عنوان: **نوشتن افزونه برای مرورگرها: قسمت اول : کروم**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۱/۰۳ ۱:۴۰
آدرس: www.dotnettips.info
گروه‌ها: Plugin, Google Chrome, Extension, Addon

افزونه چیست؟

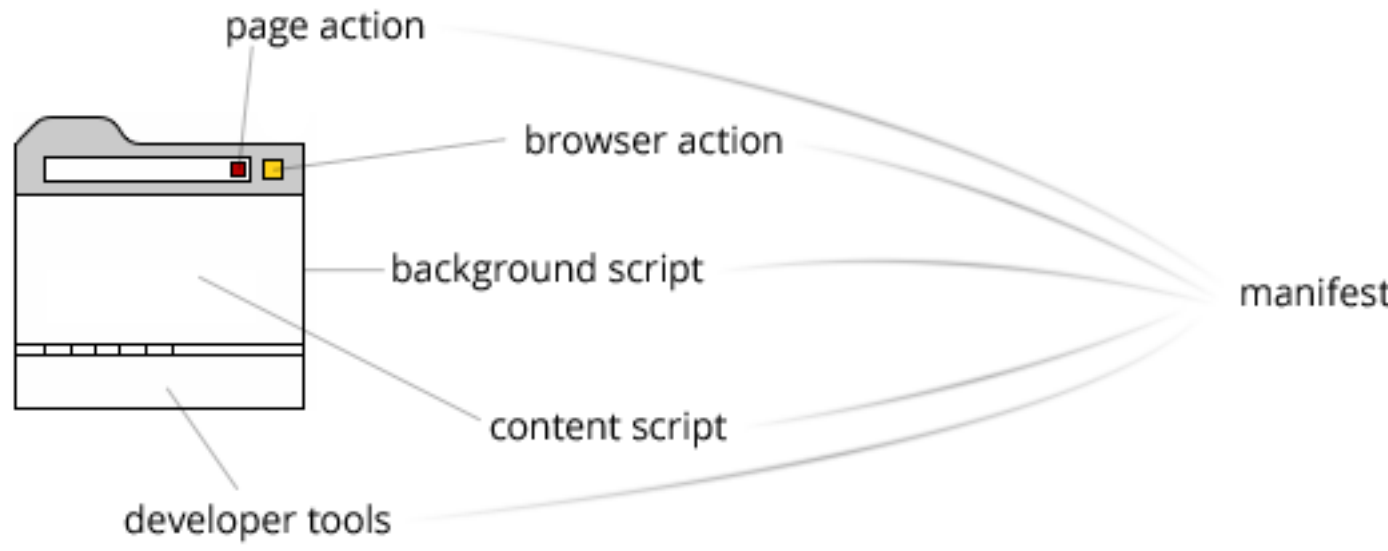
افزونه‌ها جزء مهمترین قسمت‌های یک مرورگر توسعه پذیر به شمار می‌آیند. افزونه‌ها سعی دارند تا قابلیت هایی را به مرورگر شما اضافه کنند. افزونه‌ها از آخرین فناوری‌های html,CSS و جاوااسکریپت تا به آنجایی که مرورگر آن‌ها را پشتیبانی کند، استفاده می‌کنند.
در این سری سعی خواهیم کرد برای هر مرورگر شناخته شده، یک افزونه ایجاد کنیم و ابتدا از آنجا که خودم از کروم استفاده می‌کنم، اولین افزونه را برای کروم خواهم نوشت.

این افزونه قرار است چه کاری انجام دهد؟

کاری که برای این افزونه تدارک دیده‌ام این است: موقعی‌که سایت dotnettips.info به روز شد مرا آگاه کند. این آگاه سازی را از طریق یک نوتیفیکیشن به اطلاع کاربر میرسانیم. صفحه تنظیمات این افزونه شامل گزینه‌های "آخرین مطالب"،"نظرات آخرین مطالب"،"آخرین اشتراک ها"و"آخرین نظرات اشتراک ها" خواهد بود که به طور پیش فرض تنها گزینه اول فعال خواهد بود و همچنین یک گزینه نیز برای وارد کردن یک عدد صحیح جهت اینکه به افزونه بگوییم هر چند دقیقه یکبار سایت را چک کن. چک کردن سایت هم از طریق فید RSS صورت می‌گیرد.

