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؟

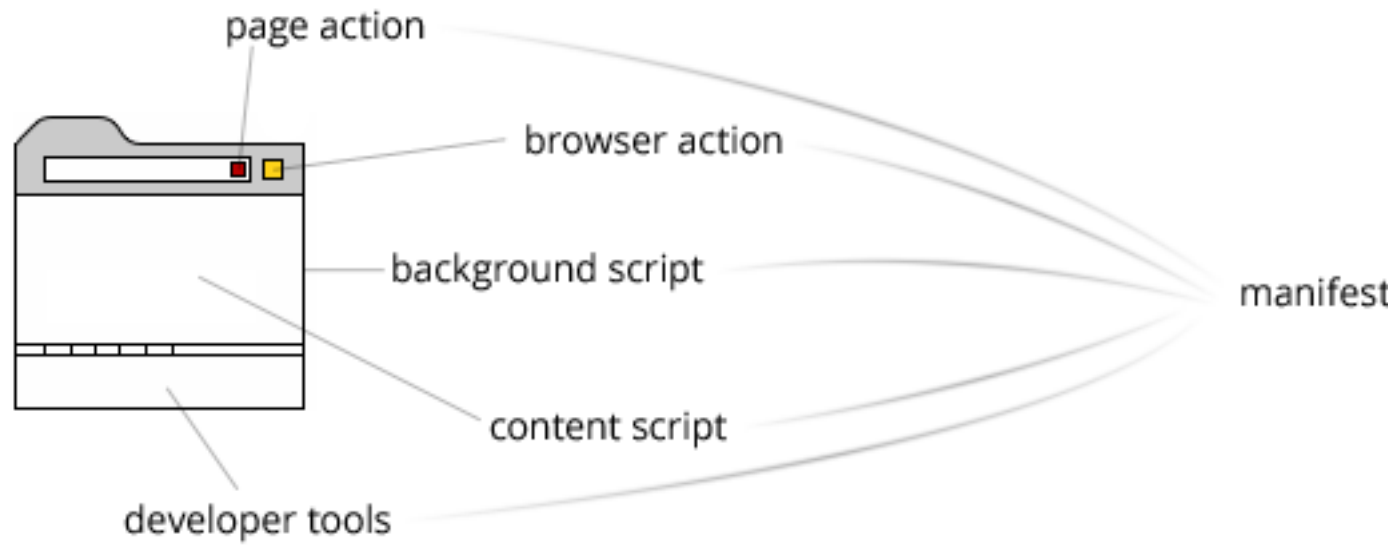
افزونه‌ها جزء مهمترین قسمت‌های یک مرورگر توسعه پذیر به شمار می‌آیند. افزونه‌ها سعی دارند تا قابلیت هایی را به مرورگر شما اضافه کنند. افزونه‌ها از آخرین فناوری‌های html,CSS و جاوااسکریپت تا به آنجایی که مرورگر آن‌ها را پشتیبانی کند، استفاده می‌کنند.
در این سری سعی خواهیم کرد برای هر مرورگر شناخته شده، 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
}
```

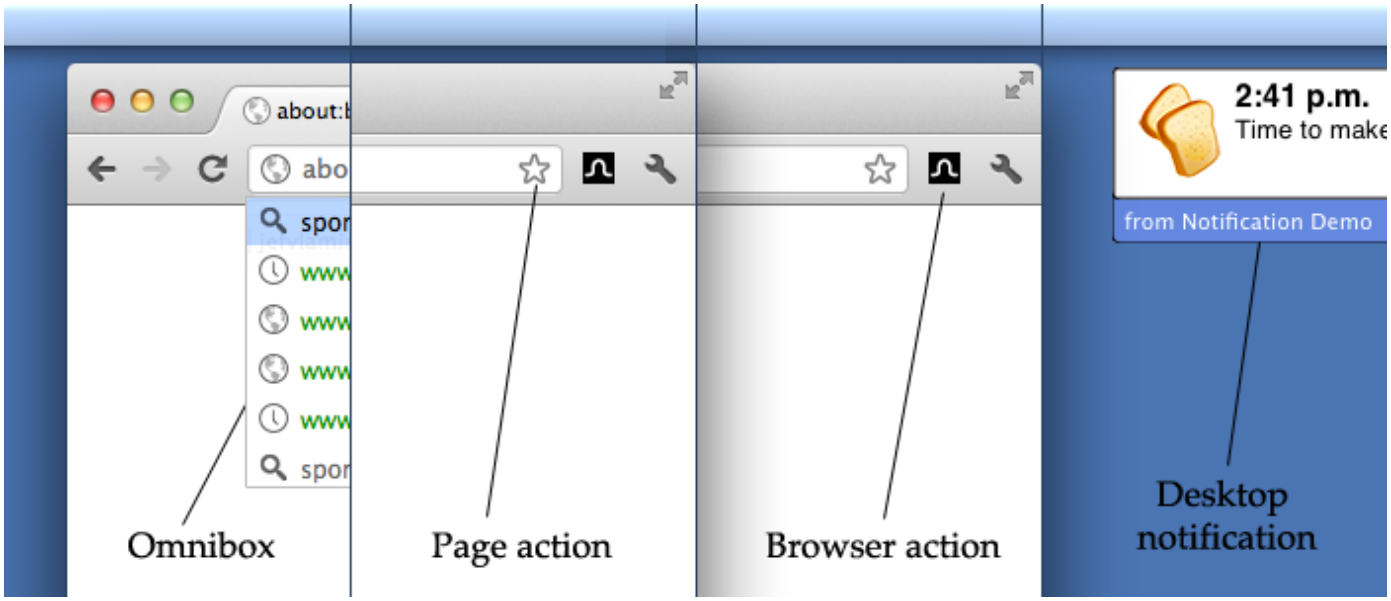
Content Script یا اسکریپت محتوا

