1. ニフティクラウド mobile backendとMonacaにユーザ登録しよう

ここではまず最初にニフティクラウド mobile backendへのユーザ登録と、Monacaへのユーザ登録を行います。どちらも無料です。

ニフティクラウド mobile backendにユーザ登録する

まずニフティクラウド mobile backendのサイトに行きます。右上に無料登録のリンクがあるのでクリックします。

@nifty会員登録をする

次に最初に @nifty会員に登録します。すでにアカウントを持っている場合は、その下にあるニフティクラウド mobile backendログインをクリックしてください。

@nifty会員登録画面ではユーザ名、パスワード、メールアドレスを入力します。入力が終わったら登録ボタンを押してください。

確認画面が出ますので、内容が問題なければ **上記の内容で【登録する】** をクリックします。

登録が完了したら、ログインをクリックします。

ニフティクラウド mobile backendにログインする

ログイン画面では先ほど登録したユーザ名、パスワードを入力してください。入力が終わったらログインボタンをクリックします。

最初のログイン時には利用規約が表示されます。内容を確認の上、問題なければ **以上の規約に同意する** のチェックボックスをつけて、アカウント登録をクリックします。

アプリを新規登録する

ニフティクラウド mobile backendに登録が完了すると、最初にアプリを作成します。これはニフティクラウド mobile backend上のデータを管理する単位となります。英数字で適当なものを設定してください。例として **FavoriteReader** としておきます。入力が終わったら、新規作成ボタンをクリックします。

これでアプリの作成が終わりました。**ここに出ているアプリケーションキーとクライアントキーを後々使いますので、この画面はこのままにしておいてください。**

Monacaにアカウント登録する

続いて新しいタブを開いてMonacaにアカウント登録します。まずMonacaのサイトに行って、右上にあるサインアップをクリックします。

アカウント登録はメールアドレスとパスワードで行います。入力したら **いますぐ登録** ボタンをクリックします。

そうすると確認メールが送信されます。

自分のメールボックスを調べてください。メールの中に確認用のURLがありますので、それをクリックします。そうすると以下のようにアカウント登録が完了します。

アカウント登録が完了するとダッシュボードが表示されます。

ここまでで一旦完了です。続いて2. Monaca x ニフティクラウド mobile backendを体験するに進んでください。

2. Monaca × ニフティクラウド mobile backendを体験する

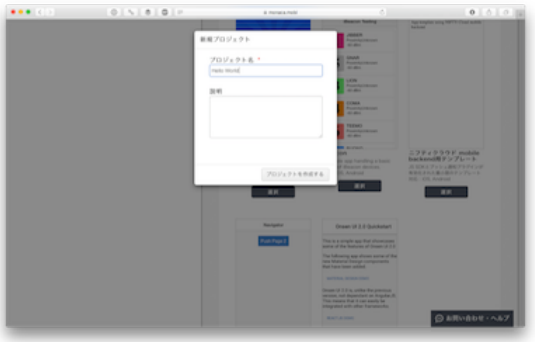
今回はMonacaアプリを通じてニフティクラウド mobile backendに触れてみたいと思います。

ベースになるアプリを作成する

まずMonacaのダッシュボードを表示して、**新規プロジェクト**をクリックします。

右側にプロジェクトテンプレートが表示されますので、下の方にある **ニフティクラウド mobile backend用テンプレート** を選択します。

プロジェクト名は自由ですが、今回は **Hello World** としておきます。入力したら**プロジェクトを作成する**をクリックしてください。

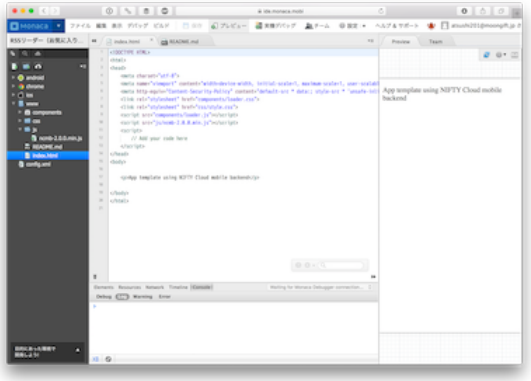


プロジェクトの一覧（左側）が更新されて、作成したプロジェクト名が出てきたら、そのプロジェクトの**開く**をクリックします。

このような画面になればOKです。

コードを書く

ではコードを書いてみましょう。左にあるファイル一覧から、 `www/index.html` をダブルクリックします。



そして、 `<script>〜</script>` の中に次のように記述しましょう。

元：（www/index.htmlの中から下記記述があるところを探してください）

```
<script>
  // Add your code here
</script>
```

更新後：

```
<script>
    // Add your code here
    function onDeviceReady() {
        alert("読み込まれました");
    }
    var event = typeof cordova === 'undefined' ? 'DOMContentLoaded' : 'dev
   document.addEventListener(event, onDeviceReady, false);
</script>
```

これを保存します。保存は上にある保存アイコンをクリックするか、Ctrl+S（Windows）またはコマンド+S（Mac OSX）を押します。そうするとアラートが表示されるはずです。

このコードはアプリの初期ロードが終わったタイミングで `onDeviceReady` 関数を実行します。 `typeof cordova === 'undefined'` という処理で、cordovaという変数が定義されている場合はMonaca上で実行されている場合、そうでない場合はPC上で実行されている場合と判定しています。

Monacaアプリの場合、イベントは `deviceready` になります。PC（Webブラウザ）の場合は `DOMContentLoaded` を使っています。 `deviceready` だけを定義するとPC上で試せないので注意してください。

