pjax - *jquery plugin*

10

pjaxはデータの読み込みと描画の冗長部分を省略することで非常に高速かつ低コストなページ移動を実現する、HTML5で実装される次期標準ブラウジング機能です。

また、キャッシュ機能によりサーバーへのアクセスと負荷を軽減することで、高性能なサーバーでなくとも多くのPVアクセスを処理することが可能になります。

このpjaxプラグインはサーバーに手を加えることなく簡単に導入することができます。
WordPressにも10分ほどで導入できます。

最新版のダウンロード

プラグインの最新版をGitHubで公開しています。ダウンロードはこちらから。
GitHub：https://github.com/falsandtru/jquery.pjax.js
ファイル：https://raw.github.com/falsandtru/jquery.pjax.js/master/jquery.pjax.js
圧縮版：https://raw.github.com/falsandtru/jquery.pjax.js/master/jquery.pjax.min.js

概要

サイト内のページ移動において指定したHTML要素（異なるコンテンツを持つ領域）のみ更新することでページ移動を高速化します。
たとえば、このサイトのpjaxによるトップページへのページ移動時間は、ajaxによりサーバーからデータを取得した場合でも最短で100-200ミリ秒しかかかりませんが、pjaxのキャッシュ機能を有効にした場合のページ移動時間はわずか20-30ミリ秒（0.02-0.03秒）です。

※Windows7 + Google Chromeでの例です。

※インストール直後のWordPressでは、ajaxで500ミリ秒、キャッシュで10ミリ秒となりました（サーバーはロリポップを使用）。

※このサイトではブラウザのコンソールにページ移動にかかった時間を出力しており、コンソールからユーザーが実際にページ移動にかかった時間を見ることができます。

※動作テストのためpjaxが正常に動作していないことがあります。恐縮ですがその際は時間をおいて再度ご覧ください。

特徴

- jQuery 1.4.2から動作します。
- 移動先のページのCSSとJavaScriptを自動的に読み込めます。
- キャッシュ機能によりサーバーへのアクセスと負荷を軽減します。
- キャッシュをspageと共有しています。
- サーバー側の設定やコードのインストール等の作業が不要です。
- 同時に複数の領域を更新できます。
- ページごとのpjaxによる移動の可否をHTMLで設定できます。
- 更新までの待ち時間を設定できます。
- 内部的に使用される `$.ajax` のオプションをほぼすべて設定できます。

対応

- CSSの読み込み
- JavaScriptの読み込み
- Android・iOSでの使用

- フォームのsubmitによるページ遷移
- Google Analytics によるアクセス解析
- WordPressへの導入
- 文字コードの異なるページの読み込み
- ローディングエフェクトの表示
- サーバーからの差分データによるページ更新
- キャッシュによるページ更新

defunkt版との比較

このpjaxプラグインは独自に開発されており、pjaxの本家であるdefunkt版の派生ではないため仕様が異なります。defunkt版（v1.7.0/2013年6月現在最新版）との主な違いは次のとおりです。

項目	defunkt版	falsandtru版
jQueryバージョン対応	1.8.x	1.4.2
Android・iOSへの対応 locationオブジェクトの更新※1	×	○
Android・iOSへの対応 スクロール位置の操作※2	×	○
ページ移動方法の自動切替 HTML以外のコンテンツへのアクセス※3	×	○
JavaScriptの実行順序維持※4	×	○
JavaScriptの読み込み 埋め込み型	×	○
CSSの読み込み	×	○
キャッシュ制御※5	×	○
複数領域の更新	×	○
ユーザー定義関数の実行形式	イベント	コールバック
ユーザー定義関数の設定箇所	9	27
部分的更新キャンセル※6	×	○
比較用デモ	defunkt	falsandtru

※ 記述に間違いがありましたら掲示板または連絡フォームからご連絡ください。

※1 AndroidとiOSでは `location` オブジェクトが `pushState` を使用しても更新されず、ブラウザのアドレスバーに表示されるURLと `location.href` により取得するURLが一致しない不具合が報告されています。jQueryMobileではこの不具合を解決しているものと思われますが（未確認）、手法が判然としなかったため当プラグインではこれと異なる手法で解決し `location` オブジェクトを更新しています。defunkt版ではアドレスバーのURLと `location.href` が別のページを指しており不具合が解決されておらず、致命的なバグが発生する可能性があるため十分注意してください。Google Analytics などは間違ったアクセスログを生成することになります。jQueryMobileの手法がお分かりの方は掲示板または連絡フォームからご連絡いただけると助かります。

