به نام خدا

مستندات ماژول های جاوااسکریپت

فهرست مطالب

مدیریت فایل ها

وظایف کلی این کتابخانه خواندن، تکه کردن و ارسال فایل به سرور است که به چند مرحله تقسیم میشود. کتابخانه به صورت کاملاً شئ‌گرا نوشته شده و برای کار با هر فایل نیاز به گرفتن نمونه‌ای از شئ است.

نمونه‌گیری

اکثر تنظیمات مربوط به فایل در هنگام نمومه‌گیری تعیین میشوند. در مثال زیر تنظیمات پیش‌فرض فایل‌ها و مقادیر قابل تعیین آن‌ها مشخص شده‌اند:

```
var fm = new FileManager({
    type: 'BinaryString', // 'DataURL', 'Text', 'ArrayBuffer'
    url: 'localhost:8000', // Base URL
    file: input.files[0],
    parts: 200, // KiloByte
    rest: 50,
    restDuration: 500, // Rest for 500 miliseconds
    data: {}, // Data passed to server (e.g. fileID)
});
```

توضیحات تنظیمات

type

تعیین کننده‌ی روش خواندن داده‌های فایل است. (اطلاعات بیشتر MDN - FileReader)

file

شئ فایلی است که از راه Drag n Drop یا ورودی های نوع فایل دریافت شده و مقدار پیشفرض ندارد.

parts

سایز تکه‌های ارسالی در هر پاکت را تعیین میکند که به طور پیشفرض ۲۰۰ کیلوبایت است.

rest

تعداد پاکت های ارسالی بدون استراحت رو تعیین میکنه. (اطلاعات بیشتر توضیحات upload)

restDuration

زمان استراحت مرورگر بعد از ارسال پاکت‌ها به اندازه مقدار rest رو تعیین میکنه. (اطلاعات بیشتر توضیحات upload)

رویداد ها

برای شنود رویدادها، از متد on استفاده کنید، برای مثال:

```
fm.on('upload:progress', function(e, data) {
   console.log(data.percentage);
};
```

رویداد ها به دو دسته تقسیم می شوند: رویداد های مربوط به فایل ها و رویداد های مربوط به آپلود و با استفاده از علامت دو نقطه : دسته بندی می شوند.

upload:start

این رویداد مربوط به شروع آپلود است. در هنگام شروع هر آپلود این رویداد بدون هیچ اطلاعات اضافه اجرا می شود.

upload:progress

بعد از دریافت پاسخ از سمت سرور مبنی بر نوشته شدن فایل (موفقیت آمیز بودن آپلود قطعه)، این رویداد همراه را اطلاعات زیر جرقه می‌خورد.

توضیحات مشخصات:

min

این مقدار بایت اولیه رو مشخص میکنه که به عنوان آرگومان اول به متد upload داده شده است. در حالتی که آپلود از اول شروع شده باشد، این مقدار صفر است. در صورتی که ما در حال ادامه دادن آپلود باشیم، مقدار بستگی به بایت اولیه ما دارد. به مثال زیر توجه کنید:

```
fm.on('upload:progress', function(e, data) {
   console.log(data.min);
});
fm.upload(100); // logs 100
```

max

این مشخصه، آخرین بایت هدف است که به عنوان آرگومان دوم به `upload` داده شده است. در صورتی که این مقدار مشخص نشده باشد، برابر با سایز فایل است.

sent

تعیین کننده آخرین بایت ارسال شده تا اون لحظه است. این به این معنی است که *تعداد* بیت های ارسالی را مشخص نمیکند، به مثال زیر توجه کنید:

```
fm.on('upload:progress', function(e, data) {
    console.log(data.sent);
});
fm.upload(100); // logs 200100 (sent 200 kilobytes = 200000 + 100)
```

percentage

درصد ارسال شده که از فرمول زیر بدست می‌آید. این مقدار به صورت اعشاری است و برای استفاده از مقدار صحیح باید از توابع ریاضی مانند `Math.floor` استفاده کنید.

```
sent * 100 / max
```

