عنوان: **افزونه نویسی برای مرورگرها : فایرفاکس : قسمت اول**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۱/۰۵ ۲۲:۵۰
آدرس: www.dotnettips.info
گروه‌ها: Firefox, Plugin, Extension, Addon

در دو مقاله پیشین [^] ، [^] به بررسی نوشتن افزونه در مرورگر کروم پرداختیم و اینبار قصد داریم همان پروژه را برای فایرفاکس پیاده کنیم. پس در مورد کدهای تکراری توضیحی داده نخواهد شد و برای فهم آن می‌توانید به دو مقاله قبلی رجوع کنید. همه‌ی ما فایرفاکس را به خوبی می‌شناسیم. اولین باری که این مرورگر آمد سرو صدای زیادی به پا کرد و بازار وسیعی از مرورگرها را که در چنگ IE بود، به دست آورد . این سر و صدا بیشتر به خاطر امنیت و کارآیی بالای این مرورگر، استفاده از آخرین فناوری‌های تحت وب و دوست داشتنی برای طراحان وب بود. همچنین یکی دیگر از مهمترین ویژگی‌های آن، امکان سفارشی سازی آن با افزونه‌ها extensions یا addon بود که این ویژگی در طول این سال‌ها تغییرات زیادی به خود دیده است. در مورد افزونه نویسی برای فایرفاکس در سطح نت مطالب زیادی وجود دارند که همین پیشرفت‌های اخیر در مورد افزونه‌ها باعث شده خیلی از این مطالب به روز نباشند. اگر در مقاله پیشین فکر می‌کنید که کروم چقدر در نوشتن افزونه جذابیت دارد و امکانات خوبی را در اختیار شما می‌گذارد، الان دیگر وقت آن است که نظر خودتان را عوض کنید و فایرفاکس را نه تنها یک سرو گردن بلکه بیشتر از این حرف‌ها بالاتر بدانید.
شرکت موزیالا برای قدرتمندی و راحتی کار طراحان یک sdk طراحی کرده است است و شما با استفاده از کدهای موجود در این sdk قادرید کارهای زیادی را انجام دهید. برای نصب این sdk باید پیش نیازهایی بر روی سیستم شما نصب باشد:
نصب پایتون 2.5 یا 2.6 یا 2.7 که فعلا در سایت آن، نسخه‌ی 2.7 در دسترس هست. توجه داشته باشید که هنوز برای نسخه‌ی 3 پایتون پشتیبانی صورت نگرفته است.

آخرین نسخه‌ی sdk را هم می‌توانید از این آدرس به صورت zip و یا از این آدرس به صورت tar دانلود کنید و در صورتیکه دوست دارید به سورس آن دسترسی داشته باشید یا اینکه از سورس‌های مشارکت شده یا غیر رسمی استفاده کنید، از این صفحه آن را دریافت کنید.

بعد از دانلود sdk به شاخه‌ی bin رفته و فایل activate.bat را اجرا کنید. موقعی که فایل activate اجرا شود، باید چنین چیزی دیده شود:

```
(C:\Users\aym\Downloads\addon-sdk-1.17) C:\Users\aym\Downloads\addon-sdk-1.17\bin>
```

برای سیستم‌های عامل Linux,FreeBSD,OS X دستورات زیر را وارد کنید:
اگر یک کاربر پوسته‌ی bash هستید کلمه زیر را در کنسول برای اجرای activate بزنید:

```
source bin/activate
```

اگر کاربر پوسته‌ی بش نیستید:

```
bash bin/activate
```

نهایتا باید کنسول به شکل زیر در آید یا شبیه آن:

```
(addon-sdk)~/mozilla/addon-sdk >
```

بعد از اینکه به کنسول آن وارد شدید، کلمه cfx را در آن تایپ کنید تا راهنمای دستورات و سوییچ‌های آن‌ها نمایش داده شوند. از این ابزار میتوان برای راه اندازی فایرفاکس و اجرای افزونه بر روی آن، پکیج کردن افزونه، دیدن مستندات و آزمون‌های واحد استفاده کرد.