※2 AndroidとiOSではページ遷移時にjQueryの `scrollTop` メソッドでスクロール位置が操作できず、当プラグインではjQueryMobileと同じく `scrollTo` メソッドを使用することでこの問題を解決しています。defunkt版では `scrollTop` メソッドを使用しているため問題が解決されていません。

※3 リンク先がJavaScriptなどHTMLページ以外を参照していた場合にContent-Typeを参照してページ移動方法を自動的にpjaxから通常のものに切り替えます。

※4 defunkt版のDOMオブジェクトの生成によるJavaScriptの動的読み込みはオライリーの「続・ハイパフォーマンスWebサイト」で実行順序が維持されない読込方法に分類されています。

※5 defunkt版はキャッシュを無効にできないようです。

※6 タイトルやURLなどの更新を個別にキャンセルできます。コールバック関数を非同期に実行している場合はキャンセルできません。

使用法

jQuery

v1.7.2の使用を推奨します。
v1.4.2から動作します。

Register

$.pjax(Parameter as object)

リンクにpjaxを登録します。 `document` オブジェクトにデリゲートを設定します。

```
$.pjax({ area: '.container' });
```

$.fn.pjax(Parameter as object)

コンテキストに含まれるリンクにのみpjaxを登録します。コンテキストにデリゲートを設定します。
`link` プロパティと `form` プロパティはコンテキストにより絞り込まれますが、 `area` プロパティは絞り込まれません。

```
$('.delegate').pjax({ area: '.container' });
```

Parameter

パラメータはすべてパラメータ用オブジェクトのプロパティに設定して渡します。パラメータとなるオブジェクトのプロパティは以下のとおりです

gns: Namespace as string

グローバルネームスペースです。通常は設定は不要です。

ns: Namespace as string

ネームスペースです。ネームスペースを設定する場合はこちらを使用してください。

area: Selector as string（必須）

pjaxにより更新するコンテンツ（HTML要素）をjQueryセレクタで選択します。
`$.fn.pjax` により選択されたコンテキストで絞り込まれません。

link: Selector as string

pjaxによりページ移動を行うリンク（アンカータグ）をjQueryセレクタで選択します。
初期値は、 `href` 属性の値が `/` で始まり、 `target` 属性がない `anchor` 要素です。ルートパス以外のリンクは対象外となっています。
`$.fn.pjax` により選択されたコンテキスト内で選択されます。

form: Selector as string

pjaxによりページ移動を行うフォーム（フォームタグ）をjQueryセレクタで選択します。リンクとフォームへのpjaxの設定は個別に行う必要があり、まとめて行うことはできません。
初期値は `undefined` でpjaxはフォームによるページ移動に使用されません。
`$.fn.pjax` により選択されたコンテキスト内で選択されます。

scrollTop: number / null

ページ移動後の縦方向のスクロール位置を設定します。 `null` を設定すると移動前のスクロール位置を維持します。初期値は `0` です。

scrollLeft: number / null

ページ移動後の横方向のスクロール位置を設定します。 `null` を設定すると移動前のスクロール位置を維持します。初期値は `0` です。

ajax: object

pjaxで内部的に使用される `$.ajax` のオプションを設定します。 `$.ajax` のコールバック関数はすべて上書きされるため使用できません。代わりに `callbacks.ajax` で設定できるのでこちらを使用してください。

contentType: string

移動先として読み込むデータで許容するコンテントタイプをカンマまたはセミコロン区切りの文字列で設定します。初期値は `text/html` です。

load: node

pjaxによるページ読み込み時のCSSとJavaScriptを読み込みにかかる設定項目を持ちます。 `load.css` と `load.script` を有効にすることで、ページ別にCSSやJavaScriptが存在するサイトでも配置や構成を変えることなくpjaxを導入することができます。

load.css: boolean

pjaxによるページ読み込み時にCSSを読み込むかを設定します。初期値は `false` で読み込みません。
読み込まれるページの、現在のページに存在しないすべてのCSS（ `link rel="stylesheet"` 要素および `style` 要素）を読み込みます。読み込まれるページに存在しないCSSは削除されます。読み込まれたCSSはすべてDOMの `head` 要素末尾のノードとして追加されます。