فایل manifest.json

این فایل برای ذخیره سازی اطلاعاتی در مورد افزونه به کار می‌رود که شامل نام افزونه، توضیح کوتاه در مورد افزونه و ورژن و ... به کار می‌رود که همه این اطلاعات در قالب یا فرمت json نوشته می‌شوند و در بالاترین حد استفاده برای تعریف اهداف افزونه و اعطای مجوز به افزونه از آن استفاده می‌کنیم. این فایل بخش‌های زیر را در یک افزونه تعریف می‌کند که به مرور با آن آشنا می‌شویم.



کد زیر را در فایل manifest.json می‌نویسیم:

```
{
  "manifest_version": 2,
```

```
  "name": "Dotnettips Updater",
  "description": "This extension keeps you updated on current activities on dotnettips.info",
  "version": "1.0",
  "icons": { "16": "icon.png",
             "48": "icon.png",
            "128": "icon.png" },

  "browser_action": {
    "default_icon": "icon.png",
    "default_popup": "popup.html"
  },
  "permissions": [
    "activeTab",
    "http://www.dotnettips.info"
  ]
}
```

اطلاعات اولیه شامل نام و توضیح و ورژن افزونه است. ورژن برنامه برای به روزآوری افزونه بسیار مهم است. موقعی که ورژن جدیدی از افزونه ارائه شود، گوگل وب استور اعلان آپدیت جدیدی را برای افزونه میکند. آیکن قسمت‌های مختلف افزونه هم با icons مشخص می‌شود که در سه اندازه باید ارائه شوند و البته اگه اندازه آن نباشد scale می‌شود. قسمت بعدی تعریف UI برنامه هست که گوگل کروم، به آن Browser Action می‌گوید. در اینجا یک آیکن و همچنین یک صفحه اختصاصی برای تنظیمات افزونه معرفی می‌کنیم. این آیکن کنار نوار آدرس نمایش داده می‌شود و صفحه popup موقعی نشان داده می‌شود که کاربر روی آن کلیک می‌کند. آیکن‌ها برای browser action در دو اندازه 19 و 38 پیکسلی هستند و در صورتی که تنها یک آیکن تعریف شود، به صورت خودکار عمل scale و تغییر اندازه صورت می‌گیرد. برای تعیین عکس برای هر اندازه می‌توانید کد را به صورت زیر بنویسید:

```
"default_icon": {                    // optional
            "19": "images/icon19.png",           // optional
            "38": "images/icon38.png"            // optional
          }
```

قسمت popup برای نمایش تنظیمات به کار می‌رود و درست کردن این صفحه همانند صفحه همیشگی html هست و خروجی آن روی پنجره popup افزونه رندر خواهد شد.

گزینه default_title نیز یکی از دیگر خصیصه‌های مهم و پرکاربرد این قسمت هست که متن tooltip می‌باشد و موقعی که که کاربر، اشاره‌گر را روی آیکن ببرد نمایش داده می‌شود و در صورتی که نوشته نشود، کروم نام افزونه را نمایش می‌دهد؛ برای همین ما هم چیزی ننوشتیم.

صفحات پس‌زمینه

اگر بخواهید برای صفحه popup کد جاوااسکریت بنویسید یا از jquery استفاده کنید، مانند هر صفحه‌ی وبی که درست می‌کنید آن را کنار فایل popup قرار داده و در popup آنها را صدا کرده و از آن‌ها استفاده کنید. ولی برای پردازش هایی که نیاز به UI وجود ندارد، می‌توان از صفحات پس زمینه استفاده کرد. در این حالت ما دو نوع صفحه داریم:

صفحات مصر یا Persistent Page

صفحات رویدادگرا یا Events Pages

اولین نوع صفحه، همواره فعال و در حال اجراست و دومی موقعی فعال می‌شود که به استفاده از آن نیاز است. گوگل توصیه می‌کند که تا جای ممکن از نوع دوم استفاده شود تا مقدار حافظه مصرفی حفظ شود و کارآیی مروگر بهبود بخشیده شود. کد زیر یک صفحه پس زمینه را از نوع رویدادگرا می‌سازد. به وضوح روشن است در صورتی که خاصیت Persistent با true مقداردهی شود، این صفحه مصرانه در تمام وقت باز بودن مرورگر، فعال خواهد بود:

```
"background": {
    "scripts": ["background.js"],
    "persistent": false
}
```