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

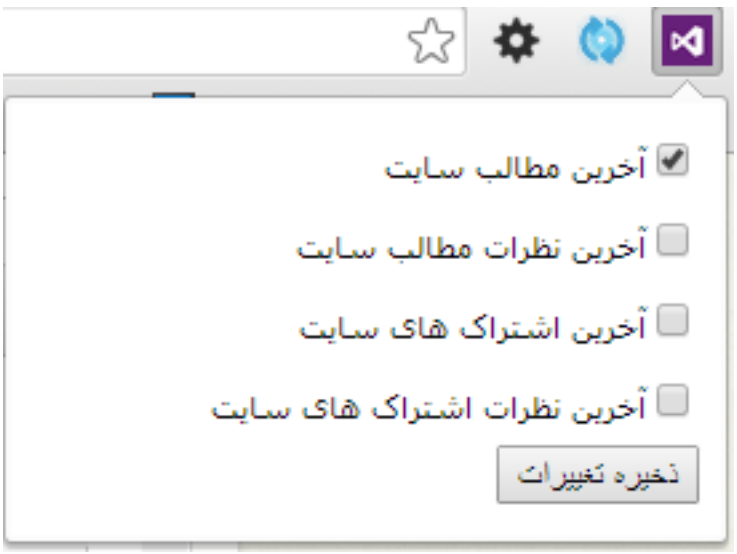
کد زیر برای پردازش اطلاعات DOM از فایل جاوااسکریپت بهره برده و در قسمت matches می‌گویید که چه صفحاتی باید از این کد استفاده کنند که در اینجا از پروتکل‌های HTTP استفاده میشود و اگر مثلا نوع FTP یا file صدا زده شود کد مورد نظر اجرا نخواهد شد. در مورد اینکه matches چگونه کار می‌کند و چگونه می‌توان آن را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

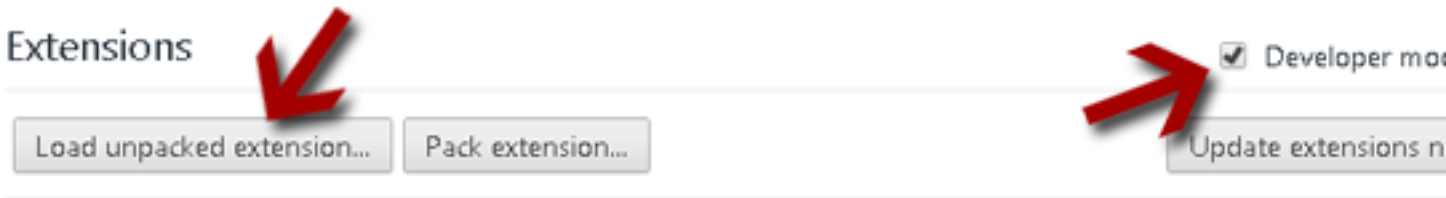
```
{
  "manifest_version": 2,