load.script: boolean

pjaxによるページ読み込み時にJavaScriptを読み込むかを設定します。初期値は `false` で読み込みません。
読み込まれるページの、現在のページに存在しないすべてのJavaScript（ `script` 要素）を読み込みます。外部ファイル以外の形式のJavaScriptは同一の内容であっても再度読み込まれます。jQueryの仕様により、JavaScriptは読み込まれていてもDOMに追加されません。
pjaxによるJavaScriptの実行順序は、HTML上の記述順序（通常の読み込み順序）と同じであることが保障されません。外部ファイル形式のJavaScriptと埋め込み形式のJavaScriptでは実行タイミングが異なるため、同一形式間内での実行順序は保たれますが、異なる形式間での実行順序は保たれません。また、埋め込み形式のJavaScriptの実行はすべての外部ファイル形式のJavaScriptの実行を待ってから行われます。このため、外部ファイル形式のJavaScriptが実行される前に埋め込み形式のJavaScriptがすでに実行されていなければならないような設計は避ける必要があります。
ページの表示直後にすべて実行されている必要のないJavaScriptは、ページ読み込み時に一括で実行せずjquery.displaytrigger.jsにより随時実行することで負荷を削減することを推奨します。ページの表示直後にすべて読み込まれている必要のないコンテンツについても同様です。

load.sync: boolean

CSSと `defer` 属性を持つJavaScript（ `script` 要素）の非同期読み込みを、pjaxによるコンテンツの更新の描画を待ってから行います。初期値は `true` で有効です。
`load.sync` による同期（的）処理は、JavaScriptの読み込み処理を同期的に開始できるように実行タイミングを調整して行うものであり、pjaxによるCSSとJavaScriptの読み込み処理自体は `load.sync` `load.async` の設定にかかわらずすべて非同期で行われます。

load.async: Millisecond as number

`defer` 属性を持たないJavaScript（ `script` 要素）の非同期読み込みをpjaxによるコンテンツの更新の描画を待たずに開始する、コンテンツの更新からの経過時間（遅延時間）をミリ秒で設定します。初期値は `0` です。

interval: Millisecond as number

pjaxにより更新されたコンテンツの描画の確認を行う間隔をミリ秒で設定します。初期値は `300` です。

cache: node

pjaxによるページ読み込み時のキャッシュの使用にかかる設定項目を持ちます。独自に作成したキャッシュを使用することでサーバーと通信を行わずにページを移動することができるため、サーバーへのアクセスと負荷を軽減することができます。また、サーバーへのリクエスト時にキャッシュが使用されることはないため、リロードによる最新のデータへのアクセスを妨げません。ページに期限が設定されキャッシュされるよう設定されている場合はブラウザのキャッシュ機能が使用できるためpjaxのキャッシュ機能は無効にすることを推奨します。
キャッシュはページを閉じるかpjaxとspage以外によりページを読み込むまで保持されます。初期設定では無効です。

cache.click: boolean

リンクのクリックによるページ移動にキャッシュを使用するかを設定します。初期値は `false` で無効です。

cache.submit: boolean

フォームの送信によるページ移動にキャッシュを使用するかを設定します。初期値は `false` で無効です。

cache.popstate: boolean

ブラウザの操作によるページ移動にキャッシュを使用するかを設定します。初期値は `false` で無効です。

cache.length: number

キャッシュを保持するページ数の上限を設定します。初期値は `9` です。

cache.size: Byte as number

キャッシュを保持するデータサイズの上限をバイト数で設定します。初期値は `1048576` (1MB)です。

cache.expire: Millisecond as number

キャッシュの有効期間をミリ秒で設定します。初期値は `1800000` (30分)です。

wait: Millisecond as number

`$.ajax` の実行からコンテンツの更新までの最低待ち時間を設定します。jQuery 1.5より前のバージョンでは無効です。

fallback: boolean / function(event)

pjaxによるページ移動が失敗した場合の対応を行うかを設定します。初期状態では代替処理として通常のページ移動が行われます。関数が設定された場合は代替処理が当該関数により上書きされます。処理はエラーにかかるコールバック関数を実行後に行われます。初期値は `true` で有効です。

server: node

サーバーとの通信にかかる設定項目を持ちます。

server.query: Query as string

pjaxによるサーバーへリクエストではページのURLにpjaxによるリクエストであることを通知するためのクエリ名が追加されており、このクエリ名を設定します。このクエリは内部処理でのみ使用されるためサイトの閲覧者の目に触れることはありません。初期値は `gns` の設定値と同じであり、 `?pjax=1` のようにクエリが付加されます。

callback: function(event, parameter, data, dataType, XMLHttpRequest)