ニフティクラウド mobile backendを試す

続いてニフティクラウド mobile backendを試してみたいと思います。 `onDeviceReady` の中身を次のように変えます。

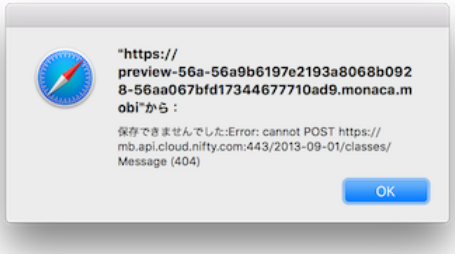
元：先ほど記述したコードです

```
function onDeviceReady() {
    alert("読み込まれました");
}
```

更新後：

```
function onDeviceReady() {
    var application_key = "APPLICATION_KEY";
    var client_key = "CLIENT_KEY";

    // ニフティクラウド mobile backendを初期化します
    var ncmb = new NCMB(application_key, client_key);

    // データストアのMessageクラス作成
    var Message = ncmb.DataStore("Message");

    // Messageクラスのインスタンス作成
    var message = new Message();
    // textカラムにHello Worldを指定
    message.set("text", "Hello World");

    // 保存処理を実行します
    message.save().then(function (obj) {
        // 保存処理成功
        alert("保存しました");
    }).catch(function (err) {
        // 保存処理失敗
        alert("保存できませんでした:"+err);
    });
}
```

これを保存すると、次のようにエラーが出ます。

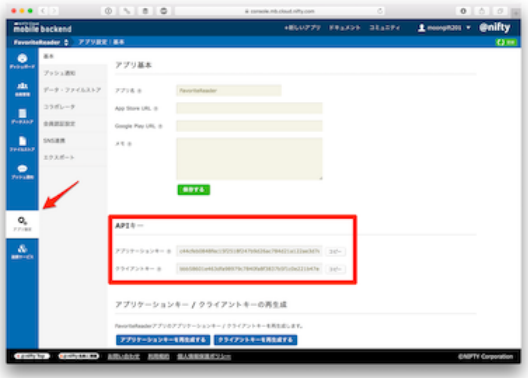


これはアプリケーションキーとクライアントキーが設定されていないからです。先ほど登録したニフティクラウド mobile backendの画面にあったアプリケーションキーとクライアントキーをそれぞれ設定してください。

```
var application_key = "APPLICATION_KEY"; // ←これを書き換えます
var client_key = "CLIENT_KEY"; // ←これを書き換えます
```

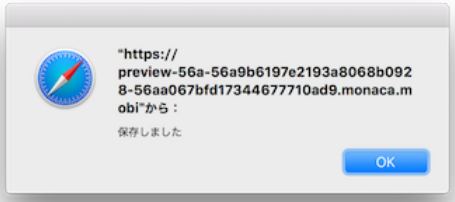
アプリケーションキーとクライアントキーは先ほどの画面で確認することができます。

もしこの画面を閉じてしまっている場合には、ニフティクラウド mobile backendの管理画面に入り、左側にあるアプリ設定メニューをクリックすると画面中央ほどにAPIキーとして確認できます。



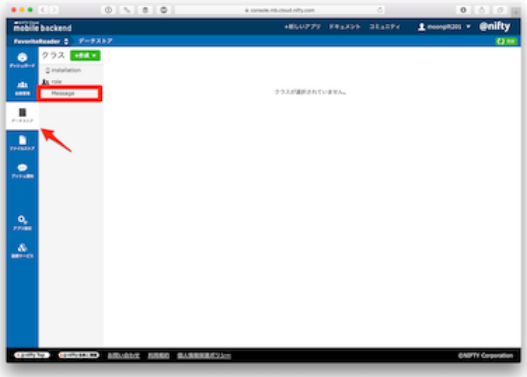
実行する

アプリケーションキーとクライアントキー、両方とも書き換えて保存すると、次のように保存成功のメッセージが確認できるはずです。



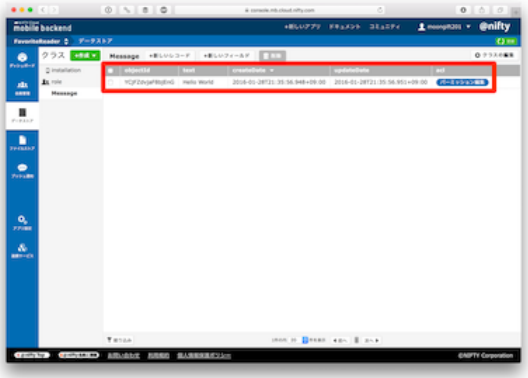
ニフティクラウド mobile backendの管理画面で確認する

では本当に保存されているか、ニフティクラウド mobile backendの管理画面で確認してみましょう。左側にあるデータストアをクリックします。



クラスの一覧にMessageというのがあるのが確認できるはずです。これをクリックします。

そうすると右側にテーブルが表示されて、textカラムがHello Worldになっているデータが確認できるかと思います。



コードの解説

先ほどのコードは次のようになっていました。

ニフティクラウド mobile backendを使う際の初期設定です。アプリケーションキーとクライアントキー、2つを停止します。

```
var ncmb = new NCMB(application_key, client_key);
```

次にデータストアのクラスを用意します。文字列として**Message** としています。これは先ほど管理画面のクラス一覧にあったMessageと同じです。このクラス名は自由に指定できるようになっていますので、**Hello**としたり、**Test** などとしても大丈夫です。変更して保存後、データストアにどのように保存されるか確かめてみてください。

```
var Message = ncmb.DataStore("Message");
```