آغاز به کار

برای شروع، فایل‌های زیادی باید ساخته شوند، ولی نگران نباشید cfx این کار را برای شما خواهد کرد. دستورات زیر را جهت

ساخت یک پروژه خالی اجرا کنید:

```
mkdir fxaddon
cd fxaddon
cfx init
```

یک پوشه را در مسیری که کنسول بالا اشاره میکرد، ساختم و وارد آن شدم و با دستور cfx init دستور ساخت یک پروژه‌ی خالی را دادم و باید بعد از این دستور، یک خروجی مشابه زیر نشان بدهد:

```
* lib directory created
* data directory created
* test directory created
* doc directory created
* README.md written
* package.json written
* test/test-main.js written
* lib/main.js written
* doc/main.md written
Your sample add-on is now ready for testing:
try "cfx test" and then "cfx run". Have fun!"
```

در این پوشه یک فایل به اسم package.json هم وجود دارد که اطلاعات زیر داخلش هست:

```
{
  "name": "fxaddon",
  "title": "fxaddon",
  "id": "jid1-QfyqpNby9lTlcQ",
  "description": "a basic add-on",
  "author": "",
  "license": "MPL 2.0",
  "version": "0.1"
}
```

این اطلاعات شامل نام و عنوان افزونه، توضیحی کوتاه در مورد آن، نویسنده‌ی افزونه، ورژن افزونه و ... است. این فایل دقیقا معادل manifest.json در کروم است. در افزونه نویسی‌های قدیم این فایل install.rdf نام داشت و بر پایه‌ی فرمت rdf بود. ولی در حال حاضر با تغییرات زیادی که افزونه نویسی در فایرفاکس کرده‌است، الان این فایل بر پایه یا فرمت json است. اطلاعات package را به شرح زیر تغییر می‌دهیم:

```
{
  "name": "dotnettips",
  "title": ".net Tips Updater",
  "id": "jid1-QfyqpNby9lTlcQ",
  "description": "This extension keeps you updated on current activities on dotnettips.info",
  "author": "yeganehaym@gmail.com",
  "license": "MPL 2.0",
  "version": "0.1"
}
```

رابطهای کاربری Action Button و Toggle Button فایل main.js را در دایرکتوری lib باز کنید:
موقعی که در کروم افزونه می‌نوشتیم امکانی به اسم browser action داشتیم که در اینجا با نام action button شناخته می‌شود. در اینجا باید کدها را require کرد، همان کاری در خیلی از زبان‌ها مثلا مثل سی برای صدا زدن سرآیندها می‌کنید. مثلا برای action button اینگونه است:

```
var button= require('sdk/ui/button/action');
```

نحوه‌ی استفاده هم بدین صورت است:

```
buttons.ActionButton({...});
```

که در بین {} خصوصیات دکمه‌ی مورد نظر نوشته می‌شود. ولی من بیشتر دوست دارم از شیء دیگری استفاده کنم. به همین جهت ما از یک مدل دیگر button که به اسم toggle button شناخته می‌شود، استفاده می‌کنیم. از آن جا که این button دارای دو حالت انتخاب (حالت فشرده شده) و غیر انتخاب (معمولی و آماده فشرده شدن توسط کلیک کاربر) است، بهترین انتخاب هست.

کد زیر یک toggle button را برای فایرفاکس می‌سازد که با کلیک بر روی آن، صفحه‌ی popup.htm به عنوان یک پنل روی آن رندر می‌شود:

```
var tgbutton = require('sdk/ui/button/toggle');
var panels = require("sdk/panel");
var self = require("sdk/self");

var button = tgbutton.ToggleButton({
  id: "updaterui",
  label: ".Net Updater",
  icon: {
    "16": "./icon-16.png",
    "32": "./icon-32.png",
    "64": "./icon-64.png"
  },
  onChange: handleChange
});

var panel = panels.Panel({
  contentURL: self.data.url("./popup.html"),
  onHide: handleHide
});

function handleChange(state) {
  if (state.checked) {
    panel.show({
      position: button
    });
  }
}

function handleHide() {
  button.state('window', {checked: false});
}
```

در سه خط اول، فایل‌هایی را که نیاز است Required شوند، می‌نویسیم و در یک متغیر ذخیره می‌کنیم. اگر در متغیر نریزیم مجبور هستیم همیشه هر کدی را به جای نوشتن عبارت زیر:

```
tgbutton.ToggleButton
```

به صورت زیر بنویسیم:

```
require('sdk/ui/button/toggle').ToggleButton
```

که اصلا کار جالبی نیست. اگر مسیرهای نوشته شده را از مبدا فایل zip که اکسترکت کرده‌اید، در دایرکتوری sdk در شاخه lib بررسی کنید، با دیگر موجودیت‌های sdk آشنا خواهید شد.