ページ移動後に実行されるコールバック関数を設定します。ページの更新処理が成功したときに `update.complete( event, parameter, data, dataType, XMLHttpRequest )` の直後に実行されます。コールバック関数にはイベントの発生もとのオブジェクトがコンテキストとして与えられます。 `callback` `callbacks` ともに `callbacks.async` に `true` を設定することでコールバック関数の実行を非同期に行えます。コールバック関数を非同期で実行することで処理を高速化することができますが、戻り値に `false` を設定することによる処理のキャンセルができなくなります。

parameter: any

すべてのコールバック関数に共通で渡されるパラメータを設定します。

callbacks: object

内部の各タイミングにおいて実行されるコールバック関数を設定します。コールバック関数にはイベントの発生もとのオブジェクトがコンテキストとして与えられます。
`ajax` を除くすべてのコールバック関数は戻り値に `false` を設定することで現在の処理を抜けることができます。 `before` では以降の処理をすべてキャンセルします。このときフォールバック処理は `fallback` の設定にかかわらず行われません。 `update.any.before` `update.any.after` ではページ更新処理のうちanyの示す部分の更新処理をキャンセルないし抜けます。ページ移動でエラーが発生した際に `update.error` `update.complete` で処理を抜けるとフォールバック処理が `fallback` の設定にかかわらず行われません。

使用できる `callbacks` のプロパティと渡されるパラメータ、実行タイミングは次のとおりです。

async
コールバック関数の実行を非同期にするかを設定します。初期値は`false`で無効です。

before(event, parameter)
コード上の実行順序において最初に実行されます。

after(event, parameter)
コード上の実行順序において最後に実行されます。

ajax.xhr(event, parameter)
ajax通信において同名のメソッド内で実行されます。

ajax.beforeSend(event, parameter, data, dataType)
〃

ajax.dataFilter(event, parameter, data, dataType)
〃

ajax.success(event, parameter, data, dataType, XMLHttpRequest)
〃

ajax.error(event, parameter, XMLHttpRequest, textStatus, errorThrown)
〃

ajax.complete(event, parameter, XMLHttpRequest, textStatus)
〃

update.before(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理において最初に実行されます。

update.after(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理において最後に実行されます。

update.cache.load.before(event, parameter, cache)
ページの更新処理においてcacheの読み込み前に実行されます。

update.cache.load.after(event, parameter, cache)
ページの更新処理においてcacheの読み込み後に実行されます。

update.title.before(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理においてタイトルの更新前に実行されます。

update.title.after(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理においてタイトルの更新後に実行されます。

update.content.before(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理においてコンテンツの更新前に実行されます。

update.content.after(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理においてコンテンツの更新後に実行されます。

update.css.before(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理においてCSSの読み込み前に実行されます。

update.css.after(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理においてCSSの読み込み後に実行されます。

update.script.before(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理においてJavaScriptの読み込み前に実行されます。

update.script.after(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理においてJavaScriptの読み込み後に実行されます。

update.cache.save.before(event, parameter, cache)
ページの更新処理においてcacheの作成前に実行されます。

update.cache.save.after(event, parameter, cache)
ページの更新処理においてcacheの作成後に実行されます。

update.verify.before(event, parameter)

ページの更新処理において更新結果の検証前に実行されます。

update.verify.after(event, parameter)
ページの更新処理において更新結果の検証後に実行されます。

update.success(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理が成功したときに実行されます。

update.error(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理が失敗したときに実行されます。

update.complete(event, parameter, data, dataType, XMLHttpRequest)
ページの更新処理が完了したときに実行されます。

Method

なし

Property

なし

記述例

導入

シンプルな実行例です。リンクをクリックするとPrimaryのみ更新されます。

demo

```
$.pjax({ area: 'div.pjax' });
```

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="UTF-8">
<meta http-equiv="content-language" content="ja">
<title>pjax</title>
<script type="text/javascript" charset="utf-8" src="/lib/jquery-1.7.2.min.js"></script>
<script type="text/javascript" charset="utf-8" src="/lib/jquery.pjax.min.js"></script>
<script type="text/javascript">
$(function(){
  $.pjax({ area: 'div.pjax' });
});
</script>
<style type="text/css">

/* 省略 */

</style>
</head>
<body>
  <div id="container">
    <div id="header">
      <div class="layout">
        <p>header1</p>
        <p>pjax demo</p>
      </div>
    </div>
    <div id="wrapper" class="clearfix">
      <div class="layer">
        <div class="primary pjax">
          <div class="layout">
            <p>primary1</p>
            <p>pjax enable あア亜</p>
            <ul>
              <li><a href="/output/pjax/demo/install/">page1 enable</a></li>
              <li><a href="/output/pjax/demo/install/2.html">page2 enable</a></li>
              <li><a href="/output/pjax/demo/install/3.html">page3 enable</a></li>
            </ul>
          </div>
        </div>
        <div class="secondary">
          <div class="layout">
            <p>secondary1</p>
```