次の行ではインスタンスを作成しています。先ほどのMessageはクラスであり、データベースで言うところのテーブルに相当します。インスタンスを作成するということはデータベースで言うところの一つ一つのデータ（行）に相当します。

```
var message = new Message();
```

次の行では作成したインスタンスにsetメソッドを呼んでいます。これは行に新しい値を設定していて、引数の一つ目がカラム、二つ目が値になります。今回はtextというカラムに対して Hello Worldという値を指定しています。カラム名は自由に指定することができ

ます。値は文字列、数値、日付、位置情報、オブジェクトなどが指定できます（今回の例は文字列です）。

```
message.set("text", "Hello World");
```

そして保存を実行します。この処理でデータがニフティクラウド mobile backendに送られます。これはsaveメソッドを使います。

```
message.save().then(function (obj) {
    // 保存処理成功
    alert("保存しました");
}).catch(function (err) {
    // 保存処理失敗
    alert("保存できませんでした:"+err);
});
```

この処理は非同期で処理されるので、Promiseを使っています。そして保存がうまくいった場合は、thenメソッドが呼ばれます。エラーだった場合はcatchメソッドです。

thenメソッドでは保存したデータオブジェクトが返ってきます。catchにはエラーメッセージが返ってきます。

Promiseについては詳しくは解説しませんが、オンライン上に情報がたくさんありますので調べてみてください。

Monacaデバッガーで確認してみる

最後にMonacaが提供しているMonacaデバッガーを使って確認してみたいと思います。MonacaデバッガーはiOS、Android向けに提供されているアプリで、App StoreやGoogle Playからダウンロードできます。それぞれアプリストアで **Monaca** で検索してください。

Monacaデバッガーをダウンロードしたら、そのまま起動してください。最初にログインが表示されますので、Monacaに登録したユーザIDとパスワードでログインしてください。

ログインするとプロジェクト一覧が表示されます。今回作成したプロジェクトが確認できるはずなので、それをタップしてください。

このようにアプリが開いて、保存しましたというアラートが出ればOKです。ニフティクラウド mobile backendの管理画面を見ると、データが追加されているのが分かるでしょう。

ここまでで一旦完了です。続いて RSSリーダー（お気に入り登録機能付き）のベースを取り込むに進んでください。

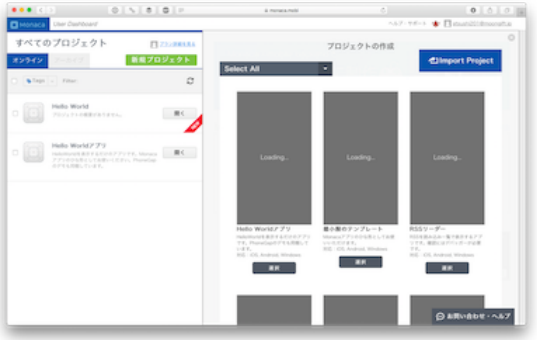
3. RSSリーダー（お気に入り登録機能付き）のベースを取り込む

2. Monaca × ニフティクラウド mobile backendを体験するに続いて、今回からRSSリーダーを作っていきたいと思います。今回はまず、Monacaで提供されているRSSリーダーアプリをカスタマイズしたプロジェクトを取り込むところからはじめていきます。

プロジェクトをインポートする

先ほどと同様に、新規プロジェクトをクリックします。このメニューは左上のMonacaと書かれたロゴの横の三角形をクリックすると出ます。

プロジェクトの一覧が出ましたら、 Import Project をクリックします。



フローティングが出たら、プロジェクト名を「RSSリーダー（お気に入り登録機能付き）」とします。そしてインポート方法として、URLを指定してインポートを選択します。値は下記を入力して

ください。

```
https://raw.githubusercontent.com/NIFTYCloud-mbaas/CW_Monaca_NCMB_Handson/master/project.zip
```

終わったらインポートボタンを押します。

インポートが終わるとプロジェクト一覧が更新されますので開くをクリックします。

このような表示になればOKです。

主な変更点について

まずベースとなるMonacaプロジェクトで提供されているRSSリーダーとの違いを説明します。以下の修正を自分で行うことで、素のMonacaプロジェクト（RSSリーダー）から同じようにすることもできます。

index.html

スタイルシート追加（12行目）

`css/favorite.css` の追加と、Font awesomeのCDN版を追加してあります。

```
<link rel="stylesheet" href="css/favorite.css">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.4.0/css/font-awesome.min.css">
```

JavaScriptファイルの読み込み（10行目）

後述する `favorite-online.js` を追加してあります。

```
<script src="js/favorite-online.js"></script>
```

フィードのURLを変更（16行目）

フィードのURLを変更してあります。デスクトップの場合、外部ドメインのXMLファイルは取得できませんので、ローカルにXMLファイルを置いて、それを読み込むようにしています。アプリか否かの判定は前に行った window.cordova があるかどうかを使っています。

```
var feedUrl = typeof window.cordova == "undefined" ? "./rss.xml" : "http://blog.mb.cloud.nifty.com/?feed=rss2";
```

ニフティクラウド mobile backendのアプリケーションキーとクライアント

キーを用意（19行目）

後で書き換えてもらうため、アプリケーションキーとクライアントキーの変数を用意してあります。

```
var application_key = "YOUR_APPLICATION_KEY";
var client_key      = "YOUR_CLIENT_KEY";
```

お気に入りクラスの生成（24行目）

RSSリーダーに追加するお気に入り登録機能を管理するFavoriteクラスを生成しています。実際のコードは後述します。引数としてアプリケーションキーとクライアントキーを渡しています。

```
// お気に入りクラスを定義します
// 内容は www/js/favorite-online.js になります
var favorite = new Favorite({
    applicationKey: application_key,
    clientKey: client_key
});
```