  "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:

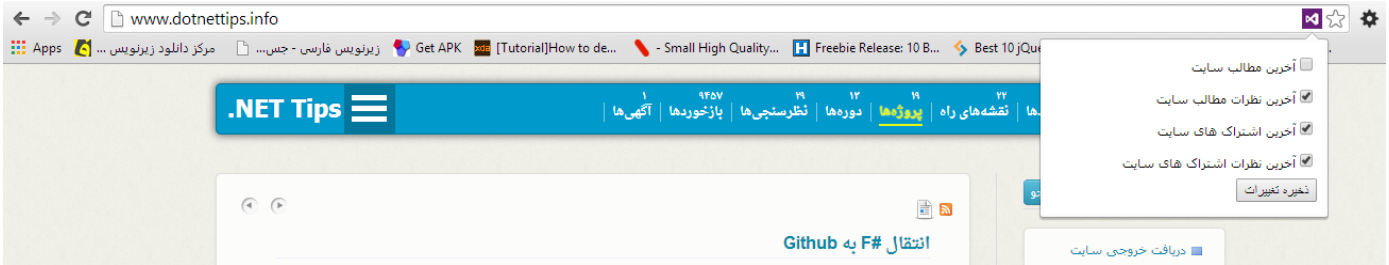
```
"background": {
    "scripts": ["page_action_validator.js"]
}
```

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.

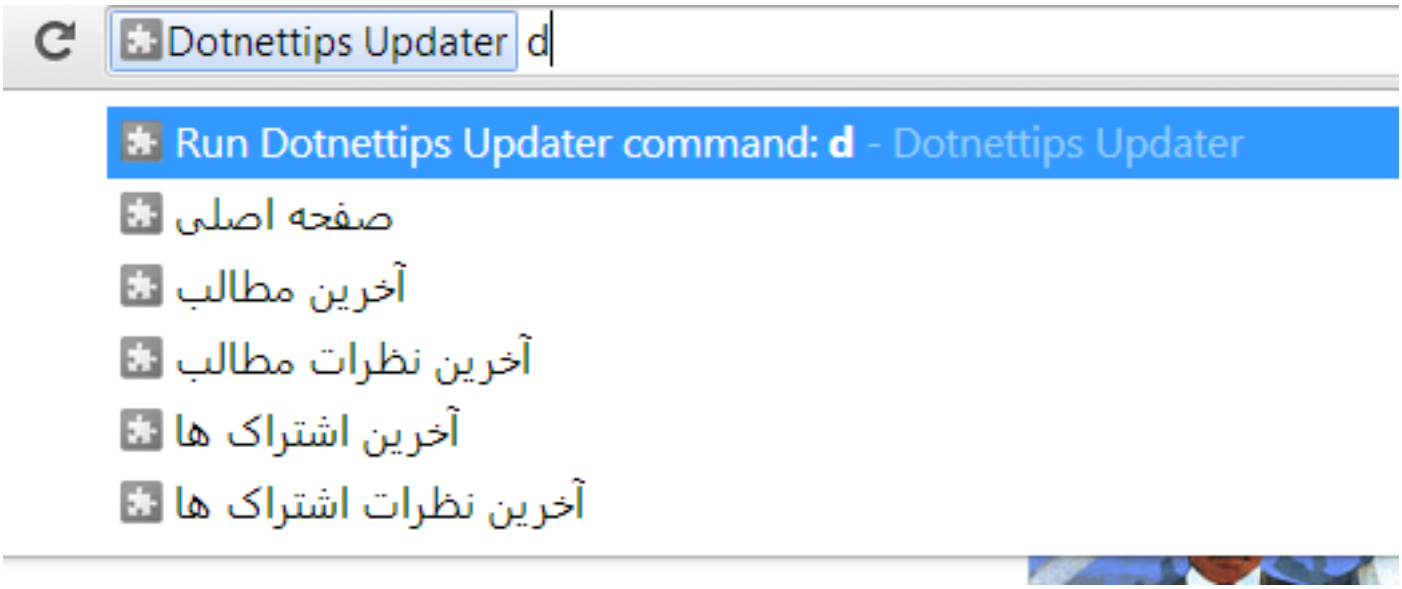
OmniBox

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.