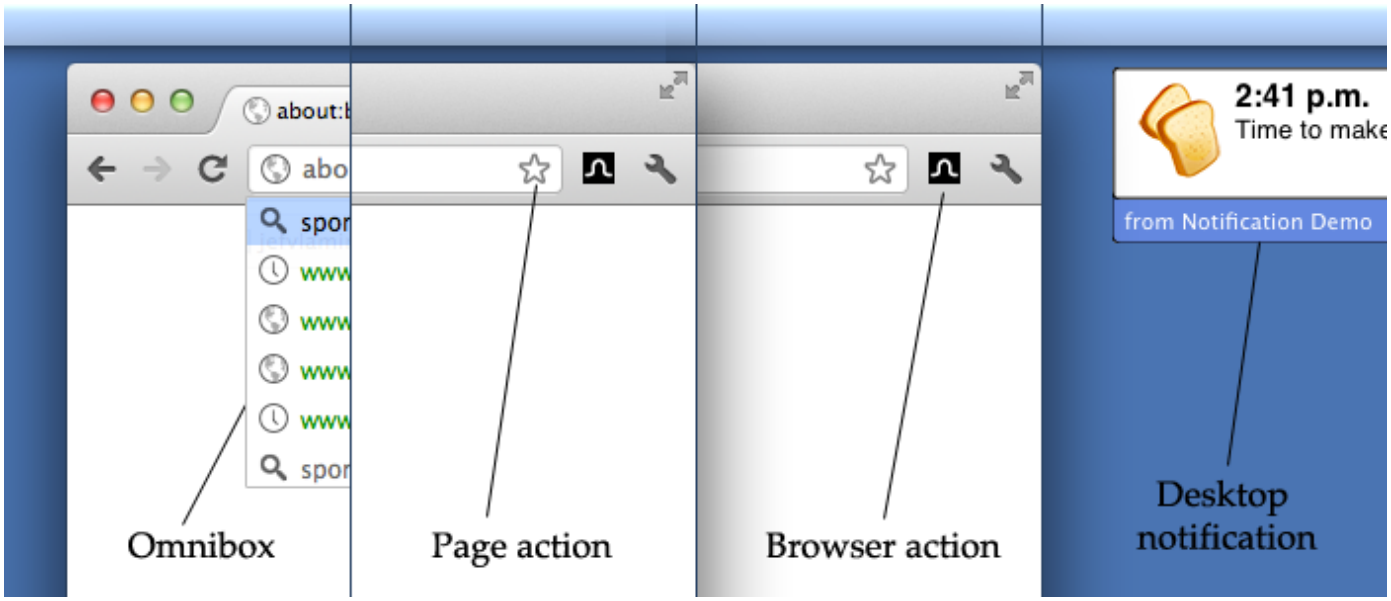
Content Script یا اسکریپت محتوا

در صورتی که بخواهید با هر صفحه‌ای که باز یا رفرش می‌شود، به DOM آن دسترسی پیدا کنید، از این خصوصیت استفاده کنید. در

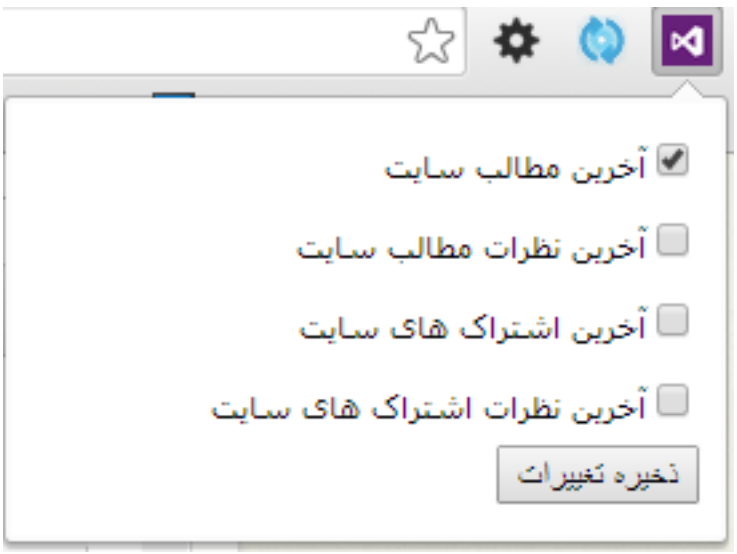
کد زیر برای پردازش اطلاعات DOM از فایل جاوااسکریپت بهره برده و در قسمت matches می‌گویید که چه صفحاتی باید از این کد استفاده کنند که در اینجا از پروتکل‌های HTTP استفاده میشود و اگر مثلا نوع FTP یا file صدا زده شود کد مورد نظر اجرا نخواهد شد. در مورد اینکه matches چگونه کار می‌کند و چگونه می‌توان آن را نوشت، از این صفحه استفاده کنید.

```
"content_scripts": [
    {
        "matches": ["http://*/*", "https://*/*"],
        "js": ["content.js"]
    }
]
```

آغاز کدنویسی (رابطهای کاربری)



