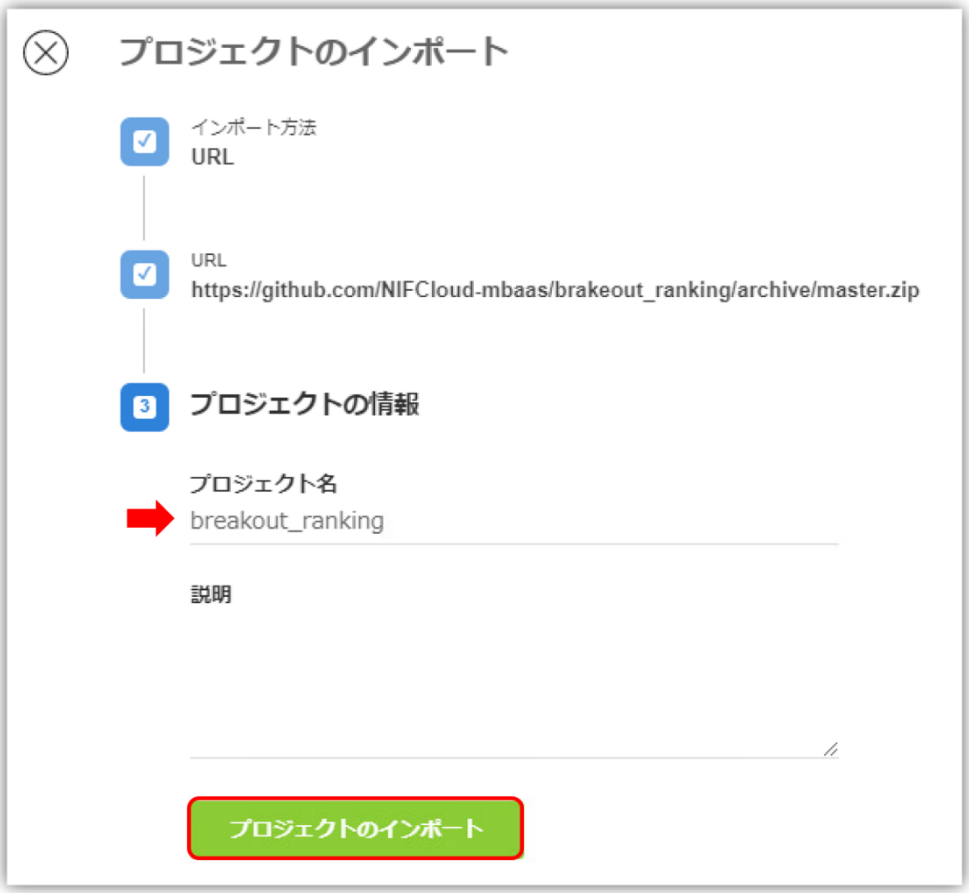
API キーは右上「アプリ設定」で確認できます。また、新しいアプリを作成する場合は左上「+新しいアプリ」をクリックしてください。

Monaca にサンプルプロジェクトをインポートする

次は Monaca のサンプルプロジェクトをインポートします。Monaca にログインし、「インポート」ボタンをクリックします。インポート方法は「URL」をクリックします。

URL には
ブロック崩しの場合「https://github.com/NIFCloud-mbaas/brakeout_ranking/archive/master.zip」
あしあとアプリの場合「https://github.com/NIFCloud-mbaas/ashiato_map/archive/master.zip」
を入力して「次」ボタンをクリックします。

プロジェクト名には
ブロック崩しの場合は「brakeout_ranking」、あしあとアプリの場合は「ashiato_map 」
と入力して、「プロジェクトのインポート」をクリックします。



以上で準備は完了です。次ページから各チュートリアルの解説に移ります。

ブロック崩しゲームにランキング機能を追加する

このチュートリアルでは mobile backed を使って Monaca のテンプレートとして公開されているブロック崩しゲームにスコアを保存し、ランキング一覧を表示する機能を追加します。
※本チュートリアルを行う前に「チュートリアルアプリの事前準備」を行ってください

概要

今回のコードでは、ベースになるブロック崩しゲームに以下の機能を追加しています。

- ゲーム終了時にスコアを保存しユーザ名を聞く
- mobile backed にユーザ名、スコアを保存する
- 保存したら順位をアラートで表示する
- ゲーム画面に RANKING という項目を追加
- RANKING をタップしたら mobile backed よりデータを抽出し、スコア順に一覧表示する