متدها

load

این متد برای خواندن فایل استفاده میشود و در آخر یک `promise` بازمیگرداند. مثال استفاده:

```
var fm = new FileManager({ file: input.files[0] });
fm.load().then(function(e) {
    console.log('File contents:', e.target.result);
}, function(err) {
    console.log('Error ocurred:', err);
});
```

slice

این متد برای تکه کردن فایل استفاده میشود و به صورت زنجیره‌ای عمل میکند. تغییرات اعمال شده توسط این متد تا زمانی که با استفاده از متد `full` از بین نروند باقی خواهند ماند. برای مثال اگر بخواهیم فقط ۱۰۰ بایت اول یک فایل را بخوانیم:

```
fm.slice(0, 100).load();
fm.slice(200, 500);
fm.load()
```

full

این متد فایل تکه شده رو به حالت اولیه برمیگردونه یعنی عمل slice رو خنثی میکند و به صورت زنجیره ای استفاده می شود. اجرای این متد برابر اجرای کد زیر است:

```
fm.slice(0, fm.size);
```

createStream

این متد برای ایجاد یک ارتباط برای آپلود به سرور اجرا می شود. روش کار این متد به این شکل است که اطلاعات فایل موجود (نام و سایز) و data به سرور با عنوان meta ارسال می شود. سپس منتظر پاسخی با عنوان ready می‌شود و با دریافت پاسخ، مشخصه‌ی fileID و data.fileID خود را برابر آی دی دریافتی از سرور میکند. (سرور باید رکوردی برای اطلاعات داده شده بسازد/پیدا کند و آی دی را برای کاربر بفرستد).

```
var meta = _.extend({
   file: this.originalFile.name,
   size: this.size,
}, this.data);

socket.emit('meta', meta);
```

send

این متد برای ارسال تکه فایل موجود استفاده می‌شود. روش کار این متد به این شکل است که پیغامی با عنوان data از طریق سوکت ارسال شده و هنگام دریافت پاسخ با عنوان data-answer کار خود را تمام شده اعلام میکند. این متد یک Promise باز میگرداند. در مثال زیر قطعه ۲۹۹ کیلوبایتی بین ۲۰۰ تا ۵۰۰ ارسال می شود.

```
fm.slice(200, 500).send().then(function() {
// Do something
}, function() {
// Error
});
```

upload

این متد از تمامی متد های بالا استفاده میکند تا کار تکه تکه کردن فایل، خواندن هر تکه و ارسال آن تکه به سرور را انجام دهد. این متد سه ورودی اختیاری دارد که تعیین کننده بایت شروع، پایان و تنظیمات هستند. این

متد یک promise باز میگرداند. ابتدا به مثال استفاده از آن دقت کنید.

```
fm.upload(50, 1000);
```

آپلود به صورت پشت‌سر هم (نه همزمان) انجام می‌شوند، به طوری که در هر مرحله، فایل به قسمت‌هایی به اندازه مقدار parts تقسیم شده (توسط متد slice) و سپس ارسال می‌شوند (توسط متد send). پس از ارسال تعدادی پاکت (به اندازه مقدار rest)، متد به اندازه restDuration به مرورگر استراحت می‌دهد و سپس به آپلود ادامه می‌دهد.

pause

این متد برای متوقف سازی عملیات آپلود استفاده می شود.

resume

این متد برای ادامه دادن به آپلود یک فایل استفاده می شود. روش کار این متد به این شکل است که پیغامی با عنوان resume-status و داده‌های موجود در مشخصه data ارسال می‌کند. سرور باید در پاسخ حجم فایل را در سرور پاسخ دهد. سپس آپلود از بایت مشخص شده از سرگیری می شود. (اینجا باید fileID در مشخصه data مشخص شده باشد تا سرور بتواند تشخیص دهند کدام فایل هدف ما است.)

کلیک‌راست

این کتاب‌خانه برای شخصی سازی کلیک راست استفاده میشود و به صورت یک پلاگین jQuery کار میکند. این کتابخانه پس از نمونه‌گیری، شئ با امکان کنترل آیتم ها به شما میدهد.

نمونه گیری

برای ساخت نمونه، مثل اکثر پلاگین‌های jQuery لازمه که عناصر مورد نظر که نیاز به شخصی سازی کلیک راست دارند رو انتخاب کنیم و سپس تنظیمات رو [در صورت نیاز] مشخص کنیم.

```
$(document).ctxMenu({
   item: { // default item properties
     link: false,
     text: 'contextmenu item',
     events: {
       click: function(e) {
         alert('Test')
       }
     }
   },
    items: [
         {text: 'Item 1'}
    ]
});
```

توضیحات تنظیمات

تنظیمات این شئ در حالت عادی محدود هستند ولی شما اجازه تغییر در روند کاری رو دارید که بعدا توضیح داده میشود.