FavoriteクラスをFeedクラスに取り込み

RSSフィードを処理するFeedクラスの中に生成したFavoriteオブジェクトを登録しています。

元：

```
var feed = new Feed({
    url: feedUrl
});
```

変更後：

```
var feed = new Feed({
    url: feedUrl,      // カンマの追加を忘れずに
    favorite: favorite // 追加
});
```

スタイルシート css/style.css の修正

デフォルトのスタイルシート、css/style.css を修正しています。修正点は以下の2カ所です。1つ目は、

```
.feed-item {
    position: relative; /* 追加 */
    border-bottom: 1px solid #ccc;
    line-height: 1.2;
    padding: 6px;
}
```

2つ目は、

```
.feed-item h2 {
    font-size: 115%;
    color: #313732;
    font-weight: bold;
    /* margin: 2px 0 5px; */ /* 元 */
    margin: 2px 70px 5px 0;  /* 変更後 */
}
```

js/feed-reader.js の修正

Favoriteオブジェクトを登録する変数を初期化（10行目）

オブジェクト変数を扱うためのプロパティを初期化します。

```
// お気に入りオブジェクトの初期化
this.favorite = null;
```

フィードを読み込んだ後のお気に入り状態を反映（54行目）

フィードを読み込んだ後にお気に入りの状態を反映するためのメソッドを追加します。実際に内容は後述します。

```
self.favorite.applyAll();  // 追加
```

js/favorite-online.js の追加

Favoriteクラスの内容が書かれています。このファイルは新しく作っています。大きく分けて、次のような処理を行っています。

1. 初期設定
2. 全ての記事に対してお気に入り状況を反映させる
3. 一つの記事に対してお気に入り状態を反映させる
4. イベント処理を設定
5. お気に入り登録処理
6. お気に入り解除処理
7. アプリ+端末を特定するためのuuidを取得

コードは次のようになります。

```
var Favorite = (function() {
    // 初期設定
    var Favorite = function(options) {
    }

    // 全ての記事に対してお気に入り状況を反映させる
    Favorite.prototype.applyAll = function() {
    };

    // 一つの記事に対してお気に入り状態を反映させる
    Favorite.prototype.apply = function(item) {

    };

    // イベント処理を設定
    Favorite.prototype.addClickHandler = function() {
    };

    // お気に入り登録処理
    Favorite.prototype.add = function(item) {
    };

    // お気に入り解除処理
    Favorite.prototype.remove = function(item) {
    };

    // アプリ+端末を特定するためのuuidを取得
    // uuidはアプリアンインストールで削除されます
    var getUuid = function() {
    };
    return Favorite;
})();
```

各メソッドの内容は以下に説明します。

初期設定

この処理では以下のような処理を行っています。

1. ニフティクラウド mobile backendの初期化&データストアのお気に入りクラスを登録
2. 変数の設定
3. タップした時の処理（イベント処理を設定する、を呼び出し）

ニフティクラウド mobile backendの初期化&データストアのお気に入りクラスを登録（7行目）

ニフティクラウド mobile backendの初期化は先ほど行った通りです。インポートした段階ではNCMBオブジェクトは定義されていませんので処理分けしてあります。NCMBオブジェクトがあれば初期化と、データストアのお気に入りクラス（favorite）を定義しています。

```
if (typeof NCMB !== 'undefined') {
    this.ncmb = new NCMB(options.applicationKey, options.clientKey);
    // 保存先クラスを定義するところ
    this.FavoriteClass = this.ncmb.DataStore("favorite");
}
```

変数の設定

今回使う変数は以下の通りです。

- 記事リストを指定するためのID
- アプリ＋端末を特定するためのuuidを取得
- お気に入りのOn/Offイベントの有効フラグ

それぞれ次のように定義されます。

```
// 記事リストを指定するためのID
this.listEl = "#feed-list";

// アプリ＋端末を特定するためのuuidを取得
this.uuid = getUuid();

// お気に入りのOn/Offイベントの有効フラグ
this.clickEnabled = true;
```

getUuid については後述します。

全ての記事に対してお気に入り状況を反映させる（31行目）

この処理は記事一覧ごとにapplyメソッドを呼び出すだけです。

```
Favorite.prototype.applyAll = function() {
    var self = this;
        $(this.listEl).children('li').each(function(index) {
        var item = $(this);
        self.apply(item);
    });
};
```

一つの記事に対してお気に入り状態を反映させる

この処理は次のステップ以降で作っていきます。ニフティクラウド mobile backendを検索して、お気に入りの登録状態をチェックします。

```
Favorite.prototype.apply = function(item) {

};
```

イベント処理を設定（45行目）

イベント処理は星マークのクリックイベントの管理です。jQueryを使ってイベントハンドリングを行っています。上記で定義した this.listEl（#feed-list）以下にある .star をクリック（またはタップ）した際にイベントが発火します。処理内容としては、星マークの状態を使って、お気に入りの登録または削除を行います。

```
Favorite.prototype.addClickHandler = function() {
    var self = this;

    // 記事一覧の中のstarクラスに対してイベントを指定します。
    // スマートフォンの場合はtapstart、デスクトップの場合はclickイベントとします
    $(this.listEl).on('click', '.star', function(event) {
        // タップ設定が有効であれば処理を行います
        // これは二重処理の防止です
        if (self.clickEnabled == true) {
            // 一旦二重処理を防ぎます
            self.clickEnabled = false;

            // フラグは1秒後に立て直します
            setTimeout(function(){ self.clickEnabled = true; }, 1000);

            // 星マークのクラスで処理を判別します。
            if ($(this).hasClass('fa-star-o')) {
                // 空であればお気に入り未登録→お気に入り登録処理
                self.add($(this).closest('li'));
            } else {
                // 塗りつぶされている場合はお気に入り登録済み→お気に入り解除処理
                self.remove($(this).closest('li'));
            }
        }
        event.stopPropagation();
    });
};
```