ゲームの流れ

ゲームの流れを上図の左から順に説明します。アプリを立ち上げるとすぐにゲームが開始できます。すべてのブロックを消すか、ボールが下に落ちるとゲームオーバーです。ゲームオーバーになるとユーザ名が聞かれるので入力します。入力したユーザ名はスマートフォン端末に保存されるので、最初の 1 回しか聞かれません。データを mobile backed に保存した後、アラートが出て自分の順位が表示されます。ゲームの画面下には RANKING というラベルがあります。これをタップするとランキングが最大 10 位まで表示されます。

実習

プログラムはあらかじめ穴埋め状態で記述されています。
このマークがついている枠内が皆さんの作業を行う箇所です。

ソースコードの追加・解説

今回のコードの殆どは「www/js/ncmbController.js」に入っています。このソースコードは全体として次のような構成になっています。

```
var ncmbController = {

  // 1. 変数の定義

  // 2. 初期化
  // 3. スコア送信
  // 4. ランキング表示

  // 5. ランキング画面を閉じる

}
```

1. 変数の定義

今回の変数は次の 3 つです。
・アプリケーションキー
・クライアントキー
・mobile backed のオブジェクト
それぞれ次のように定義します。「YOUR_APPLICATION_KEY」、「YOUR_CLIENT_KEY」をそれぞれ自身の mobile backed アプリのアプリケーションキー、クライアントキーに変えましょう。


```
var ncmbController = {
  APPLICATION_KEY: "YOUR_APPLICATION_KEY",
  CLIENT_KEY: "YOUR_CLIENT_KEY",

  ncmb: null,
  ：(省略)
};
```

2. 初期化

初期化処理では次のような処理を行います。
1. mobile backed オブジェクトの作成
2. イベントリスナ（閉じるボタンを押した時の動作を設定）
コードは以下のようになっています。

```
：(省略)
```

```
// 初期化
init: function(screenSize) {
  var self = this;
  self.ncmb = new NCMB(self.APPLICATION_KEY, self.CLIENT_KEY); // mobile backend の初期化

  // 閉じるボタンの動作を設定

  document.getElementById("closeRanking").addEventListener("click", function () {
    self.closeRanking();
  });
},
  ：(省略)
```

3. スコア送信

スコアを送信する処理の流れは次のようになります。

1. スコア（Score）クラスの生成
2. ユーザ名の設定
3. スコアのインスタンスを生成
4. 送信処理実行
5. 保存が成功した場合アラートを表示

スコア送信のコード全体は次のようになります。
「**//スコア(Score)クラスの生成**」の下から実装してきましょう。実装が終わったら一度、Monaca デバッガーで正常に動作するかどうかをご確認ください。


```
// スコア送信
sendScore: function(score) {
    // スコア（Score）クラスの生成
    var Score = this.ncmb.DataStore("ScoreClass");

    // ユーザ名の設定

    var username = localStorage.getItem("username");
    if (username === null || username === "") {

      username = prompt("ユーザ名を指定してください");

      localStorage.setItem("username", username);
    }

    // スコアのインスタンスを生成
    var scoreData = new Score({score: score, username: username});

    // 送信処理実行
    scoreData.save()
```

```
    .then(function (saved) {

        // 保存がうまくいったらアラート表示

        alert("保存しました。もう一度挑戦してください！");

    })
    .catch(function(err){
        console.log(err);
    });
},
```

データストアでは、「クラス」というのが一つの単位になります。アプリで利用するデータの種類ごとにクラスを用意します。今回のアプリはゲームのスコアを管理する機能だけを持つので、作成するクラスはスコアクラス一つだけとなります。もしゲームのキャラクターやアイテム、進行状況などの様々な種類のデータを管理したい場合は、その種類に応じてクラスの数も増やす必要があります。

それでは処理内容について解説します。
データストアに対してクラス名を指定すると、スコア（Score）クラスが生成されます。このクラス名は自由に指定できます（英数字のみ）。

```
var Score = this.ncmb.DataStore("ScoreClass");
```

「//　スコアのインスタンスを生成」というコメントがついている箇所では、データストアに保存する１件分のデータを生成する処理を行っています。この時点ではまだデータストアへの保存はされていない状態です。score や username というデータのキーは自由に決められます。

```
// スコアのインスタンスを生成
var scoreData = new Score({score: score, username: username});
```