`item`

برای تعیین مشخصات پیشفرض آیتم‌ها از این مشخصه استفاده میشود. در مثال بالا مشخصاتی که به صورت آماده پشتیبانی میشوند را نشون داده‌ایم.

`link`

در صورتی که این مشخصه رو مقدار دهی کنید، لینکی در آیتم قرار میگیره با آدرس داده شده.

text

این مشخصه هم متن آیتم رو مشخص میکنه.

events

این مشخصه یک شئ دربر گیرنده رویداد های آیتم است که به صورت `delegate` شده هستند (اطلاعات بیشتر روند گردش رویداد ها، `Understanding Event Delegation`)

items

آرایه ای از اشیا که در اصل همان آیتم ها هستند. برای اضافه و حذف آیتم‌ها بعد از نمونه‌گیری متد های `add` و `remove` وجود دارند.

متدها

init

در اصل تمام کار های `constructor` در این متد انجام می‌شوند. دلیل این موضوع نیاز به دوباره‌سازی شئ بعد از برخی اعمال مثل تغییر متد `createWrapper` است. در صورتی که این متد رو دستی اجرا کنید، به طور خودکار عنصر قبلی حذف شده و عنصر جدید ساخته می‌شود تا از روی‌هم جمع شدن عناصر بی استفاده جلوگیری شود.

add

متدی برای اضافه کردن آیتم به منوی کلیک راست. این متد اندیس آیتم اضافه شده را به بازمیگرداند که برای حذف آیتم از ان استفاده میشود. نمونه استفاده:

```
var ctx = $(document).ctxMenu();
ctx.add({
  text: 'سلام'
  });
```

remove

این متد برای حذف آیتمی از منو استفاده میشود به صورتی که اندیس آیتم مورد نظر رو دریافت کرده و بعد از حذف آیتم، طول آرایه‌ی آیتم‌ها را بازمیگرداند.

createItems

این متد استفاده‌ی داخلی دارد و برای ساخت عناصر HTML آیتم ها از این متد استفاده می‌شود. در حالت پیش‌فرض این متد تنها متد محافظت شده‌ی `_createItems` را اجرا می‌کند. نحوه کار پیش‌فرض در صفحه بعد:

```
var $stack = $('<ul></ul>');
for(var i = 0, len = this.items.length; i < len; i++) {
    var item = _.extend(defaults.item, this.items[i]);
    var $el = $('<li data-id="' + i + '"><a ' +
                (item.link ? 'href="' + item.link + '"' : '') + '>' +
                item.text + '</a></li>');
    $stack = $stack.append($el);
}

this.$wrapper.empty().append($stack);
return $stack;
```

createWrapper

این متد برای ساخت عنصر دربر گیرنده یا والد آیتم ها به کارمیرود که در حالت پیشفرض متد محافظت‌شده `_createWrapper` را استفاده میکند. دقت کنید بعد از تعریف دوباره این متد لازم است متد `init` را دوباره اجرا کنید تا عنصر جدید جایگزین قبلی شود. روش کار پیشفرض: (اینجا متغیر `id` به صورت مخفی در `closure` تعریف شده)

```
var $wrapper = $('<div class="ctx-menu" id="' + (id++) + '" data-modal></div>');
    $wrapper.css({
    display: 'none',
    position: 'absolute'
});
$wrapper.find('ul').css({
    'list-style': 'none',
    margin: '0'
});
return $wrapper;
```

درخواست های ایژکس - ارسال فرم‌ها

برای ارسال فرم‌ها و یا ارسال درخواست های عملیاتی به سرور با ajax پلاگین jQuery نوشته شده که این کار رو انجام میده و سعی شده حداکثر امکان شخصی سازی فراهم بشه. وظیفه‌ی این ماژول خواندن اطلاعات فرم (با استفاده از FormData)، ارسال اطلاعات به سرور، دریافت پاسخ و در انتها نمایش نتیجه‌ی عملیات در سرور، و در صورت نیاز اجرای توابع خواص. این ماژول به صورت یک مدل / شئ نیست و استفاده از آن یک‌باره است.