お気に入り登録処理（74行目）

ここは今後のステップで作成します。今はダミーのコードがあります。

```
Favorite.prototype.add = function(item) {
    var self = this;

    // タップしたデータのURLを取得
    var url = item.data('link');

    // ここから下はダミーです。後ほど消します。
    var icon = item.find("i");
    icon.addClass('fa-star');
    icon.removeClass('fa-star-o');
    // ここまではダミーです。後ほど消します。

    // 保存するオブジェクトを生成

    // 保存したい値をセットし、保存

};
```

お気に入り解除処理（94行目）

お気に入り削除の処理も、今後のステップで作っていきます。今は最低限の変数だけ書いてあります。

```
Favorite.prototype.remove = function(item) {
    var self = this;
    var url = item.data('link');

    // uuidとurlの両方が合致するオブジェクトを検索し、見つけたものを削除する

};
```

アプリ+端末を特定するためのuuidを取得（104行目）

この処理は端末ごとにユニークなIDを生成することで自分のお気に入りなのか、別端末のものなのかを判別するのに使っています。このような方法があるのかと見ておいてもらう程度で構いません。なお、ニフティクラウド mobile backendには会員管理や、匿名会員機能がありますので、それを使って管理することもできます（時間の都合上、今回は扱いませんが、興味がある方はぜひチェックしてみてください）。

```
var getUuid = function() {
    var uuid = localStorage.getItem('uuid');
    if (uuid === null) {
      // uuid未生成の場合は新規に作る
      var S4 = function(){
        return (((1+Math.random())*0x10000)|0).toString(16).substring(1);
      };
      uuid = (S4()+S4()+"-"+S4()+"-"+S4()+"-"+S4()+"-"+S4()+S4()+S4());
      localStorage.setItem('uuid', uuid);
    }
    return uuid;
};
```

動かしてみる

プレビューで動かしてみましょう（Monacaデバッガーでも可）。クリックすると、星が黄色になるのが確認できるはずです。黄色いのがお気に入りに登録されている状態です。

ただし、右上にある再読み込みマークをクリックすると、お気に入り状態が解除されてしまいます。これはデータがHTML上だけでしかなく、どこにも保存されていないためです。

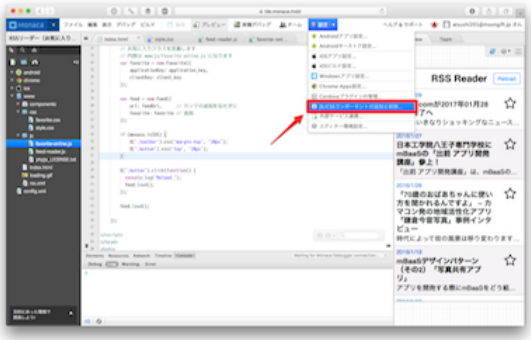
次のステップ、4. RSSリーダー（お気に入り登録機能付き）を開発する（その1）から、お気に入りをニフティクラウド mobile backendに登録していきたいと思います。

4. ニフティクラウド mobile backendの準備 & お気に入り登録機能を開発する

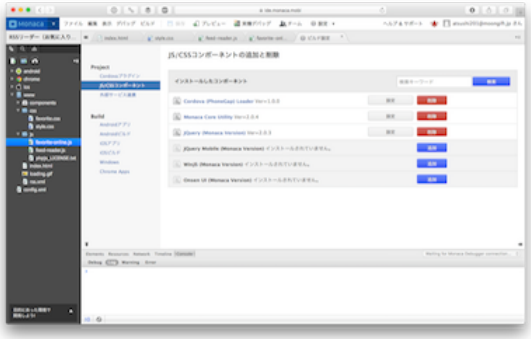
このステップではニフティクラウド mobile backendの利用準備と、お気に入り登録機能を作ってみたいと思います。

ニフティクラウド mobile backendのJavaScript SDKをインストールする

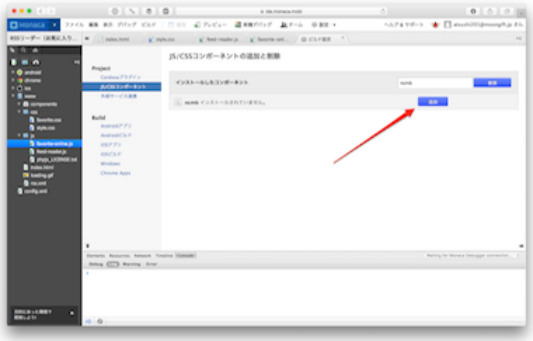
まず最初に設定メニューにある **JS/CSSコンポーネントの追加と削除** をクリックします。



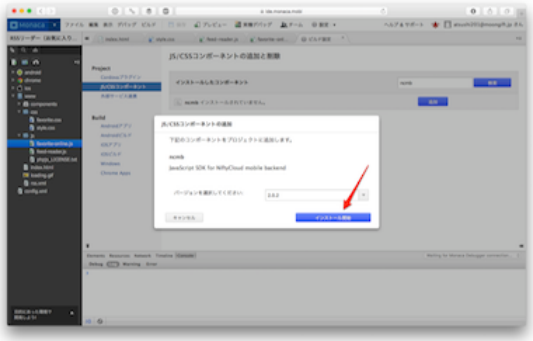
そして出てきたウィンドウで、右側にある検索キーワードに **ncmb** と入力して検索ボタンを押します。