そして最後に保存処理を実行して、mobile backend へデータを保存します。これによりデータがサーバー上に残るようになります。mobile backend ではデータの保存を行った際、エラーがなければ then()という処理が呼び出されます。エラーが起きたら catch()になります。

```
scoreData.save()
  .then(function (saved) {

    // 保存がうまくいったらアラート表示

    alert("保存しました。もう一度挑戦してください！");
```

```
  })
  .catch(function(err){
    console.log(err);
  });
```

4. ランキング表示

スコアを表示する処理は、「www/js/main.js」 の reset メソッドの中に記述されています。
処理内容は次のようになります。

1. HTML 上に「RANKING」テキストを表示し、タップされたら mobile backend からスコアデータを取得
2. スコアデータを HTML に整形
3. ランキングデータを表示

全体のコードは次のようになります。

```
：(省略)
reset: function() {
  ：(省略)
  rankingLabel.click = rankingLabel.tap = function(data) {
      ncmbController.showRanking();
  };
  setTimeout(function() {
      rankingLabel.setText("RANKING"); //for Android
  }, 1000, rankingLabel);
  ：(省略)
}
```

RANKING という文字をタップしたときのイベントとして、「ncmbController.showRanking();」を実行しています。
「ncmbController.showRanking();」は「www/js/ncmbController.js」の中に記述されています。

```
：(省略)
  showRanking: function() {

    //スコア情報を取得するため、クラスを作成

    var Score = this.ncmb.DataStore("ScoreClass");

    //スコアを降順に 10 件取得

    Score.order("score", true)
      .limit(10)
      .fetchAll()
      .then(function(results){
```

```
        //ランキング表の HTML 生成

        var tableSource = "";
          if(results.length > 0){
            for(i=0; i<results.length; i++){
              var score = results[i],
                rank = i + 1,
                value = parseInt(score.score),
                displayName = "NO NAME";
                tableSource += "<li class=¥"list__item list__item--inset¥">"
                        + rank + ":"
                        + score.username
                        + " (" + value + ")</li>";
                }
          } else {
            tableSource +=
            "<li class=¥"list__item list__item--inset¥">ランキングはありません</li>";
          }
            document.getElementById("rankingTable").innerHTML = tableSource;

            //ランキング画面を表示する

            document.getElementById("ranking").style.display = 'block';
          })
        .catch(function(err){
          console.log(err);
        });
：省略)
```

まず先ほどのスコア保存と同じく Score クラスを定義しています。

```
var Score = this.ncmb.DataStore("ScoreClass");
```

次に検索条件を指定しています。今回はスコアの降順に、最大 10 件としています。order が並び順（後ろにある true は降順の指定です）、limit が取得件数を意味しています。fetchAll は対象をすべて取得するという意味です。他に count（カウント。データの件数だけを返す）もあります。

```
Score.order("score", true)
  .limit(10)
  .fetchAll()
```

検索が実行され、エラーがなかった場合は保存と同じく then が呼ばれます。エラーがあった場合は catch になります。results 変数の中に検索結果のデータが配列で入ってきます。

```
Score.order("score", true)
  .limit(10)
  .fetchAll()
  .then(function(results){

    // 検索がうまくいった場合

  })
  .catch(function(err) {

    // 検索がうまくいかなかった場合

  })
```

検索がうまくいった場合はランキング表の作成に入ります。tableSource という変数の中にランキング一覧を表示する HTML 文字列を入れています。順位と名前、そしてスコアを順番に並べています。

```
var tableSource = "";
if(results.length > 0){
  for(i=0; i<results.length; i++){
    var score = results[i],
    rank = i + 1,
    value = parseInt(score.score);
    tableSource += "<li class=¥"list__item list__item--inset¥">"
           + rank + ":"
           + score.username
           + " (" + value + ")</li>";
    }
} else {
  tableSource += "<li class=¥"list__item list__item--inset¥">ランキングはありません</li>";
}
document.getElementById("rankingTable").innerHTML = tableSource;
document.getElementById("ranking").style.display = 'block';
```

ランキング一覧は「www/index.html」の中の、rankingTable という ID がついた<ul>要素内に表示しています。

```
<body>
```

```
  <div id="ranking" class="ranking">
  <h2>ランキング</h2>
    <ul id="rankingTable" class="list list--inset rankingTable">
    </ul>

  <ons-button class="login-button" id="closeRanking">閉じる</ons-button>

  </div>
</body>
```