اجازه دهید بقیه موارد را در حین کدنویسی تجربه کنیم و هر آنچه ماند را بعدا توضیح خواهیم داد. در اینجا من از یک صفحه با کد HTML زیر بهره برده ام که یک فرم دارد به همراه چهار چک باکس و در نهایت یک دکمه جهت ذخیره مقادیر. نام صفحه را popup.htm گذاشته ام و یک فایل popup.js هم دارم که در آن کد jquery نوشتم. قصد من این است که بتوان یک action browser به شکل زیر درست کنم:



کد html آن به شرح زیر است:

```
<html>
<head>
<meta charset="utf-8"/>

<script src="jquery.min.js"></script> <!-- Including jQuery -->
<script type="text/javascript" src="popup.js"></script>
</head>
<body style="direction:rtl;width:250px;">
<form >
<input type="checkbox" id="chkarticles" value="" checked="true">آخرین مطالب سایت</input><br/>
<input type="checkbox" id="chkarticlescomments" value="" >آخرین نظرات مطالب سایت</input><br/>
<input type="checkbox" id="chkshares" value="" >آخرین اشتراک‌های سایت</input><br/>
<input type="checkbox"  id="chksharescomments" value="" >آخرین نظرات اشتراک‌های سایت</input><br/>
<input id="btnsave" type="button" value="ذخیره تغییرات" />
    <div id="messageboard" style="color:green;"></div>
</form>

</body>
</html>
```

کد popup.js هم به شرح زیر است:

```
$(document).ready(function () {
    $("#btnsave").click(function() {
        var articles = $("#chkarticles").is(':checked');
        var articlesComments = $("#chkarticlescomments").is(':checked');
        var shares = $("#chkshares").is(':checked');
        var sharesComments = $("#chksharescomments").is(':checked');

        chrome.storage.local.set({ 'articles': articles, 'articlesComments': articlesComments,
'shares': shares, 'sharesComments': sharesComments }, function() {
            $("#messageboard").text( 'تنظیمات جدید اعمال شد');
        });
    });
});
```

در کد بالا موقعی که کاربر بر روی دکمه ذخیره، کلیک کند رویداد کلیک jquery فعال شده و مقادیر چک باکس‌ها را در متغیرهای مربوطه نگهداری می‌کند. نهایتا با استفاده از کلمه کلیدی کروم به ناحیه ذخیره سازی داده‌های کروم دست پیدا کرده و درخواست ذخیره مقادیر چک باکس را بر اساس ساختار نام و مقدار، ذخیره می‌کنیم و بعد از اعمال، توسط یک تابع callback به کاربر اعلام می‌کنیم که اطلاعات ذخیره شده است.

اولین مورد جدیدی که در بالا دیدیم، کلمه‌ی کلیدی chrome است. کروم برای توسعه دهندگانی که قصد نوشتن افزونه دارند api هایی را تدارک دیده است که میتوانید با استفاده از آنها به قسمت‌های مختلف مرورگر مثل بوک مارک یا تاریخچه فعالیت‌های مرورگر و ... دست پیدا کنید. البته برای اینکار باید در فایل manifest.json هم مجوز اینکار را درخواست نماییم. این ویژگی باید برای برنامه نویسان اندروید آشنا باشد. برای آشنایی هر چه بیشتر با مجوزها این صفحه را ببینید.

برای دریافت مجوز، کد زیر را به manifest اضافه می‌کنیم:

```
"permissions": [
    "storage"
  ]
```

مجوزی که در بالا درخواست کرده‌ایم مجوز دسترسی به ناحیه ذخیره سازی است. بعد از کلمه کلیدی chrome، کلمه‌ی local آمده است و می‌گوید که باید داده‌ها به صورت محلی و لوکال ذخیره شوند ولی اگر میخواهید داده‌ها در گوگل سینک شوند، باید به جای لوکال از کلمه کلیدی sync استفاده کنید یعنی:

```
chrome.storage.sync.set
```

فایل manifest نهایی:

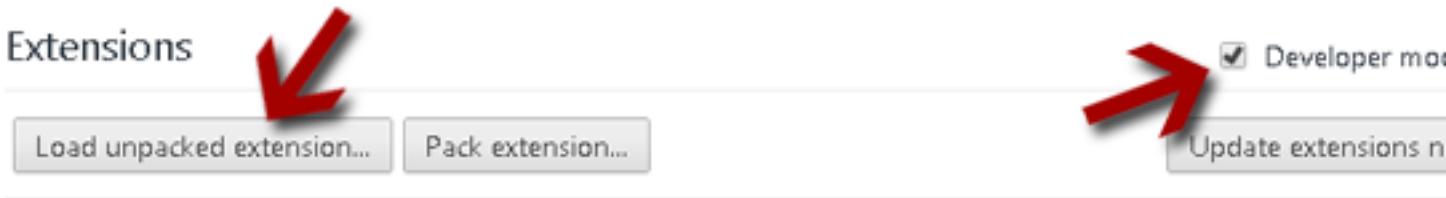
```
{
  "manifest_version": 2,

  "name": "Dotnettips Updater",
  "description": "This extension keeps you updated on current activities on dotnettips.info",
  "version": "1.0",

  "browser_action": {
    "default_icon": "icon.png",
    "default_popup": "popup.html"
  },
  "permissions": [
    "storage"
  ]
}
```

الان باید 4 فایل داشته باشید: فایل آیکن، popup.htm,popup.js و manifest.json. همه را داخل یک دایرکتوری قرار داده و در مروگر کروم به قسمت extensions بروید و گزینه Developer mode را فعال کنید تا یک تستی از کد نوشته شده بگیریم. گزینه Load Unpacked Extension را بزنید و آدرس دایرکتوری ایجاد شده را به آن بدهید.

```
chrome://extensions
```



الان باید مانند تصویر بالا یک آیکن کنار نوار آدرس یا به قول گوگل، Omni box ببینید. گزینه‌ها را تیک بزنید و روی دکمه ذخیره کلیک کنید. باید پیام مقادیر ذخیره شدند، نمایش پیدا کند. الان یک مشکل وجود دارد؛ داده‌ها ذخیره می‌شوند ولی موقعی که دوباره تنظیمات افزونه را باز کنید حالت اولیه نمایش داده میشود. پس باید تنظیمات ذخیره شده را خوانده و به آن‌ها اعمال کنیم. کد زیر را جهت دریافت مقادیر ذخیره شده می‌نویسیم. اینبار به جای استفاده از متد set از متد get استفاده می‌کنیم. به صورت آرایه، رشته نام مقادیر را درخواست می‌کنیم و در تابع callback، مقادیر به صورت آرایه برای ما برگشت داده می‌شوند.

```
chrome.storage.local.get(['articles', 'articlesComments', 'shares', 'sharesComments'], function (
items) {
        console.log(items[0]);
        $("#chkarticles").attr("checked", items["articles"]);
        $("#chkarticlescomments").attr("checked", items["articlesComments"]);
        $("#chkshares").attr("checked", items["shares"]);
        $("#chksharescomments").attr("checked", items["sharesComments"]);

    });
```

حالا برای اینکه افزونه‌ی شما متوجه تغییرات شود، به تب extensions رفته و در لیست افزونه‌ها به دنبال افزونه خود بگردید و گزینه Reload را انتخاب نمایید تا افزونه تغییرات را متوجه شود و صفحه را تست کنید.

Page Action

روش دیگر برای ارائه یک رابط کاربری، page action هست. این روش دقیقا مانند روش قبلی است، ولی جای آیکن عوض می‌شود. قبلا بیرون از نوار آدرس بود، ولی الان داخل نوار آدرس قرار می‌گیرد. جالب‌ترین نکته در این مورد این است که این آیکن در ابتدا مخفی شده است و شما تصمیم می‌گیرید که این آیکن چه موقع نمایش داده شود. مثلا آیکن RSS تنها موقعی نمایش داده

می‌شود که وب سایتی که باز شده است، دارای محتوای RSS باشد یا بوک مارک کردن یک آدرس برای همه‌ی سایت‌ها باز باشد و سایر موارد.

کد زیر نحوه‌ی تعریف یک page action را در manifest نشان می‌دهد. ما در این مثال یک page action را به طور موقت اضافه می‌کنیم و موقعی هم آن را نشان میدهیم که سایت dotnettips.info باز باشد. دلیل اینکه موقت اضافه می‌کنیم این است که باید یکی از دو گزینه رابط کاربری که تا به حال گفتیم، استفاده شود. در غیر این صورت کروم در هنگام خواندن فایل manifest در هنگام افزودن افزونه به مرورگر، پیام خطا خواهد داد و این مطلب را به شما گوشزد می‌کند. پس نمی‌توان دو گزینه را همزمان داشت و من میخواهم افزونه را در حالت browser action ارائه کنم. پس در پروژه نهایی، این مطلب page action نخواهد بود.