عنوان: **نوشتن افزونه برای مرورگرها: قسمت دوم : کروم**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۱/۰۴ ۱:۱۵
آدرس: www.dotnettips.info
گروه‌ها: RSS, Plugin, Google Chrome, Extension, Addon

در مقاله پیشین ما ظاهر افزونه را طراحی و یک سری از قابلیت‌های افزونه را معرفی کردیم. در این قسمت قصد داریم پردازش پس زمینه افزونه یعنی خواندن RSS و اعلام به روز آوری سایت را مورد بررسی قرار دهیم و یک سری قابلیت هایی که گوگل در اختیار ما قرار داده است.

خواندن RSS توسط APIهای گوگل

گوگل در تعدادی از زمینه‌ها و سرویس‌های خودش apiهایی را ارائه کرده است که یکی از آن ها خواندن فید است و ما از آن برای خواندن RSS یا اتم وب سایت کمک می‌گیریم. روند کار بدین صورت است که ابتدا ما بررسی می‌کنیم کاربر چه مقادیری را ثبت کرده است و افزونه قرار است چه بخش هایی از وب سایت را بررسی نماید. در این حین، صفحه پس زمینه شروع به کار کرده و در هر سیکل زمانی مشخص شده بررسی می‌کند که آخرین بار چه زمانی RSS به روز شده است. اگر از تاریخ قبلی بزرگتر باشد، پس سایت به روز شده است و تاریخ جدید را برای دفعات آینده جایگزین تاریخ قبلی کرده و یک پیام را به صورت نوتیفیکیشن جهت اعلام به روز رسانی جدید در آن بخش به کاربر نشان می‌دهد.
اجازه دهید کدها را کمی شکیل‌تر کنیم. من از فایل زیر که یک فایل جاوااسکریپتی است برای نگه داشتن مقادیر بهره می‌برم تا اگر روزی خواستم یکی از آن‌ها را تغییر دهم راحت باشم و در همه جا نیاز به تغییر مجدد نداشته نباشم. نام فایل را (const.js) به خاطر ثابت بودن آن‌ها انتخاب کرده‌ام.

```
برای ذخیره مقادیر از ساختار نام و مقدار استفاده می‌کنیم که نام‌ها را اینجا ثبت کرده ام//
var Variables={
 posts:"posts",
 postsComments:"postsComments",
 shares:"shares",
 sharesComments:"sharesComments",
}

برای ذخیره زمان آخرین تغییر سایت برای هر یک از مطالب به صورت جداگانه نیاز به یک ساختار نام و مقدار//
است که نام‌ها را در اینجا ذخیره کرده ام
var DateContainer={
 posts:"dtposts",
 postsComments:"dtpostsComments",
 shares:"dtshares",
 sharesComments:"dtsharesComments",
 interval:"interval"
}

//برای نمایش پیام‌ها به کاربر
var Messages={
SettingsSaved:"تنظیمات ذخیره شد",
SiteUpdated:"سایت به روز شد",
PostsUpdated:"مطلب ارسالی جدید به سایت اضافه شد",
CommentsUpdated:"نظری جدیدی در مورد مطالب سایت آرسال شد",
SharesUpdated:"اشتراک جدید به سایت ارسال شد",
SharesCommentsUpdated:"نظری برای اشتراک‌های سایت اضافه شد"
}
//لینک‌های فید سایت
var Links={
 postUrl:"http://www.dotnettips.info/feeds/posts",
 posts_commentsUrl:"http://www.dotnettips.info/feeds/comments",
 sharesUrl:"http://www.dotnettips.info/feed/news",
 shares_CommentsUrl:"http://www.dotnettips.info/feed/newscomments"
}
//لینک صفحات سایت
var WebLinks={
Home:"http://www.dotnettips.info",
 postUrl:"http://www.dotnettips.info/postsarchive",
 posts_commentsUrl:"http://www.dotnettips.info/commentsarchive",
 sharesUrl:"http://www.dotnettips.info/newsarchive",
 shares_CommentsUrl:"http://www.dotnettips.info/newsarchive/comments"
}
```

موقعی که اولین بار افزونه نصب می‌شود، باید مقادیر پیش فرضی وجود داشته باشند که یکی از آن‌ها مربوط به مقدار سیکل زمانی