これでランキングの表示が完了になります。

5. ランキング画面を閉じる

では最後にランキング画面を閉じる処理です。これは先ほど表示したランキング一覧を非表示にするだけです。

```
//ランキング画面を閉じる

var ncmbController = {
：(省略)
  closeRanking:function() {
    document.getElementById("ranking").style.display = 'none';
  }
};
```

地図にマーカーを立てるあしあとアプリを作ろう

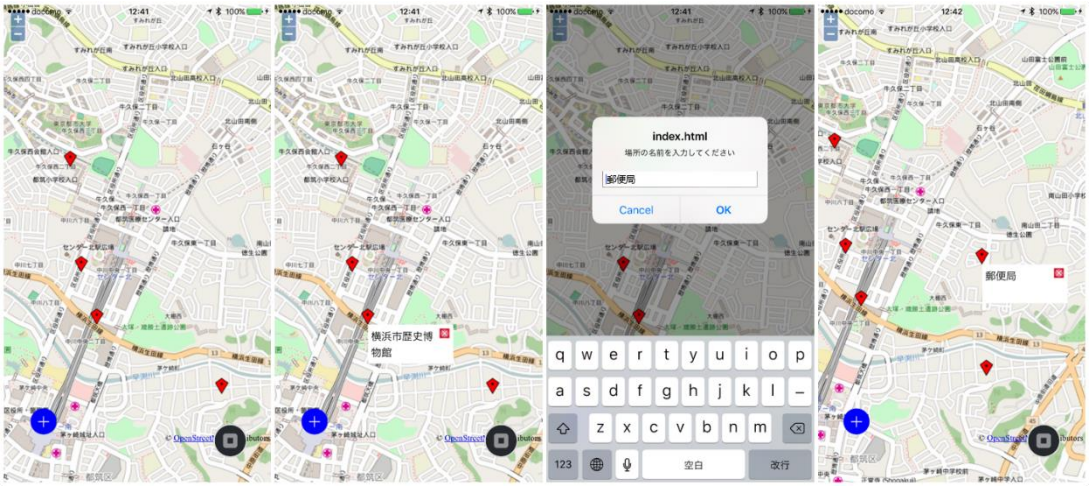
このチュートリアルではスマートフォンから現在位置を取得して地図上にマーカーを立てたり、付近のマーカーを検索できたりするアプリを作ります。マーカーの位置情報は mobile backend 上に保存します。
※本チュートリアルを行う前に「チュートリアルアプリの事前準備」を行ってください。

概要

今回のコードではベースになるアプリに以下の機能を追加していきます。

- 地図ライブラリの導入
- 現在位置の取得と地図の表示
- 現在位置をマークする
- 付近にあるマークを検索する

アプリの流れ



アプリの流れについて上図の左から順に説明します。
アプリを開くと付近にあるマーカーが表示されます。マーカーをタップすると、説明が表示されます。左下にある「＋ボタン」をタップすると、現在地の場所の名前を入力するダイアログが表示され、「OK」をタップすると追加したマーカーが表示されます。

実習

プログラムはあらかじめ穴埋め状態で記述されています。
このマークがついている枠内が皆さんの作業を行う箇所です。

ソースコードの追加・解説

今回のコードの殆どは「www/js/ncmbController.js」に入っています。このソースコードは全体として次のような構成になっています。

```
// 1. 変数の定義

var ncmbController = {
  // 2. 初期化/初期表示処理

  // 3. 付近のマーカーを検索

  // 4. マーカーを表示する処理

  // 5. 場所を保存する「＋ボタン」を表示

  // 6. マーカーを追加する処理

};
```

1. 変数の定義

今回利用する変数は次の通りです。

```
- application_key: アプリケーションキー

- client_key: クライアントキー
- ncmb: mBaaS のオブジェクト

- Places: データストア（Place）のオブジェクト

- map: OpenLayers（汎用的な地図ライブラリ）オブジェクト

- mapnik: OpenStreetMap 用のレイヤー

- projection3857: 座標系（EPSG:3857）

- projection4326: 座標系（EPSG:4326）

- popup: 地図のバルーン表示用
```

これらを最初に定義します。「YOUR_APPLICATION_KEY」、「YOUR_CLIENT_KEY」をそれぞれ自身の mobile backend アプリのアプリケーションキー、クライアントキーに変えましょう。

```
var application_key = "YOUR_APPLICATION_KEY";
var client_key = "YOUR_CLIENT_KEY";
var ncmb = new NCMB(application_key, client_key);
var Places = ncmb.DataStore("Places");

var map = new OpenLayers.Map("map");
var mapnik = new OpenLayers.Layer.OSM();
map.addLayer(mapnik);

var projection3857 = new OpenLayers.Projection("EPSG:3857");
var projection4326 = new OpenLayers.Projection("EPSG:4326");
var popup = null;
```