در خط بعدی به تعریف یک شیء از نوع toggle button به اسم button میپردازیم و خصوصیاتی که به این دکمه داده ایم، مانند یک کد شناسایی، یک برچسب که به عنوان tooltip نمایش داده خواهد شد و آیکن‌هایی در اندازه‌های مختلف که در هرجایی کاربر آن دکمه را قرار داد، در اندازه‌ی مناسب باشد و نهایتا به تعریف یک رویداد می‌پردازیم. تابع handlechange زمانی صدا زده می‌شود که در وضعیت دکمه‌ی ایجاد شده تغییری حاصل شود. در خط بعدی شیء panel را به صورت global میسازیم. شیء self دسترسی ما را به اجزا یا فایل‌های افزونه خودمان فراهم می‌کند که در اینجا دسترسی ما به فایل html در شاخه‌ی data میسر شده است و مقدار مورد نظر را در contentURL قرار می‌دهد. نهایتا هم برای رویداد onhide تابعی را در نظر می‌گیریم تا موقعی که پنجره بسته شد بتوانیم وضعیت toggle button را به حالت قبلی بازگردانیم و حالت فشرده نباشد. چرا که این دکمه تنها با کلیک ماوس به حالت فشرده و حالت معمولی سوییچ میکند. پس اگر کاربر با کلیک بر روی صفحه‌ی مرورگر پنجره را ببندد، دکمه در همان وضعیت فشرده باقی می‌ماند.

همانطور که گفتیم تابع handlechnage موقعی رخ میدهد که در وضعیت دکمه، تغییری رخ دهد و نمیدانیم که این وضعیت فشرده شدن دکمه هست یا از حالت فشرده خارج شده است. پس با استفاده از ویژگی checked بررسی میکنم که آیا دکمه‌ای فشرده شده یا خیر؛ اگر برابر true بود یعنی کاربر روی دکمه، کلیک کرده و دکمه به حالت فشرده رفته، پس ما هم پنل را به آن نشان می‌دهیم و خصوصیات دلخواهی را برای مشخص کردن وضعیت پنل نمایشی به آن پاس می‌کنیم. خصوصیت یا پارامترهای زیادی را می‌توان در حین ساخت پنل برای آن ارسال کرد. با استفاده از خصوصیت position محل نمایش پنجره را مشخص می‌کنیم. در صورتی که ذکر نشود پنجره در وسط مرورگر ظاهر خواهد شد.

تابع onhide زمانی رخ میدهد که به هر دلیلی پنجره بسته شده باشد که در بالا یک نمونه‌ی آن را عرض کردیم. ولی اتفاقی که می‌افتد، وضعیت تابع را با متد state تغییر می‌دهیم و خصوصیت checked آن را false می‌کنیم. بجای پارامتر اولی، دو گزینه را میتوان نوشت؛ یکی window و دیگری tab است. اگر شما گزینه tab را جایگزین کنید، اگر در یک تب دکمه به حالت فشرده برود و به تب دیگر بروید و باعث بسته شدن پنجره بشوید، دکمه تنها در تبی که فعال است به حالت قبلی باز می‌گردد و تب اولی همچنان حالت خود را حفظ خواهد کرد پس می‌نویسیم window تا این عمل در کل پنجره اعمال شود.

Context Menus

برای ساخت منوی کانتکست از کد زیر استفاده می‌کنیم:

```
var contextMenu = require("sdk/context-menu");

var home = contextMenu.Item({
  label: "صفحه اصلی",
  data: "http://www.dotnettips.info/"
});
var postsarchive = contextMenu.Item({
  label: "مطالب سایت",
  data: "http://www.dotnettips.info/postsarchive"
});

var menuItem = contextMenu.Menu({
  label: "Open .Net Tips",
  context: contextMenu.PageContext(),
   items: [home, postsarchive],
  image: self.data.url("icon-16.png"),
  contentScript: 'self.on("click", function (node, data) {' +
                 '  window.location.href = data;' +
                 '});'
});
```