است (هر چند وقت یکبار فید را چک کند) و دیگری ذخیره مقادیر پیش فرض رابط کاربری که قسمت پیشین درست کردیم؛ پروسه پس زمینه برای کار خود به آن‌ها نیاز دارد و بعدی هم تاریخ نصب افزونه است برای اینکه تاریخ آخرین تغییر سایت را با آن مقایسه کند که البته با اولین به روزرسانی تاریخ فید جای آن را می‌گیرد. جهت انجام اینکار یک فایل init.js ایجاد کرده‌ام که قرار است بعد از نصب افزونه، مقادیر پیش فرض بالا را ذخیره کنیم.

```
chrome.runtime.onInstalled.addListener(function(details) {
var now=String(new Date());

var params={};
params[Variables.posts]=true;
params[Variables.postsComments]=false;
params[Variables.shares]=false;
params[Variables.sharesComments]=false;

params[DateContainer.interval]=1;

params[DateContainer.posts]=now;
params[DateContainer.postsComments]=now;
params[DateContainer.shares]=now;
params[DateContainer.sharesComments]=now;

 chrome.storage.local.set(params, function() {
  if(chrome.runtime.lastError)
   {
       /* error */
       console.log(chrome.runtime.lastError.message);
       return;
   }
        });
});
```

chrome.runtime شامل رویدادهایی چون onSuspend ، onStartup ، onInstalled و ... است که مربوطه به وضعیت اجرایی افزونه میشود. آنچه ما اضافه کردیم یک listener برای زمانی است که افزونه نصب شده است و در آن مقادیر پیش فرض ذخیره می‌شوند. اگر خوب دقت کنید می‌بینید که روش دخیره سازی ما در اینجا کمی متفاوت از مقاله پیشین هست و شاید پیش خودتان بگویید که احتمالا به دلیل زیباتر شدن کد اینگونه نوشته شده است ولی مهمترین دلیل این نوع نوشتار این است که متغیرهای بین {} آنچنان فرقی با خود string نمی‌کنند یعنی کد زیر:

```
chrome.storage.local.set('mykey':myvalue,....
```

با کد زیر برابر است:

```
chrome.storage.local.set(mykey:myvalue,...
```

پس اگر مقداری را داخل متغیر بگذاریم آن مقدار حساب نمی‌شود؛ بلکه کلید نام متغیر خواهد شد.
برای معرفی این دو فایل const.js و init.js به manifest.json می‌توانید به صورت زیر عمل کنید:

```
"background": {
    "scripts": ["const.js","init.js"]
}
```

در این حالت خود اکستنشن در زمان نصب یک فایل html درست کرده و این دو فایل js را در آن صدا میزند که البته خود ما هم می‌توانیم اینکار را مستقیما انجام دهیم. مزیت اینکه ما خودمان مسقیما این کار را انجام دهیم این است که در صورتی که فایل‌های js ما زیاد شوند، فایل manifest.jason زیادی شلوغ شده و شکل زشتی پیدا می‌کند و بهتر است این فایل را تا آنجا که می‌توانیم خلاصه نگه داریم. البته روش بالا برای دو یا سه تا فایل js بسیار خوب است ولی اگر به فرض بشود 10 تا یا بیشتر بهتر است یک فایل جداگانه شود و من به همین علت فایل background.htm را درست کرده و به صورت زیر تعریف کرده‌ام:
نکته:نمی توان در تعریف بک گراند هم فایل اسکریپت معرفی کرد و هم فایل html

```
"background": {
    "page": "background.htm"
}
```

background.htm

```
<html>
  <head>
  <script type="text/javascript" src="const.js"></script>
    <script type="text/javascript" src="https://www.google.com/jsapi"></script>
    <script type="text/javascript" src="init.js"></script>
<script type="text/javascript" src="omnibox.js"></script>
<script type="text/javascript" src="rssreader.js"></script>
<script type="text/javascript" src="contextmenus.js"></script>
  </head>
  <body>
  </body>
</html>
```

لینک‌های بالا به ترتیب معرفی ثابت‌ها، لینک api گوگل که بعدا بررسی می‌شود، فایل init.js برای ذخیره مقادیر پیش فرض، فایل ominibox که در مقاله پیشین در مورد آن صحبت کردیم و فایل rssreader.js که جهت خواندن rss در پایینتر در موردش بحث می‌کنیم و فایل contextmenus که این را هم در مطلب پیشین توضیح دادیم.
جهت خواندن فید سایت ما از Google API استفاده می‌کنیم؛ اینکار دو دلیل دارد:
کدنویسی راحت‌تر و خلاصه‌تر برای خواندن RSS
استفاده اجباری از یک پروکسی به خاطر Content Security Policy و حتی CORS