2. 初期化/初期表示処理

初期化、初期表示は次のように処理を行います。
1. 現在位置の取得
2. 現在位置を地図の中央に設定
3. すでに保存してあるマーカーを検索/地図上に配置
4. マーカーを追加するボタンを表示

コードは以下のようになります。 **3. すでに保存してあるマーカーを検索/地図上に配置、 4. マーカーを追加するボタンを表示** についてはこの後作成します。

```
var ncmbController = {
  run: function() {
  var self = this;

  // 現在位置を取得します

  navigator.geolocation.getCurrentPosition(function(location) {

  // 現在位置を取得すると、location という変数の位置情報オブジェクトが入ります

  // 位置情報を使って、OpenLayers の位置情報オブジェクトに変換します

  // その際、EPSG:4326 から EPSG:3857 に変換する指定を行います

  var lonLat = new OpenLayers.LonLat(location.coords.longitude, location.coords.latitude)
                            .transform(
                                      projection4326,
                                      projection3857
                                     );

  // 作成した位置情報を地図の中央に設定します

  map.setCenter(lonLat, 15);

  // マーカーを検索する処理です

  ncmbController.findMarkers(location.coords.latitude, location.coords.longitude);

  // ボタンを追加する処理です

  ncmbController.addButton();
}
```

3. 付近のマーカーを検索

マーカーを検索する処理にあたる「ncmbController.findMarkers」を実際に作ってみます。この処理は mobile backend の位置情報検索機能を使っています。引数は緯度、経度になります。

「**// mobile backend の位置情報オブジェクトを作成**」の下から、実際に自分で書いてみてください。


```
var ncmbController = {

  : （省略）

  findMarkers: function(latitude, longitude) {

  // mobile backend の位置情報オブジェクトを作成

  var current_position = new ncmb.GeoPoint(latitude, longitude);
  Places
    .withinKilometers("point", current_position, 1000)
    .limit(20)
    .fetchAll()
    .then(function(places) {
      for (var i = 0; i < places.length; i++) {
       var place = places[i];
       ncmbController.addMarker(place.areaName,
                                place.point.latitude, place.point.longitude);
      }
    })
    .catch(function(err) {

      alert("エラーが発生しました");

    });
},
```

引数で渡された緯度、経度を使って mobile backend の位置情報オブジェクトを作ります。これは検索に用いるものです。

```
var current_position = new ncmb.GeoPoint(latitude, longitude);
```

Places は「**1. 変数の定義**」で定義した mobile backend のデータストアオブジェクトです。次にある「withinKilometers」は距離（半径）を使った検索条件を使う場合の指定で、「"point"」というカラム（列）を検索します。その際の値は「current_position」、つまり現在いる場所になります。最後の「1000」は 1,000km を意味します。次の行にある「.limit(20)」はデータを最大 20 個取得するという指定になります。つまり、現在置を中心として、半径 1,000km 以内にあるデータを最大 20 件取得するという指定を行っています。最後

にある「fetchAll()」は検索処理を実行する命令になります。

```
Places
  .withinKilometers("point", current_position, 1000)
  .limit(20)
  .fetchAll()
```

検索が成功すると「.then」というメソッドが呼ばれます。その際には取得したマーカー情報が返ってきます。
失敗した場合には「.catch」が呼ばれ、エラー原因が返ってきます。

```
.then(function(places) {

  // データ取得成功時

  for (var i = 0; i < places.length; i++) {
    var place = places[i];
    ncmbController.addMarker(place.areaName, place.point.latitude, place.point.longitude);
  }
})
.catch(function(err) {

  // データ取得失敗時


  alert("エラーが発生しました");

});
```

「.then」の中では取得したデータを一つずつ、「ncmbController.addMarker」に渡しています。この処理は地図上にマーカーを表示する処理になります。
引数として、エリア名/位置情報を渡しています。「place.areaName」と「place.point」は「**6. マーカーを追加する処理**」で登録を行っているエリア名と位置情報です。位置情報から緯度と経度と抽出する場合は「place.point」の後ろに latitude（緯度）と longitude（経度）をそれぞれ追記します。