```
            <ul>
              <li><a href="/output/pjax/demo/install/">page1 enable</a></li>
              <li><a href="/output/pjax/demo/install/2.html">page2 enable</a></li>
              <li><a href="/output/pjax/demo/install/3.html">page3 enable</a></li>
            </ul>
          </div>
        </div>
        <div class="tertiary">
          <div class="layout">
            <p>tertiary1</p>
          </div>
        </div>
      </div>
    </div>
    <div id="footer">
      <div class="layout">
        <p>footer1</p>

      </div>
    </div>
  </div>
</body>
</html>
```

更新範囲 - area

pjaxによる更新範囲を選択します。次のように複数の範囲を同時に更新することもできます。双方のページで更新範囲が一致していないか片方に更新範囲がひとつもない場合は更新を中断し `update.error` を返します。

demo

```
$.pjax({ area: 'div.primary.pjax, div.tertiary.pjax' });
```

リンク - $.fn.pjax, link

pjaxによりページ移動を行うリンクを選択します。

demo

```
$('div.primary.pjax').pjax(
{
  area: 'div.primary.pjax'
});
```

`$.fn.pjax` によりコンテキストをpjaxの `area` プロパティの子孫要素（pjaxによる更新範囲内）に設定することはできません。pjaxによる更新範囲内でpjaxによりページ移動を行うリンクを選択するには `link` プロパティを使用してください。

demo

```
// NG
$('div.primary.pjax li').pjax(
...
```

```
// OK
$('div.primary.pjax').pjax(
{
  area: 'div.primary.pjax' ,
  link: 'li a:not([target])[href^="/"]'
});
```

CSS自動読み込み - load.css

pjaxによる移動先のページのCSSを自動的に読み込みます。移動先のページに存在しない現在のページのCSSは削除されます。

CSSの `link` 要素には必ず `rel="stylesheet"` を付けてください。 `link` 要素は `rel` 属性の値が `stylesheet` に大文字小文字半角全角含めて完全に一致しなければ読み込まれません。

demo

```
$.pjax(
{
  area: 'div.pjax' ,
  load: { css: true }
});
```

JavaScript自動読み込み - load.script

pjaxによる移動先のページのJavaScriptを自動的に読み込みます。

同一の外部ファイルにより記述されるJavaScriptは重複して読み込まれませんが、埋め込みにより記述される同一のJavaScriptは重複して実行されます。

demo

```
$.pjax(
{
  area: 'div.pjax' ,
  load: { script: true }
});
```

上記pjax登録処理は実際には外部ファイルに記述してください。埋め込みで記述した場合、pjax登録処理がページ移動ごとに不要に繰り返されます。

代替処理 - fallback

pjaxによるページ移動が失敗した場合に通常のページ移動を行います。初期値で有効になっているためこのための設定は不要です。

demo

```
$.pjax({ area: 'div.pjax' });
```

```
$.pjax({ area: 'div.pjax', fallback: true });
```

スクロール位置 - scrollTop, scrollLeft

pjaxによるページ移動後のスクロール位置を設定します。 `null` を設定すると移動前のスクロール位置を維持します。

demo

```
$.pjax({ area: 'div.pjax', scrollTop: null, scrollLeft: 50 });
```

最低待ち時間 - wait

`$.ajax` の実行からコンテンツの更新までの最低待ち時間を設定します。pjaxによるページ移動が速すぎる場合などに使用します。

demo

```
$.pjax({ area: 'div.pjax', wait: 1000 });
```

ajax通信設定 - ajax

pjaxで内部的に使用される `$.ajax` のオプションを設定できます。

```
$.pjax({ area: 'div.pjax', ajax: { timeout: 3000 } });
```

コールバックとパラメータ - callback, callbacks, parameter

コールバックに設定した関数を実行します。コールバック関数の第一引数はイベントオブジェクトが渡され、第二引数に設定したパラメータが渡され、以降は各もととなるコールバック関数に渡された引数を引き継ぎます。すべてのコールバック関数にはイベントの発生元のオブジェクトがコンテクストとして与えられます。例えば、 `anchor` 要素のクリックにより実行されるコールバックの `this` は `anchor` 要素であり、コールバック関数内で `this.href` などが使用できます。

demo

```
$.pjax(
{
  area: 'div.pjax' ,
  callback: function( event, arg ){ alert( arg + ': callback' ) ; } ,
  callbacks:
  {
    before: function( event, arg ){ alert( arg + ': before' ) ; } ,
    ajax:
    {
      beforeSend: function( event, arg ){ alert( arg + ': ajax.beforeSend' ) ; } ,
      dataFilter: function( event, arg, data ){ alert( arg + ': ajax.dataFilter' ) ; return d
      success: function( event, arg ){ alert( arg + ': ajax.success' ) ; } ,
      error: function( event, arg ){ alert( arg + ': ajax.error' ) ; } ,
      complete: function( event, arg ){ alert( arg + ': ajax.complete' ) ; }
    } ,
    update:
```