این منو هم مثل کروم دو زیر منو دارد که یکی برای باز کردن صفحه‌ی اصلی و دیگری برای باز کردن صفحه‌ی مطالب است. هر کدام یک برچسب برای نمایش متن دارند و یکی هم دیتا که برای نگهداری آدرس است. در خط بعدی منوی پدر یا والد ساخته می‌شود که با خصوصیت items، زیر منوهایش را اضافه می‌کنیم و با خصوصیت image، تصویری را در پوشه‌ی دیتا به آن معرفی می‌کنیم که اندازه‌ی آن 16 پیکسل است و دومی هم خصوصیت context است که مشخص می‌کند این گزینه در چه مواردی بر روی context menu نمایش داده شود. الان روی همه چیزی نمایش داده می‌شود. اگر گزینه، SelectionContext باشد، موقعی که متنی انتخاب شده باشد، نمایش می‌یابد. اگر SelectorContext باشد، خود شما مشخص می‌کنید بر روی چه مواردی نمایش یابد؛ مثلا عکس یا تگ p یا هر چیز دیگری، کد زیر باعث می‌شود فقط روی عکس نمایش یابد:

```
SelectorContext("img")
```

کد زیر هم روی عکس و هم روی لینکی که href داشته باشد:

```
SelectorContext("img,a[href]")
```

موارد دیگری هم وجود دارند که میتوانید مطالب بیشتری را در مورد آن‌ها در اینجا مطالعه کنید. آخرین خصوصیت باقی مانده، content script است که می‌توانید با استفاده از جاوااسکریپت برای آن کد بنویسید. موقعی که برای آن رویداد کلیک رخ داد، مشخص شود تابعی را صدا میزند با دو آرگومان؛ گره ای که انتخاب شده و داده‌ای که به همراه دارد که آدرس سایت است و آن را در نوار آدرس درج می‌کند.

آن منوهایی که با متد item ایجاد شده‌اند منوهایی هستند که با کلیک کاربر اجرا می‌شوند؛ ولی والدی که با متد menu ایجاد شده است، برای منویی است که زیر منو دارد و خودش لزومی به اجرای کد ندارد. پس اگر منویی میسازید که زیرمنو ندارد و خودش قرار است کاری را انجام دهد، به صورت همان item بنویسید که پایین‌تر نمونه‌ی آن را خواهید دید.

الان مشکلی که ایجاد می‌شود این است که موقعی که سایت را باز می‌کند، در همان تبی رخ می‌دهد که فعال است و اگر کاربر بر روی صفحه‌ی خاصی باشد، آن صفحه به سمت سایت مقصد رفته و سایت فعلی از دست میرود. روش صحیح‌تر اینست که تبی جدید بار شود و آدرس مقصد در آن نمایش یابد. پس باید از روشی استفاده کنیم که رویداد کلیک توسط کد خود افزونه مدیریت شود، تا با استفاده از شیء tab، یک تب جدید با آدرسی جدید ایجاد کنیم. پس کد را با کمی تغییر می‌نویسیم:

```
var tabs = require("sdk/tabs");
var menuItem = contextMenu.Menu({
  label: "Open .Net Tips",
  context: contextMenu.PageContext(),
   items: [home, postsarchive],
  image: self.data.url("icon-16.png"),
  contentScript: 'self.on("click", function (node, data) {' +
                 '  self.postMessage(data);' +
                 '});',
 onMessage: function (data) {
     tabs.open(data);
  }
});
```

با استفاده از postmessage، هر پارامتری را که بخواهیم ارسال می‌کنیم و بعد با استفاده از رویداد onMessage، داده‌ها را خوانده و کد خود را روی آن‌ها اجرا می‌کنیم.