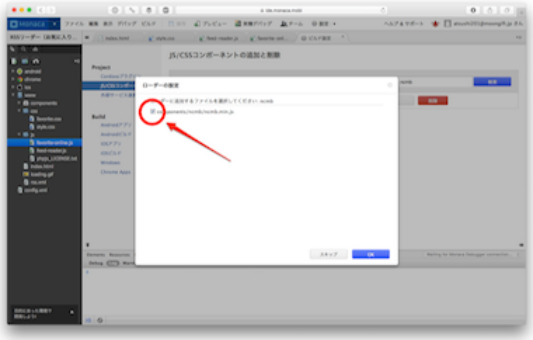
ncmbが結果に出てきますので、追加ボタンを押します。



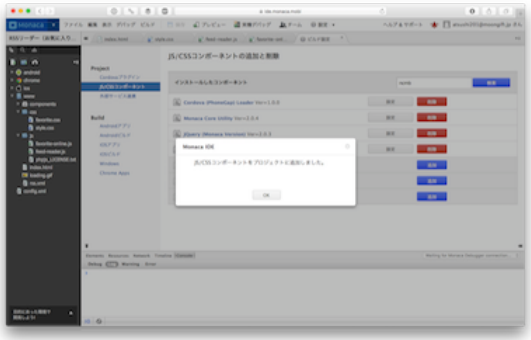
バージョンは 2.0.2（2016年2月時点）になっていればOKです。インストール開始を押します。



ローダーの設定というのが出ますので、**忘れずにチェックをつけてください**。



追加しました、というアラートが出たら完了です。これでニフティクラウド mobile backendのJavaScript SDKがインストールされました。



ニフティクラウド mobile backendの初期設定を行う

初期設定は前に行ったアプリケーションキーとクライアントキーの設定になります。ニフティクラウド mobile backendの管理画面に入って、アプリケーションキーとクライアントキーをそれぞれ `www/index.html` の所定の場所を置き換えてください。

www/index.htmlの19行目から

```
var application_key = "YOUR_APPLICATION_KEY";
var client_key      = "YOUR_CLIENT_KEY";
```

これで初期設定が完了です。

お気に入り登録処理を実装する

では今回はまずお気に入りを登録するところまで処理をしてみたいと思います。 `js/favorite-online.js` を修正していきます。先ほど星マークをクリックすると色が変わるのを確認してもらったと思いますので、その処理を変更します。

js/favorite-online.jsの74行目付近

```
// お気に入り登録処理
Favorite.prototype.add = function(item) {
    var self = this;

    // タップしたデータのURLを取得
    var url = item.data('link');

    // ここから下はダミーです。後ほど消します。
    var icon = item.find("i");
    icon.addClass('fa-star');
    icon.removeClass('fa-star-o');
    // ここまではダミーです。後ほど消します。

    // 保存するオブジェクトを生成

    // 保存したい値をセットし、保存

};
```

修正後

```
// お気に入り登録処理
Favorite.prototype.add = function(item) {
    var self = this;

    // タップしたデータのURLを取得
    var url = item.data('link');

    // 保存するオブジェクトを生成

    // 保存したい値をセットし、保存

};
```

まず前に行ったようにデータストアのインスタンスを作成します。

```
// 保存するオブジェクトを生成
var favorite = new this.FavoriteClass();
```

次に保存したい値をセットします。今回はURLと、端末を指し示すUUIDをセットします。値の設定は set メソッドを使います。一つ目の引数がカラム名、二つ目の引数は値になります。

```
favorite.set("uuid", self.uuid)
        .set("url", url)
```

そしてそのまま保存処理を実行します。保存処理は `save()` になります。保存処理がうまくいけば `then()` メソッドに、エラーが起きた場合は `catch()` メソッドに返ってきます。キーチェーンメソッドでつなげられますので、上記の処理からそのまま継続します。

この時、保存がうまくいったら（いかなくても）お気に入り登録状態を反映する `apply` メソッ

ドを呼び出しておきます。なお、まだ `apply` メソッドの中身はありませんので今後作っていきましょう。

```
favorite.set("uuid", self.uuid)
  .set("url", url)
  // 保存したい値をセットし、保存
  .save()
  .then(function(favorite){
    // 保存が成功した場合
    self.apply(item);
  })
  .catch(function(error){
    // 保存が失敗した場合
    self.apply(item);
  });
```

ここまでで登録処理が完了になります。お気に入り登録機能では次のようになります。

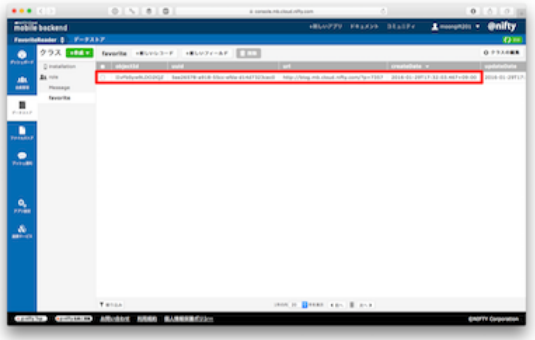
```
// お気に入り登録機能
Favorite.prototype.add = function(item) {
  var self = this;
  var url = item.data('link');

  // 保存するオブジェクトを生成
  var favorite = new this.FavoriteClass();
  favorite.set("uuid", self.uuid)
    .set("url", url)
    // 保存したい値をセットし、保存
    .save()
    .then(function(favorite){
      // 保存が成功した場合
      self.apply(item);
    })
    .catch(function(error){
      // 保存が失敗した場合
      self.apply(item);
    });
};
```