```
27      {
28        success: function( event, arg ){ alert( arg + ': update.success' ) ; } ,
29        error: function( event, arg ){ alert( arg + ': update.error' ) ; } ,
30        complete: function( event, arg ){ alert( arg + ': update.complete' ) ; }
31      } ,
32      after: function( event, arg ){ alert( arg + ': after' ) ; }
33    } ,
34    parameter: 'callback'
35  });
```

Google Analytics - callback

コールバックで `_gaq.push( ['_trackPageview'] )` を実行することでpjaxによるページ移動を Google Analytics に認識させることができます。

```
11  $.pjax(
12  {
13    area: 'div.pjax' ,
14    callback: function(){ if( window._gaq ){ _gaq.push( ['_trackPageview'] ) ; } }
15  });
```

`load.script` によりJavaScriptを有効にしている場合は移動先のページに埋め込まれているアクセス解析用のスクリプトが自動的に実行されますが、不要な部分まで実行されてしまうためアクセス解析用の記述を次のように置き換えることを推奨します。

```
1   if (!window._gaq) {
2     window._gaq = [];
3     window._gaq.push(['_setAccount', 'UA-xxxxxxxx-x']);
4     window._gaq.push(['_trackPageview']);
5
6     (function() {
7       var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
8       ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.goog
9       var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
10    })();
11  } else {
12    window._gaq.push(['_trackPageview']);
13  };
```

UTF-8以外の文字コードへの対応 - callbacks.ajax.beforeSend

`beforeSend` でMimeTypeをオーバーライドすることでUTF-8以外の文字コードを使用したHTMLを文字化けすることなく読み込むことができます。ただし、移動先のページの文字コードを事前に判別することができないため、複数の文字コードの混在したサイトでは文字コードの設定ができずpjaxは使用できません（CSSやJavaScriptなどの外部ファイルは異なる文字コードで作成されていても問題ありません）。
文字コード変換のデモは、当サイトのサーバーがUTF-8以外で作成されたページもUTF-8として強制的に表示させる設定となっていたことから正常に動作しないため公開していません。

```
$.pjax(
{
  area: 'div.pjax' ,
  callbacks:
  {
    ajax:
    {
      beforeSend: function( event, arg, XMLHttpRequest )
      {
          XMLHttpRequest.overrideMimeType( 'text/html;charset=UTF-8' ) ;
          XMLHttpRequest.overrideMimeType( 'text/html;charset=Shift_JIS' ) ;
          XMLHttpRequest.overrideMimeType( 'text/html;charset=EUC-JP' ) ;
      }
    }
  }
});
```

フォーム - form

pjaxでフォームの送信によるページ遷移を行います。

demo

```
$.pjax({ area: 'div.pjax', form: 'form.pjax' });
```

ローディングエフェクト - callback, callbacks

コールバックをカスタマイズすることでページ移動時にローディングエフェクトを表示させることができます。

demo

```
$.pjax(
{
  area: 'div.pjax',
```

```
    callback: function(){ $('div.loading').fadeOut(500); },
    callbacks:
    [
      before: function(){ $('div.loading').fadeIn(100); }
    ],
    ajax:
    [
      timeout: 3000
    ],
    wait: 1000
  });
        <div class="loading" style="background:url(/images/loading.png);display:none;position:fixed;top:0;left:0;z-index:9999;width:10
          <div style="position:absolute;top:45%;left: 50%;margin-top:-64px;margin-left:-64px;text-align:center;">
            <img src="/images/loading.gif" alt="" style="display:block;">
            <span style="font-size:18px;font-weight:bold;position:absolute;white-space:nowrap;">now loading...</span>
          </div>
        </div>
```

ローディングエフェクトは頻繁に表示されると煩わしいため多用しないことを推奨します。

ページ移動時のローディングエフェクトの使用量を削減するには、ロードに1秒以上かかった場合のみ1秒経過した時点からローディングエフェクトを表示するなどの方法が考えられます。