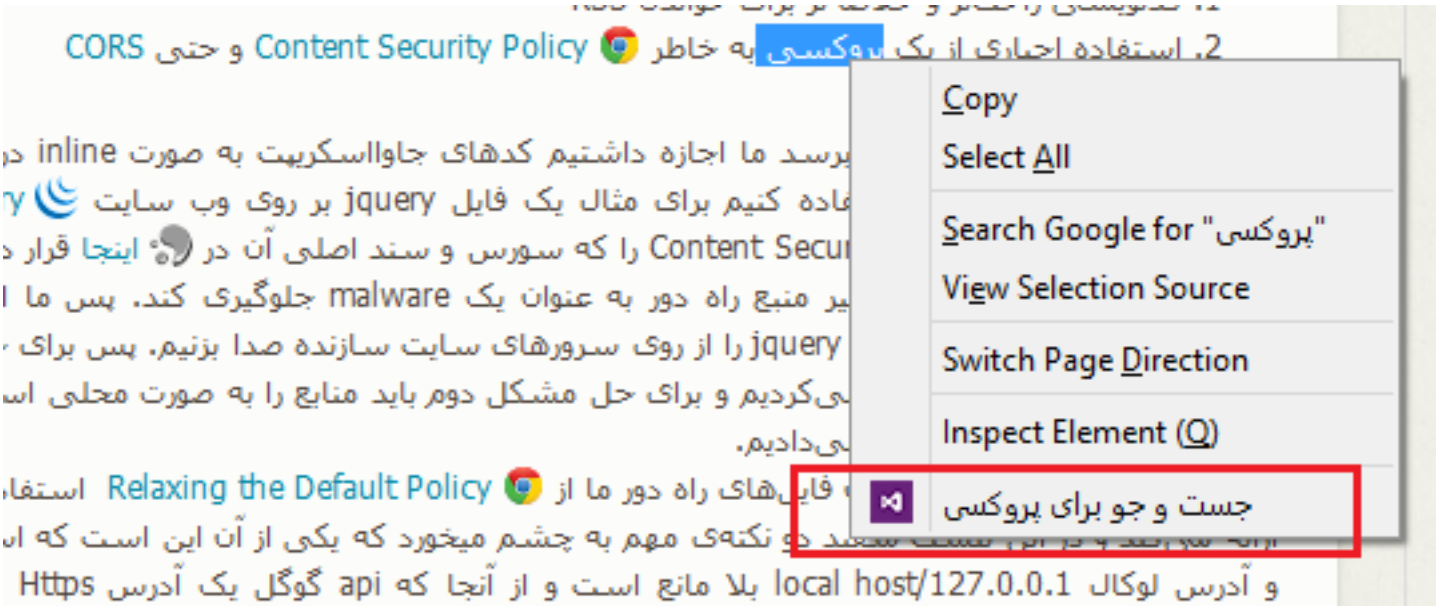
بگذارید کد زیر را هم جهت سرچ مطالب بر روی سایت پیاده کنیم:

```
var Url="http://www.dotnettips.info/search?term=";
var searchMenu = contextMenu.Item({
  label: "search for",
  context: [contextMenu.PredicateContext(checkText),contextMenu.SelectionContext()],
    image: self.data.url("icon-16.png"),
  contentScript: 'self.on("click", function () {' +
  '  var text = window.getSelection().toString();' +
                 '  if (text.length > 20)' +
                 '   text = text.substr(0, 20);' +
                 '  self.postMessage(text);'+
                '})',
onMessage: function (data) {
```

```
    tabs.open(Url+data);
  }

});

function checkText(data) {

       if(data.selectionText === null)
           return false;

       console.log('selectionText: ' + data.selectionText);

       //handle showing or hiding of menu items based on the text content.
       menuItemToggle(data.selectionText);

       return true;
};

function menuItemToggle(text){
var searchText="جست و جو برای ";
    searchMenu.label=searchText+text;

};
```

در ساخت این منو، ما از ContextSelection استفاده کرده‌ایم. بدین معنی که موقعی که چیزی روی صفحه انتخاب شد، این منو ظاهر شود و گزینه‌ی دیگری که در کنارش هست، گزینه contextMenu.PredicateContext وظیفه دارد تابعی که به عنوان آرگومان به آن دادیم را موقعی که منو کانتکست ایجاد شد، صدا بزند و اینگونه میتوانیم بر حسب اطلاعات کانتکست، منوی خود را ویرایش کنیم. مثلا من دوست دارم موقعی که متنی انتخاب می‌شود و راست کلیک می‌کنم گزینه‌ی "جست و جو برای..." نمایش داده شود و به جای ... کلمه‌ی انتخاب شده نمایش یابد. به شکل زیر دقت کنید. این چیزی است که ما قرار است ایجاد کنیم:

در کل موقع ایجاد منو تابع checkText اجرا شده و متن انتخابی را خوانده به عنوان یک آرگومان برای تابع menuItemToggle ارسال می‌کند و به رشته "جست و جو برای" می‌چسباند. در خود پارامترهای آیتم اصلی، گزینه content scrip، با استفاده از جاوااسکریپت، متن انتخاب شده را دریافت کرده و با استفاده از متد postmessage برای تابع onMessage ارسال کرده و با ساخت یک تب و چسباندن عبارت به آدرس جست و جو سایت، کاربر را به صفحه مورد نظر هدایت کرده و عمل جست و جو در سایت انجام می‌گیرد.



در قسمت آینده موارد بیشتری را در مورد افزونه نویسی در فایرفاکس بررسی خواهیم کرد و افزونه را تکمیل خواهیم کرد