روش استفاده

برای استفاده از این ماژول، ابتدا با استفاده از jQuery عناصر مورد نظر (فرم‌ها، لینک‌ها) را انتخاب میکنیم و با استفاده از متد ajaxify، پلاگین ما، و تنظیمات زیاد کار را به پلاگین میسپاریم. نمونه استفاده (تنظیمات پیشفرض):

```
$('form').ajaxify({
    ajax: {
      type: 'post',
      url: '',
      processData: false,
      contentType: false,
      dataType: 'json',
      cache: false
    },
    link: false,
    event: false,
    immediate: false,
    noLoading: false,

    createElement: _.noop
});
```

مشخصه‌ی ajax، شئ داده شده به jQuery.ajax است که البته در آخر مشخصه‌ی data_ـه اون با مقدار گرفته شده از FormData پر میشود.

دقت کنید که درصورتی که فرم انتخاب کرده باشید برای متد ارسال درخواست http از مشخصه‌ی method عنصر و برای url از مشخصه‌ی action استفاده می‌شود.

اگر بخواهید لینک (a) را به صورت ایژکس اجرا کنید، لازم است مشخصه‌ی link را با مقدار true پر کنید. در این حالت آدرس درخواست از مشخصه‌ی href لینک برداشته خواهد شد و متد ارسالی به صورت پیشفرض به

get تغییر خواهد کرد. همچنین در این حالت هیچ داده‌ای به سرور ارسال نخواهد شد (مثلا FormData).

مشخصه ی event برای شخصی سازی رویدادی‌است که ارسال درخواست با جرقه خوردن آن اجرا خواهد شد، به صورت پیشفرض، در حالتی که مقدار link برابر false باشد، رویداد submit و در غیر این صورت رویداد click می باشد.

مشخصه ی immediate برای زمانی است که شما قصد شنود رویدادی را ندارید و میخواهید درخواست در زمان اجرای تابع ajaxify ارسال شده و پس از آن بر روی رویدادی شنود نکند.

در صورتی که تابع createElement مقدار دهی شده باشد، از آن برای ساخت عنصر مورد نظر برای نمایش نتایج درخواست استفاده می‌شود.

رویداد ها

برای کنترل بهتر روند اجرایی فرم می‌توان رویداد های متعددی تعریف کرد. برای شنود روی رویداد ها از متد on استفاده کنید. قبل از نام هر رویداد باید ajaxify: قرار دهید. دقت کنید که‌آرگومان های داده شده رویداد همان‌آرگومان های دریافتی از jQuery بوده، با استثنای آرگومان اول که یک شی رویدادی jQuery بوده و استفاده از آن دلیلی ندارد.

```
$('#myForm').on('ajaxify:success', function(e, data, status, xhr) {
    // Do something
});
```

before

این رویداد قبل از انجام هرکاری اجرا می‌شود (لحظه‌ی رخداد رویداد submit).

success

این رویداد در صورت گرفتن پاسخ صحیح (۲۰۰ - ۳۰۰) از سرور اجرا می‌شود.

complete

این رویداد در صورت گرفتن هرگونه پاسخ از سمت سرور اجرا خواهد شد.

fail

این رویداد در صورت گرفتن پاسخ بد از سرور (۴۰۰ - ۵۰۰) از سرور اجرا خواهد شد.

توضیحات بیشتر

برای فهم بهتر روند اجرایی رویدادهای ماژول به مثال‌های زیر توجه کنید:

در صورتی که پاسخ صحیح از سمت سرور دریافت کنیم (۲۰۰ - ۳۰۰) رویدادها به شکل زیر اجرا خواهند شد:

```
before > success && complete
```

در صورتی که پاسخ صحیح از سرور دریافت نشود (۴۰۰ - ۵۰۰) رویداد ها به شکل زیر اجرا خواهند شد:

```
before > fail && complete
```

جابجایی صفحات و تاریخ مرورگر

وظیفه‌ی این ماژول کار با History API و جابجای بین صفحات بدون نیاز به بازنشانی (Refresh) است