```
  $.pjax(
  {
    area: 'div.pjax',
    callback: function()
    {
      clearTimeout($.data($('div.loading').get(0), 'pjax-effect-id'));
      $('div.loading').fadeOut(500);
      $.data($('div.loading').get(0), 'pjax-effect-id', 0);
    },
    callbacks:
    [
      before: function()
      {
        clearTimeout($.data($('div.loading').get(0), 'pjax-effect-id'));
        $.data($('div.loading').get(0), 'pjax-effect-id', setTimeout(function(){ $('div.loading').fadeIn(100); }, 1000));
      }
    ],
    ajax:
    [
      timeout: 3000
    ],
    wait: 100
  });
```

サーバーへの対応 - PHPなどによる差分データを使用した更新

pjaxによる通信をサーバー側で識別し、pjax用の差分データを返させ、これを使用してページの更新（移動）を行います。データサイズを削減できるため、より少ない転送量と帯域で多くのアクセスを処理できます。

pjaxは通信時にHTTPリクエストヘッダに `X-Pjax` `X-Pjax-Area` `X-Pjax-CSS` `X-Pjax-Script` のフィールドと値を追加します。また、リクエストするURLに `?pjax=1` のようにクエリを追加します。サーバーはこれによりpjaxの使用の有無と必要なデータを知ることができます。なお、レスポンスヘッダの `Content-Type` に必ず `contentType` プロパティで設定したいずれかの値が含まれている必要があります。

demo ※これは移動先のページを差分データに置き換えた擬似的なデモです。

```
  $.pjax({ area: 'div.pjax' });
<html>
<head>
<title>pjax demo</title>
</head>
<body>
  <div class="primary pjax">
    <div class="layout">
      <p>primary2</p>
      <p>pjax enable あア7亜</p>
      <ul>
        <li><a href="/output/pjax/demo/server/">page1 enable</a></li>
        <li><a href="/output/pjax/demo/server/2.html">page2 enable</a></li>
        <li><a href="/output/pjax/demo/server/3.html">page3 enable</a></li>
      </ul>
    </div>
  </div>
</body>
</html>
<title>pjax demo</title>
<div class="primary pjax">
  <div class="layout">
    <p>primary3</p>
    <p>pjax enable あア7亜</p>
    <ul>
      <li><a href="/output/pjax/demo/server/">page1 enable</a></li>
      <li><a href="/output/pjax/demo/server/2.html">page2 enable</a></li>
      <li><a href="/output/pjax/demo/server/3.html">page3 enable</a></li>
    </ul>
  </div>
</div>
```

pjaxによる通信とそれ以外の通信により返すレスポンス（HTML）の切り替えは、ページごとにPHPにより行う方法もありますが、ページが大量にある場合はWordpressのように差分データをデータベースで管理するとより簡便です。アクセスされるページのURLはmod_rewriteを使用すれることで `http://example/a/b/c/` → `http://example/?dir1=a&dir2=b&dir3=c` のようにクエリに変換することができます。この方法であればページファイルは `http://example/index.php` １つで済み、あとはGETクエリに応じたSQLクエリを生成しデータベースから必要なデータを持ってくるだけです。ただし、サーバーにpjax用の差分データを返させる（静的なページから動的なページと構成に変更する）場合は、大なり小なりサーバーの負荷が増加し従前より処理能力が低下する可能性があることに留意してください。

WordPressへの導入

WordPressにも既存の設定を変更することなく簡単にpjaxを導入することができます。
WordPressの各種プラグインも概ね共存し併用することができます。

※WordPressインストール直後、twentytwelveテーマを使用して確認。

※プラグインはメジャーなものを2,30個インストールして確認、競合は1,2個ほど。

※文字コードの差異は修正する必要があります。

※本稼働中のWordPressサイトでの動作は未確認。現在動作および負荷検証用WordPressサイトを制作中。

初期テーマ（twentytwelve）への導入例

1. WordPressにjQueryとpjaxプラグインをアップロードしてください。ここでは `/lib/` ディレクトリにアップロードしたものとします。
2. テキストエディタで `initialize.js` ファイルを作成し、次のように記述したのちUTF-8で保存し先ほどと同じディレクトリにアップロードします。 `http://host/` の部分はサイトにあわせて変更してください。

```
$(function(){
  $.pjax({
    area: '#primary',
    link: 'a:not([target])[href^="http://host/"]',
    load: { css : true , script : true , sync : true , async : 0 }
  });
});
```

3. テーマ編集画面を開き、 `header.php` の `</head>` タグの直前に次のようにコードを追加してアップロードしたファイルを読み込ませてください。jQueryは外部のサイトから読み込ませてもかまいません。