動かしてみる

では実際に動かしてみましょう。星マークをクリックしてみます。今回は特に画面上の変化は何もありません。

うまく登録が行われていると、ニフティクラウド mobile backendの管理画面を開き、データストアを表示するとfavoriteクラスが追加されているはずです。

favoriteという表示をクリックすると、データが登録されているのが分かるでしょう。



今回までで、まずはお気に入り登録機能ができあがりました。次のステップではお気に入り登録状

態を画面に反映してみます。

なお、ここまでのコードはCW_Monaca_NCMB_Handson/4-project at master · NIFTYCloud-mbaas/CW_Monaca_NCMB_Handsonにアップしてあります。うまく動かない場合は参考にしてください。

5. お気に入り登録状態を画面に反映する

このステップでは先ほど登録したお気に入りのデータを取得して、画面に反映するところまで行います。処理としては、 `Favorite.prototype.apply` の開発になります。

js/favorite-online.jsの修正（41行目付近）

元々のコードは次のようになっていますので探してください。ここで送られてくるitemという変数は記事一覧の一つ一つのデータになります。

js/favorite-online.jsの41行目付近

```
// 一つの記事に対してお気に入り状態を反映させる
Favorite.prototype.apply = function(item) {

};
```

処理の内容としては次のようになります。

1. 変数の定義
2. お気に入り登録数を検索
3. 自分がお気に入り登録済みか調べる
    1. 登録済みであれば星マークを黄色に
    2. 未登録の場合は星マークを黒に

```
// 一つの記事に対してお気に入り状態を反映させる
Favorite.prototype.apply = function(item) {
    // 変数の定義
    // お気に入り登録数を検索
    // 自分がお気に入り登録済みか調べる
        // 登録済みであれば星マークを黄色に
        // 未登録の場合は星マークを黒に
};
```

変数の定義

扱う変数は3つです。

- 自分自身（this）
- URL
- アイコン

URL、アイコンはitemから取得できます。

```
var self = this;
var url = item.data('link');
var icon = item.children('i');
```

お気に入り登録数を検索

お気に入り登録数はURLを使ってfavoriteクラスを検索します。この時、 `count()` を使うことでデータ件数だけを取得できます。絞り込み条件として、文字列の一致は `equalTo` を使います。一つ目の引数はカラム、二つ目は値（今回はURL）になります。他にも以上、以下や In、Not Inなどの条件が使えます。詳しくは基本的なクエリのオペランドを参照してください。

```
this.FavoriteClass
.equalTo("url", url)
.count()
```

そして、 `fetchAll()` メソッドで検索処理を実行します。

```
this.FavoriteClass.equalTo("url", url)
  .count()
  .fetchAll()
```

検索処理がうまくいった場合は `then()` メソッドに、失敗した場合は `catch()` メソッドが呼ばれます。

```
this.FavoriteClass
    .equalTo("url", url)
    .count()
    .fetchAll()
    .then(function(results){
      // 検索がうまくいった場合
    })
    .catch(function(error){
      // 検索が失敗した場合
    });
```

検索がうまくいった場合、 `results.count` で検索にマッチするデータの件数が取得できますので、これを星マークの横に表示します。失敗した場合は強制的に0を表示します。

結果として登録数の検索は次のようになります。

```
// お気に入り登録数を検索
this.FavoriteClass
    .equalTo("url", url)
    .count()
    .fetchAll()
    .then(function(results){
      if (results.count > 0) {
        icon.text(results.count);
      } else {
        icon.text("0");
      }
    })
    .catch(function(error){
      console.log(error.message);
      icon.text("0");
    });
```

自分がお気に入り登録済みか調べる

次に自分がそのURLを登録済みかどうかチェックします。これは先ほどの条件に加えて、UUIDも条件に追加します。

```
this.FavoriteClass
    .equalTo("uuid", self.uuid)
    .equalTo("url", url)
    .count()
```

そして `fetchAll()` メソッドを実行し、返ってきた結果が 0 であれば未登録、0以上であれば登録済みと判定します。今回はエラーの場合はメッセージだけ出すのみとしています。

```
.fetchAll()
.then(function(results){
    if (results.count > 0) {
        // 登録済みであれば星マークを黄色に
        icon.addClass('fa-star');
        icon.removeClass('fa-star-o');
    } else {
        // 未登録の場合は星マークを黒に
        icon.removeClass('fa-star');
        icon.addClass('fa-star-o');
    }
})
.catch(function(error){
    console.log('own favorite check error: ' + error.message);
});
```

自分がお気に入り登録しているかどうかの判定処理は次のようになります。

```
// 自分がお気に入り登録済みか調べる
this.FavoriteClass
    .equalTo("uuid", self.uuid)
    .equalTo("url", url)
    .count()
    .fetchAll()
    .then(function(results){
        if (results.count > 0) {
            // 登録済みであれば星マークを黄色に
            icon.addClass('fa-star');
            icon.removeClass('fa-star-o');
        } else {
            // 未登録の場合は星マークを黒に
            icon.removeClass('fa-star');
            icon.addClass('fa-star-o');
        }
    })
    .catch(function(error){
        console.log('own favorite check error: ' + error.message);
    });
```