برای داشتن یک page action کد زیر را در manifest بنویسید.

```
"page_action": {
    "default_icon": {
        "19": "images/icon19.png",
        "38": "images/icon38.png"
    },
    "default_popup": "popup.html"
```

گزینه page action تعریف شد حالا باید کاری کنیم تا هر موقع صفحه‌ای باز می‌شود چک کند آیا سایت مورد نظر است یا خیر، اینکار را توسط صفحه‌ی پردازشی انجام می‌دهیم. پس تکه کد زیر را هم به manifest اضافه می‌کنیم:

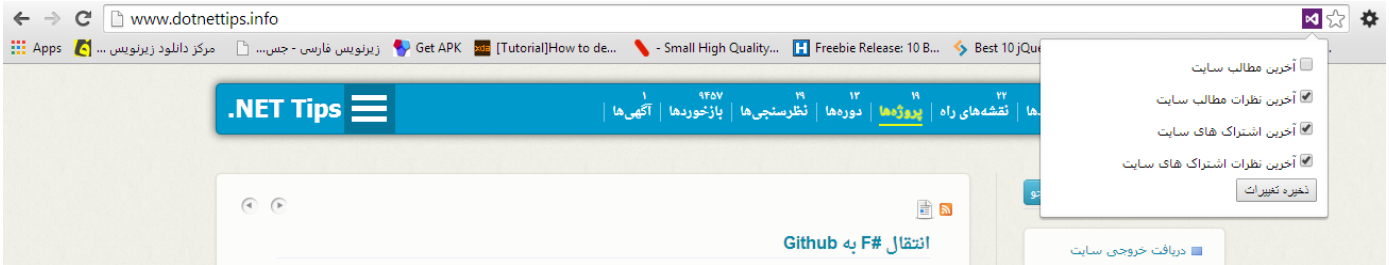
```
"background": {
    "scripts": ["page_action_validator.js"]
}
```

تا اینجا فایل جاوااسکریپت معرفی شد که کد زیر را دارد و در پس زمینه شروع به اجرا می‌کند.

```
function UrlValidation(tabId, changeInfo, tab) {
if (tab.url.indexOf('dotnettips.info') >-1) {
chrome.pageAction.show(tabId);
}
};
chrome.tabs.onUpdated.addListener(UrlValidation);
```

چون از api در این کد بهره برده‌ایم و آن هم مدیریت بر روی تب هاست، پس باید مجوز آن هم گرفته شود. کلمه "tabs" را در قسمت permissions اضافه کنید.

یک listener برای tabها ایجاد کرده‌ایم که اگر تب جدید ایجاد شد، یا تب قبلی به آدرس جدیدی تغییر پیدا کرد تابع UrlValidation را اجرا کند و در این تابع چک می‌کنیم که اگر url این تب شامل نام وب سایت می‌شود، page action روی این تب ظاهر شود. پس از انجام تغییرات، مجددا افزونه را بارگذاری می‌کنیم و تغییرات اعمال شده را می‌بینیم. سایت dotnettips را باز کنید یا صفحه را مجددا رفرش کنید تا تغییر اعمال شده را ببینید.



تغییرات موقت را حذف و کدها را به حالت قبلی یعنی browser action بر میگردانم.