```
<script type="text/javascript" charset="utf-8" src="/lib/jquery-1.7.2.min.js"></script>
<script type="text/javascript" charset="utf-8" src="/lib/jquery.pjax.min.js"></script>
<script type="text/javascript" charset="utf-8" src="/lib/initialize.js"></script>
</head>
```

以上で終わりです。 `header.php` の編集を保存したらpjaxが動作しているはずです。

競合により不具合のあるWordPressプラグインへの対応

JavaScriptを使用しているWordPressプラグインで不具合が発生する場合がありますが、JavaScriptの実行タイミングを調整することで不具合を解消し共存させられる可能性があります。
不具合が発生する主な状況と対応は以下のとおりです。

WordPressプラグインのJavaScriptの想定外のページでの使用

pjaxではJavaScriptの実行状態がページ移動後も維持されるため、ページ移動により変更されたDOMの差異からエラーが発生する可能性があります。ページ移動時に当該JavaScriptを終了ないし停止させ、適宜再開させることができれば回避が可能です。終了ないし停止ができない場合はあとはWordPressプラグインのJavaScriptの例外処理の問題であるためWordPressプラグインの修正以外による対応は困難です。

WordPressプラグインのJavaScriptを使用するページへの再アクセス

このプラグインは移動先のページのJavaScriptが読み込み済みである場合、コードが外部ファイルに記述されている場合はこれを読み込まず、同一ページに埋め込まれている場合は再度読み込み実行します。このため、併用するJavaScriptによっては正常に動作させるために適宜再実行により実行状態をリセットし、または読み込ませずリセットさせない処理を追加する必要があります。

注意点

リンクパスの記述

pjaxを使用したサイトでは相対パスはルートパスで書くのが基本です。ルートパス以外の相対パスで書いた場合、リンクが意図しないURLを示すことがあるため内部リンクの相対パスは必ずルートパスで書いてください。

`http://host/from/` から `http://host/from/to/` へのリンク

GOOD

BAD

フォームの取り扱い

後述の Google Chrome のバグの影響を回避するため、フォームを通常動作で使用する場合は必ずフォームが更新範囲に含まれないようにしてください。

フォームによりデータの送受信を行うページではpjaxの使用を推奨しません。pjaxによりページ遷移を行った

場合ではフォームへの入力情報が前後へのページ移動のつどリセットされるため、入力情報のリセット対策を施さなければ送信までのコンバージョン率の大幅な低下が予想されること、POST情報の二重送信防止の一般的な手法であるページのリダイレクトを行うために結局は通常のページ遷移が要求されることが理由です。
フォームによるデータの送受信はpjaxではなく通常のページ遷移またはajaxによる入力情報の送信とページ内容の書き換えにより行うことを推奨します。

```
<div class="primary pjax">
↓
<div class="primary">
```

なお、検索フォームのようなGET送信フォームで使用する分には問題ありません。

補足

ドキュメント内の用語の用法にはあまり自信がありません。間違いやバグに気づかれた方は掲示板または連絡フォームからご連絡ください。

ライセンス - MIT License

以下に定める条件に従い、本ソフトウェアおよび関連文書のファイル（以下「ソフトウェア」）の複製を取得するすべての人に対し、ソフトウェアを無制限に扱うことを無償で許可します。これには、ソフトウェアの複製を使用、複写、変更、結合、掲載、頒布、サブライセンス、および/または販売する権利、およびソフトウェアを提供する相手に同じことを許可する権利も無制限に含まれます。
上記の著作権表示および本許諾表示を、ソフトウェアのすべての複製または重要な部分に記載するものとします。
ソフトウェアは「現状のまま」で、明示であるか暗黙であるかを問わず、何らの保証もなく提供されます。ここでいう保証とは、商品性、特定の目的への適合性、および権利非侵害についての保証も含みますが、それに限定されるものではありません。 作者または著作権者は、契約行為、不法行為、またはそれ以外であろうと、ソフトウェアに起因または関連し、あるいはソフトウェアの使用またはその他の扱いによって生じる一切の請求、損害、その他の義務について何らの責任も負わないものとします。
http://opensource.org/licenses/mit-license.php
http://sourceforge.jp/projects/opensource/wiki/licenses%2FMIT_license

jQuery Plugins

pjax

HTML5による高速なページ移動機能をウェブサイトに実装します。

displaytrigger

スクロールにより特定のHTML要素が画面に表示されることを条件としてスクリプトを遅延実行させます。

clientenv

サイトの閲覧者のOS、ブラウザ、フォント対応などを判定してクロスブラウザ対応の労力を軽減します。

validator

JavaScriptの動作検証とエラーレポートを行う、インストール不要の埋め込み型検証ツールです。

spage

AutoPagerやAutoPatchWorkのようなページの自動読み込み＆継ぎ足し機能をウェブサイトに実装します。

pagenation - auto page load by jquery.spage.js -

first prev pjax next last