روش استفاده

برای جابجایی بین صفحه‌ها، از تابع Navigate استفاده میکنیم. تنظیمات این تابع عبارتند از مقادیر زیر:

```
Navigate({
    html: '',
     title: null,
     url: '/',
     replace: false,
    filter: null,
    fake: false,
     ajax: {
       type: 'get'
     }
});
```

توضیحات تنظیمات

url

این مقدار مشخص کننده آدرس صفحه بعد از جابجای است. همچنین در صورتی که مقدار html مشخص نشده باشد، درخواست ایژکسی به این آدرس ارسال خواهد شد و از نتیجه‌ی آن به عنوان html استفاده خواهد شد.

html

در صورتی که مقدار HTML مورد نظر برای جابجایی را از قبل دارید، از این مشخصه استفاده کنید. این مشخصه اختیاری است. برای توضیحات بیشتر به مشخصه‌ی url رجوع کنید.

title

این مشخصه تعیین کننده تیتر صفحه‌ی مورد نظر است. در صورتی که این مقدار مشخص نشود، دو حالت وجود دارد: ۱. در صورتی که HTML دارای تگ title در head باشد، جایگزین خواهد شد و به درستی نمایش داده خواهد شد. ۲. در صورتی که هیچ تیتری تعریف نشده باشن، تیتر صفحه‌ی قبل باقی خواهد ماند.

replace

در صورتی که بخواهید بجای اضافه شدن یک رکورد به تاریخچه‌ی صفحه، رکورد حاظر بازنویسی شود، از این مشخصه استفاده کنید. (اطلاعات بیشتر replaceState)

fake

این مشخصه برای تغییر `url` بدون ایجاد هیچ تغییر دیگه در صفحه استفاده میشود.

ajax

این شی همراه با تنظیمات تعیین شده توسط تابع `Navigate` به `jQuery.ajax` داده می شوند. می توانید از این شی برای ارسال اطلاعات اضافه استفاده کنید. مثلا برای ارسال هدر های خاص:

```
Navigate({
    url: '/test',
    ajax: {
          headers: {
                'Test': 'MyValue'
          }
    }
});
```

filter

این مشخصه برای تعیین فیلتری برای جایگزینی عناصر است. برای اطلاعات بیشتر در مورد جایگزینی عناصر به بخش مربوطه مراجعه کنید. مقدار این مشخصه میتواند آرایه یا یک رشته باشد که مشخص کننده مقدار `data-xhr` است. برای مثال:

```
Navigate({
    filter: ['salam', 'test3'] // => [data-xhr="salam"], [data-xhr="test3"]
});
```

رویداد ها

این تابع تنها یک رویداد دارد که در پایان انجام عملیات های مربوطه انجام می شود. برای شنود این رویداد از کد زیر استفاده کنید:

```
$(window).on('statechange', function() {
    console.log(history.state);
});
```

این رویداد در صورت هرگونه تغییر روی `history.state` جرقه میخورد. یعنی در صورتی که کاربر بر روی دکمه های `Back/Forward` مرورگر کلیک کند نیز این رویداد اجرا خواهد شد.

جایگزینی عناصر مشخص شده

یکی از امکاناتی که این ماژول در اختیار قرار می‌دهد امکان آپدیت صفحه به صورت قسمت به قسمت است. نحوه‌ی کار به این شکل است که در HTML داده شه به ماژول (چه دریافت شده از طریق ajax و چه ورودی دستی) به دنبال عناصری با مشخصه‌ی data-xhr میگردیم، و به ازای هریک از آنها، عنصری با مقدار مشابه را پیدا کرده و با عنصر داده شده جایگزین میکنیم.

فرض کنید HTML صفحه‌ای که در اون هستیم به این شکل باشه:

```
<body>
    <div data-xhr="salam">
          Hello
    </div>
    <div data-xhr="test">
          Test2
    </div>
</body>
```

حال تابع Navigate را با تنظیمات زیر اجرا میکنیم:

```
Navigate({
    url: '/test',
    html: '<div data-xhr="salam">Yo</div>'
});
```

در این صورت، بعد از اجرای کامل تابع HTML صفحه ما به شکل زیر خواهد بود:

```
<body>
    <div data-xhr="salam">Yo</div>
    <div data-xhr="test">
          Test2
    </div>
</body>
```

برای تعیین فیلتری برای جایگزینی، میتوانید از مشخصه‌ی filter استفاده کنید.