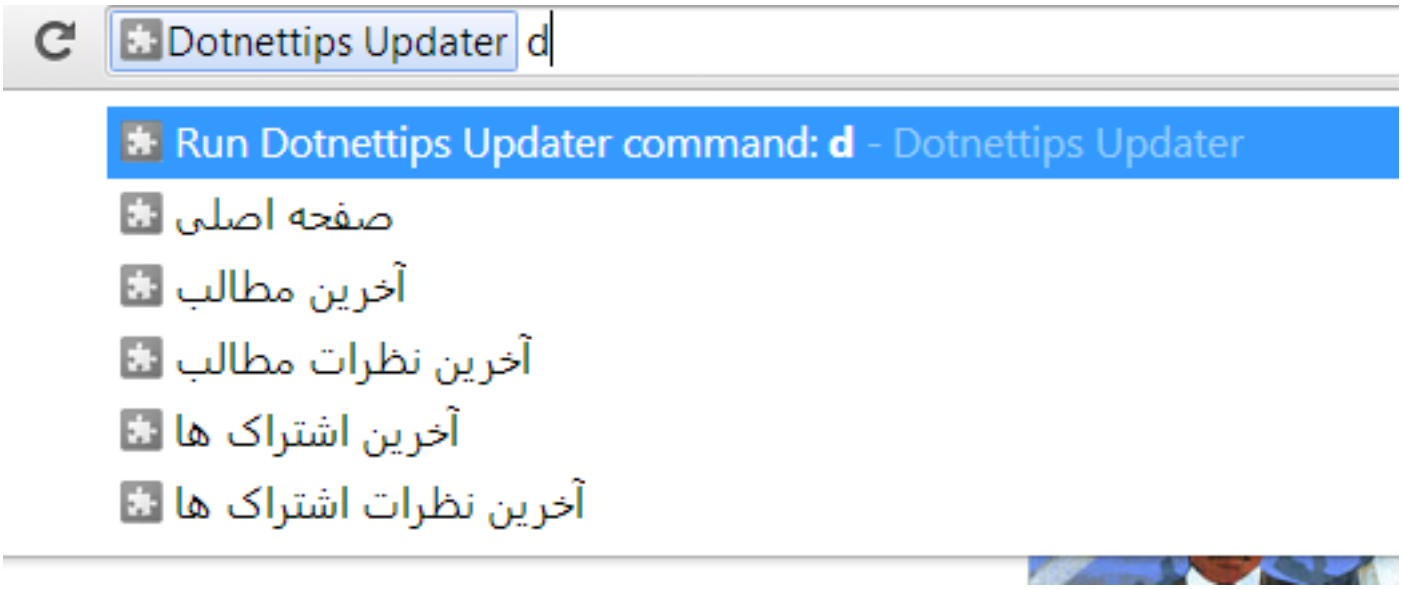
OmniBox

omnibox یک کلمه کلیدی است که در نوار آدرس مرورگر وارد می‌شود و در واقع می‌توانیم آن را نوع دیگری از رابط کاربری بنامیم. موقعی که شما کلمه کلیدی رزرو شده را وارد می‌کنید، در نوار آدرس کلماتی نشان داده میشود که کاربر میتواند یکی از آن‌ها را انتخاب کند تا عملی انجام شود. ما هم قرار است این کار را انجام دهیم. به این مثال دقت کنید:
میخواهیم موقعی که کاربرکلمه net. را تایپ می‌کند، 5 عبارت آخرین مطالب و آخرین اشتراک‌ها و آخرین نظرات مطالب و آخرین نظرات اشتراک‌ها و صفحه اصلی سایت نمایش داده شود و با انتخاب هر کدام، کاربر به سمت آن صفحه هدایت شود.
برای افزودن کلمه کلیدی در manifest خطوط زیر را اضافه کنید:

```
"omnibox": { "keyword" : ".net" }
```

با نوشتن خط بالا کلمه net. در مرورگر یک کلمه‌ی کلیدی به حساب خواهد آمد و موقعی که کاربر این کلمه را وارد کند، در سمت راست نوشته خواهد شد. در این حالت باید کلید تب را بزند تا به محیط دستوری آن برود.

در این حین می‌توانیم همزمان با تایپ کاربر، دستوراتی را به آن نشان بدهیم. من دوست دارم موقعی که کاربر حرفی را وارد کرد، لیستی از نام صفحات نوشته شود.



برای اینکار باید کدنویسی کنیم ، پس یک فایل پس زمینه را به manifest معرفی کنید:

```
"background": {
    "scripts": ["omnibox.js"]
```

در فایل ominbox.js دستوراتی که مرتبط با omnibox است را می‌نویسیم و کد زیر را به آن اضافه می‌کنیم:

```
chrome.omnibox.onInputChanged.addListener(function(text, suggest) {
    suggest([
  {content: ".net tips Home Page", description: "صفحه اصلی"},
      {content: ".net tips Posts", description: "آخرین مطالب"},
      {content: ".net tips News", description: "آخرین نظرات مطالب"},
      {content: ".net tips Post Comments", description: "آخرین اشتراک ها"},
      {content: ".net tips News Comments", description: "آخرین نظرات اشتراک ها"}
    ]);
});
```

chrome.omnibox شامل 4 رویداد می‌شود:

onInputStarted	بعد از اینکه کاربر کلمه کلیدی را وارد کرد اجرا می‌شود
onInputChanged	بعد از وارد کردن کلمه کلیدی هربار که کاربر تغییری در ورودی نوارد آدرس می‌دهد اجرا می‌شود.
onInputEntered	کاربر ورودی خود را تایید می‌کند. مثلا بعد از وارد کردن، کلید enter را می‌فشارد
onInputCancelled	کاربر از وارد کردن ورودی منصرف شده است؛ مثلا کلید ESC را فشرده است.