قبل از اینکه manifst به ورژن 2 برسد ما اجازه داشتیم کدهای جاوااسکریپت به صورت inline در فایل‌های html بنویسیم و یا اینکه از منابع و آدرس‌های خارجی استفاده کنیم برای مثال یک فایل jquery بر روی وب سایت jquery ؛ ولی از ورژن 2 به بعد، گوگل سیاست امنیت محتوا Content Security Policy را که سورس و سند اصلی آن در اینجا قرار دارد، به سیستم Extension خود افزود تا از حملاتی قبیل XSS و یا تغییر منبع راه دور به عنوان یک malware جلوگیری کند. پس ما از این به بعد نه اجازه داشتیم inline بنویسیم و نه اجازه داشتیم فایل jquery را از روی سرورهای سایت سازنده صدا بزنیم. پس برای حل این مشکل، ابتدا مثل همیشه یک فایل js را در فایل html معرفی می‌کردیم و برای حل مشکل دوم باید منابع را به صورت محلی استفاده می‌کردیم؛ یعنی فایل jquery را داخل دایرکتوری extension قرار می‌دادیم.
برای حل مشکل مشکل صدا زدن فایل‌های راه دور ما از Relaxing the Default Policy استفاده می‌کنیم که به ما یک لیست سفید ارائه می‌کند و در این لیست سفید دو نکته‌ی مهم به چشم میخورد که یکی از آن این است که استفاده از آدرس هایی با پروتکل Https و آدرس لوکال local host/127.0.0.1 بلا مانع است و از آنجا که api گوگل یک آدرس Https است، می‌توانیم به راحتی از API آن استفاده کنیم. فقط نیاز است تا خط زیر را به manifest.json اضافه کنیم تا این استثناء را برای ما در نظر بگیرد.

```
"content_security_policy": "script-src 'self' https://*.google.com; object-src 'self'"
```

در اینجا استفاده از هر نوع subdomain در سایت گوگل بلامانع اعلام می‌شود.
بنابراین آدرس زیر به background.htm اضافه می‌شود:

```
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
```

استفاده از این Api در rssreader.js

فایل rssreader.js را به background.htm اضافه می‌کنیم و در آن کد زیر را می‌نویسیم:

```
google.load("feeds", "1");
google.setOnLoadCallback(alarmManager);
```

آدرسی که ما از گوگل درخواست کردیم فقط مختص خواندن فید نیست؛ تمامی apiهای جاوااسکریپتی در آن قرار دارند و ما تنها نیاز داریم قسمتی از آن لود شود. پس اولین خط از دستور بالا بارگذاری بخش مورد نیاز ما را به عهده دارد. در مورد این دستور این صفحه را مشاهده کنید.
در خط دوم ما تابع خودمان را به آن معرفی می‌کنیم تا وقتی که گوگل لودش تمام شد این تابع را اجرا کند تا قبل از لود ما از توابع

آن استفاده نکنیم و خطای undefined دریافت نکنیم. تابعی که ما از آن خواستیم اجرا کند alarmManager نام دارد و قرار است یک آلارم و یک سیکل زمانی را ایجاد کرده و در هر دوره، فید را بخواند. کد تابع مدنظر به شرح زیر است:

```
function alarmManager()
{
chrome.storage.local.get(DateContainer.interval,function ( items) {
period_time==items[DateContainer.interval];
chrome.alarms.create('RssInterval', {periodInMinutes: period_time});
});


chrome.alarms.onAlarm.addListener(function (alarm) {
console.log(alarm);
    if (alarm.name == 'RssInterval') {

var boolposts,boolpostsComments,boolshares,boolsharesComments;
chrome.storage.local.get([Variables.posts,Variables.postsComments,Variables.shares,Variables.sharesComm
ents],function ( items) {
boolposts=items[Variables.posts];
boolpostsComments=items[Variables.postsComments];
boolshares=items[Variables.shares];
boolsharesComments=items[Variables.sharesComments];


chrome.storage.local.get([DateContainer.posts,DateContainer.postsComments,DateContainer.shares,DateCont
ainer.sharesComments],function ( items) {

var Vposts=new Date(items[DateContainer.posts]);
var VpostsComments=new Date(items[DateContainer.postsComments]);
var Vshares=new Date(items[DateContainer.shares]);
var VsharesComments=new Date(items[DateContainer.sharesComments]);

if(boolposts){var result=RssReader(Links.postUrl,Vposts,DateContainer.posts,Messages.PostsUpdated);}
if(boolpostsComments){var
result=RssReader(Links.posts_commentsUrl,VpostsComments,DateContainer.postsComments,Messages.CommentsUp
dated); }
if(boolshares){var
result=RssReader(Links.sharesUrl,Vshares,DateContainer.shares,Messages.SharesUpdated);}
if(boolsharesComments){var
result=RssReader(Links.shares_CommentsUrl,VsharesComments,DateContainer.sharesComments,Messages.SharesC
ommentsUpdated);}

});
});
    }
});
}
```