表示を確認する

`apply()` メソッド全体は次のようになります。

```
// 一つの記事に対してお気に入り状態を反映させる
Favorite.prototype.apply = function(item) {
    // 変数の定義
    var self = this;
    var url = item.data('link');
    var icon = item.children('i');

    // お気に入り登録数を検索
    this.FavoriteClass
        .equalTo("url", url)
        .count()
        .fetchAll()
        .then(function(results){
            if (results.count > 0) {
                icon.text(results.count);
            } else {
                icon.text("0");
            }
        })
        .catch(function(error){
            console.log(error.message);
            icon.text("0");
        });

    // 自分がお気に入り登録済みか調べる
    this.FavoriteClass
        .equalTo("uuid", self.uuid)
        .equalTo("url", url)
        .count()
        .fetchAll()
        .then(function(results){
            if (results.count > 0) {
                // 登録済みであれば星マークを黄色に
                icon.addClass('fa-star');
                icon.removeClass('fa-star-o');
            } else {
                // 未登録の場合は星マークを黒に
                icon.removeClass('fa-star');
                icon.addClass('fa-star-o');
            }
        })
        .catch(function(error){
            console.log('own favorite check error: ' + error.message);
        });
};
```

うまくできていれば、プレビューにて次のように表示されるはずです。先ほどお気に入り登録した記事だけ星マークが黄色くなって、数字が1になっていればOKです。

また、別な記事の星をクリックすると、その場で星マークが黄色に変わるのが確認できるはずです。

今回まででお気に入り登録ができ、表示に反映されるところまでできあがりました。次は最後のステップとして、お気に入り登録したデータを削除する処理を作っていきます。

なお、ここまでのコードはCW_Monaca_NCMB_Handson/5-project at master · NIFTYCloud-mbaas/CW_Monaca_NCMB_Handsonにアップしてあります。うまく動かない場合は参考にしてください。

6. お気に入り削除機能を開発する

このステップでは登録したお気に入りを削除する処理を作っていきます。処理としては `Favorite.prototype.remove` の開発となります。

js/favorite-online.jsの修正（135行目付近）

元々のコードは次のようになっていますので探してください。ここで送られてくるitemという変数はお気に入り登録の時と同じく、記事一覧の一つ一つのデータになります。

js/favorite-online.jsの135行目付近

```
// お気に入り解除処理
Favorite.prototype.remove = function(item) {
    var self = this;
    var url = item.data('link');

    // uuidとurlの両方が合致するオブジェクトを検索し、見つけたものを削除する

};
```

処理の内容としては次のようになります。

1. データストアを検索
2. データがあれば削除
3. 表示を更新

データストアを検索

データストアの検索はURLとUUIDを引数に行います。お気に入り登録の時はカウント（count）を使っていましたが、今回は実際のオブジェクトを得るようにします。そのため、 `fetch()` （データが1件のみ対象なので）を使います。

```
this.FavoriteClass
  .equalTo("uuid", self.uuid) // UUIDとURLで検索します
  .equalTo("url", url)
  .fetch() // 今回は count ではなく fetchを使います
```

データの検索がうまくいった場合には `then()` メソッドが、エラーが出た場合は `catch()` が呼ばれるのは変わりません。

```
this.FavoriteClass
  .equalTo("uuid", self.uuid) // UUIDとURLで検索します
  .equalTo("url", url)
  .fetch() // 今回は count ではなく fetchを使います
  .then(function(favorite){
    // データが見つかった場合
  })
  .catch(function(error){
    // エラーがあった場合
  });
```

データが見つかった場合、引数（今回はfavorite）に該当データが入ります。データの削除は `delete` メソッドを使います。

```
favorite.delete()
```

データの削除についてもうまくいった場合は `then()` メソッドが、エラーが出た場合は `catch()` が呼ばれます。

```
favorite.delete()
  .then(function(result){
    // 削除処理がうまくいった場合
  })
  .catch(function(error){
    // 削除処理がうまくいかなかった場合
  });
```

削除処理がうまくいってもいかなくても、 `apply()` メソッドを呼んで表示を更新することにします。そうすると全体の処理としては次のようになります。

```
// お気に入り解除処理
Favorite.prototype.remove = function(item) {
  var self = this;
  var url = item.data('link');

  // uuidとurlの両方が合致するオブジェクトを検索し、見つけたものを削除する
  this.FavoriteClass
    .equalTo("uuid", self.uuid) // UUIDとURLで検索します
    .equalTo("url", url)
    .fetch() // 今回は count ではなく fetchを使います
    .then(function(favorite){
      // データが見つかった場合
      favorite.delete()
      .then(function(result){
        // 削除処理がうまくいった場合
        self.apply(item);
      })
      .catch(function(error){
        // 削除処理がうまくいかなかった場合
        self.apply(item);
      });
    })
    .catch(function(error){
      // エラーがあった場合
      self.apply(item);
    });
};
```

確かめてみる

これによってお気に入り登録データを検索して、あれば削除して表示を更新するという流れができあがりました。実際にすでに黄色になっている星マークをクリックすると、数字が0に戻って黒い星になるのが確認できるはずです。

これでRSSリーダー（お気に入り登録機能付き）の実装は完了です。発展版として、会員機能を使って本格的にデータのシェアできる環境を作ったり、プッシュ通知を実装する、フィードのURLを変更できるようにするといった機能が考えられます。ぜひこれをベースに様々な機能を追加してみてください。

ここまでのコードはCW_Monaca_NCMB_Handson/6-project at master · NIFTYCloud-mbaas/CW_Monaca_NCMB_Handsonにアップしてあります。うまく動かない場合は参考にしてください。