با نوشتن chrome.omnibox.onInputChanged.addListener ما یک listener ساخته‌ایم تا هر بار کاربر ورودی را تغییر داد، یک تابع callback که دو آرگومان را دارد، صدا بزند. این آرگومان‌ها یکی متن ورودی‌است و دیگری آرایه‌ی suggest که شما با تغییر آرایه می‌توانید عباراتی که همزمان با تایپ به کاربر پیشنهاد می‌شود را نشان دهید. البته می‌توانید با تغییر کد کاری کنید تا بر اساس حروفی که تا به حال تایپ کرده‌اید، دستورات را نشان دهد؛ ولی من به دلیل اینکه 5 دستور بیشتر نبود و کاربر راحت باشد، چنین کاری نکردم. همچنین وقتی شما برای هر یک description تعریف کنید، به جای نام پیشنهادی، توضیح آن را نمایش می‌دهد.
حالا وقت این است که کد زیر را جهت اینکه اگر کاربر یکی از کلمات پیشنهادی را انتخاب کرد، به صفحه‌ی مورد نظر هدایت شود، اضافه کنیم:

```
chrome.omnibox.onInputEntered.addListener(function (text) {

var location="";
    switch(text)
{
case ".net tips Posts":
location="http://www.dotnettips.info/postsarchive";
break;
case ".net tips News":
location="http://www.dotnettips.info/newsarchive";
break;
case ".net tips Post Comments":
location="http://www.dotnettips.info/commentsarchive";
break;
case".net tips News Comments":
location="http://www.dotnettips.info/newsarchive/comments";
break;
default:
location="http://www.dotnettips.info/";
}

    chrome.tabs.getSelected(null, function (tab) {
        chrome.tabs.update(tab.id, { url: location });
    });
});
```

ابتدا یک listener برای روی رویداد onInputEntered قرار داده تا وقتی کاربر عبارت وارد شده را تایید کرد، اجرا شود. در مرحله بعد چک می‌کنیم که عبارت وارد شده چیست و به ازای هر عبارت مشخص شده، آدرس آن صفحه را در متغیر location قرار می‌دهیم. در نهایت با استفاده از عبارت chrome.tabs.getSelected تب انتخابی را به یک تابع callback بر میگردانیم. اولین آرگومان windowId است، برای زمانی که چند پنجره کروم باز است که می‌توانید وارد نکنید تا پنجره فعلی و تب فعلی محسوب شود. برای همین نال رد کردیم. در تابع برگشتی، شیء tab شامل اطلاعات کاملی از آن تب مانند url و id و title می‌باشد و در نهایت با استفاده از دستور chrome.tabs.update اطلاعات تب را به روز می‌کنیم. آرگومان اول id تب را میدهیم تا بداند کدام تب باید تغییر کند و آرگومان بعدی می‌توانید هر یک از ویژگی‌های تب از قبیل آدرس فعلی یا عنوان آن و ... را تغییر دهید که ما آدرس آن را تغییر داده ایم.

Context Menu

یکی دیگر از رابطهای کاربری، منوی کانتکست هست که توسط chrome.contextmenus ارائه می‌شود و به مجوز "contextmenus" نیاز دارد. فعال سازی منوی کانتکست در قسمت‌های زیر ممکن است:

```
all, page, frame, selection, link, editable, image, video, audio
```

من گزینه‌ی dotenettips.info را برای باز کردن سایت، به Contextmenus اضافه می‌کنم. کد را در فایلی به اسم contextmenus.js ایجاد می‌کنم و در قسمت background آن را معرفی می‌کنم. برای باز کردن یک تب جدید برای سایت، نیاز به chrome.tabs داریم که البته نیاز به مجوز tabs هم داریم.

محتوای فایل contextmenus.js

```
var root = chrome.contextMenus.create({
    title: 'Open .net tips',
    contexts: ['page']
}, function () {
    var Home= chrome.contextMenus.create({
        title: 'Home',
        contexts: ['page'],
        parentId: root,
        onclick: function (evt) {
            chrome.tabs.create({ url: 'http://www.dotnettips.info' })
        }
    });
var Posts = chrome.contextMenus.create({
        title: 'Posts',
        contexts: ['page'],
        parentId: root,
        onclick: function (evt) {
            chrome.tabs.create({ url: 'http://www.dotnettips.info/postsarchive/' })
        }
    });
});
```

در کد بالا یک گزینه به context menu اضافه میشود و دو زیر منو هم دارد که یکی صفحه‌ی اصلی سایت را باز میکند و دیگری هم صفحه‌ی مطالب سایت را باز می‌کند.

تا به اینجا ما قسمت ظاهری کار را آماده کرده ایم و به دلیل اینکه مطلب طولانی نشود، این مطلب را در دو قسمت ارائه خواهیم کرد. در قسمت بعدی نحوه خواندن RSS و اطلاع رسانی و دیگر موارد را بررسی خواهیم کرد.