4. マーカーを表示する処理

マーカーの表示には地図表示などで使われるオープンソースライブラリであ「OpenLayers」を使っています。
引数として渡されたエリア名、緯度、経度の情報を元に、マーカーを作成して地図上に表示しています。また、マーカーをタップしたときにポップアップを表示する処理も設定しています。

```
addMarker: function(areaName, latitude, longitude) {
```

```
    // マーカーを表示するためのレイヤーを準備
    var markers = new OpenLayers.Layer.Markers("Markers");
    map.addLayer(markers);

    // マーカーを作成
    var marker = new OpenLayers.Marker(
      new OpenLayers.LonLat(longitude, latitude)
      .transform(
        projection4326,
        projection3857
      )
    );
    // マーカーのタグとしてエリア名を指定
    marker.tag = areaName;

    // マーカーをタップした際にポップアップを表示します
    marker.events.register("touchstart", marker, function(event) {
      // すでに別なポップアップが開いていたら消します
      if (popup) map.removePopup(popup);
      // ポップアップを作成
      popup = new OpenLayers.Popup("chicken",
           event.object.lonlat,
           new OpenLayers.Size(100,50),
           event.object.tag,
           true);
      // 作成したポップアップを地図に追加します
      map.addPopup(popup);
    });

    // 作成したマーカーを地図(マーカーレイヤー)に追加します
    markers.addMarker(marker);
},
```

5. 場所を保存する「＋ボタン」を表示

「＋ボタン」を押した時に **6. マーカーを追加する処理** にあたる「ncmbController.createPlace」が実行さ

れるように指定しています。

```
addButton: function() {
  var custom_button = new OpenLayers.Control.Button({
    displayClass : 'olControlCustomButton',
    trigger : ncmbController.createPlace
  })
  var control_panel = new OpenLayers.Control.Panel({});
  control_panel.addControls([custom_button])
  map.addControl(control_panel);
},
```

6. マーカーを追加する処理

最後にマーカーを追加する処理にあたる 「ncmbController.createPlace」を実装したいと思います。この処理は次のような流れになっています。

1. エリア名の入力を促す
2. 現在の位置情報を取得
3. データの作成
4. mobile backend への保存を実行

全体のコードは次のようになります。
では「**// エリア名の入力を促す**」の下から、実際に自分で書いてみてください。


```
createPlace: function() {

  // エリア名の入力を促す

  var areaName = prompt("場所の名前を入力してください");

  navigator.geolocation.getCurrentPosition(function(location) {
    var geoPoint = new ncmb.GeoPoint(location.coords.latitude, location.coords.longitude);
    var place = new Places();
    place.set("areaName", areaName);
    place.set("point", geoPoint);
    place.save()
      .then(function(point) {
        ncmbController.addMarker(point);
      })
      .catch(function(err) {

        alert("エラーが発生しました。再度行ってください");

      });
```

```
  });
}
```

prompt()でテキストボックスつきのダイアログを表示して、ユーザーにエリア名（お店や建物などの名前）の入力を促しています。

```
var place = prompt("場所の名前を入力してください");
```

次に初期化処理で行ったように現在の位置情報を取得します。取得が成功したら location 変数の中に位置情報が入ります。その位置情報を使ってマーカーを立てたのと同じように mobile backend の位置情報オブジェクトを作成します。引数は緯度、経度になります。

```
navigator.geolocation.getCurrentPosition(function(location) {
  // 現在位置の取得成功
  :（省略）
}
```

```
var geoPoint = new ncmb.GeoPoint(location.coords.latitude, location.coords.longitude);
```

位置情報オブジェクトを作ったら、それを保存するために Places オブジェクトのインスタンスを作ります。これはデータストアに保存する一つのデータと考えてください。データはカラム名（任意）と値のセットで指定します。

```
var place = new Places();
place.set("areaName", areaName);
place.set("point", geoPoint);
```

そして最後に「save」メソッドを実行します。このメソッドを実行すると、データが mobile backend に保存されます。このメソッドもデータを検索した時と同じく、保存が成功すると「then」メソッドが、失敗すると「catch」メソッドが呼ばれます。「then」メソッドにはデータストアに保存された位置情報が返ってきますので、「**4. マーカーを表示する処理** 」の引数として渡しています。

```
place.save()
```

```
    .then(function(point) {
      ncmbController.addMarker(point);
    })
    .catch(function(err) {

      alert("エラーが発生しました。再度行ってください")

    });
```