خطوط اول تابع alarmManager وظیفه‌ی خواندن مقدار interval را که در init.js ذخیره کرده‌ایم، دارند که به طور پیش فرض 10 ذخیره شده است تا تایمر یا آلارم خود را بر اساس آن بسازیم. در خط chrome.alarms.create یک آلارم با نام rssinterval می‌سازد و قرار است هر 10 دقیقه وظایفی که بر دوشش گذاشته می‌شود را اجرا کند **(استفاده از api جهت دسترسی به آلارم نیاز به مجوز "alarms" دارد)** . وظایفش از طریق یک listener که بر روی رویداد chrome.alarms.onAlarm گذاشته شده است مشخص می‌شود. در خط بعدی مشخص می‌شود که این رویداد به خاطر چه آلارمی صدا زده شده است. البته از آنجا که ما یک آلارم داریم، نیاز چندانی به این کد نیست. ولی اگر پروژه شما حداقل دو آلارم داشته باشد نیاز است مشخص شود که کدام آلارم باعث صدا زدن این رویداد شده است. در مرحله بعد مشخص می‌کنیم که کاربر قصد بررسی چه قسمت‌هایی از سایت را داشته است و در تابع callback آن هم تاریخ آخرین تغییرات هر بخش را می‌خوانیم و در متغیری نگه داری می‌کنیم. هر کدام را جداگانه چک کرده و تابع RssReader را برای هر کدام صدا می‌زنیم. این تابع 4 پارامتر دارد:

آدرس فیدی که قرار است از روی آن بخواند

آخرین به روزسانی که از سایت داشته متعلق به چه تاریخی است.

نام کلید ذخیره سازی تاریخ آخرین تغییر سایت که اگر بررسی شد و مشخص شد سایت به روز شده است، تاریخ جدید را روی آن ذخیره کنیم.

در صورتی که سایت به روز شده باشد نیاز است پیامی را برای کاربر نمایش دهیم که این پیام را در اینجا قرار می‌دهیم.

کد تابع rssreader

```
function RssReader(URL,lastupdate,datecontainer,Message) {
            var feed = new google.feeds.Feed(URL);
            feed.setResultFormat(google.feeds.Feed.XML_FORMAT);
                    feed.load(function (result) {
if(result!=null)
{
var strRssUpdate = result.xmlDocument.firstChild.firstChild.childNodes[5].textContent;
var RssUpdate=new Date(strRssUpdate);

if(RssUpdate>lastupdate)
{
SaveDateAndShowMessage(datecontainer,strRssUpdate,Message)
}
}
});
}
```

در خط اول فید توسط گوگل خوانده میشود، در خط بعدی ما به گوگل میگوییم که فید خوانده شده را چگونه به ما تحویل دهد که ما قالب xml را خواسته ایم و در خط بعدی اطلاعات را در متغیری به اسم result قرار میدهد که در یک تابع برگشتی آن را در اختیار ما میگذارد. از آن جا که ما قرار است تگ **lastBuildDate** را بخوانیم که پنجمین تگ اولین گره در اولین گره به حساب می‌آید، خط زیر این دسترسی را برای ما فراهم می‌کند و چون تگ ما در یک مکان ثابت است با همین تکه کد، دسترسی مستقیمی به آن داریم:

```
var strRssUpdate = result.xmlDocument.firstChild.firstChild.childNodes[5].textContent;
```

مرحله بعد تاریخ را که در قالب رشته‌ای است، تبدیل به تاریخ کرده و با lastupdate یعنی آخرین تغییر قبلی مقایسه می‌کنیم و اگر تاریخ برگرفته از فید بزرگتر بود، یعنی سایت به روز شده است و تابع SaveDateAndShowMessage را صدا می‌زنیم که وظیفه ذخیره سازی تاریخ جدید و ایجاد notification را به عهده دارد و سه پارامتر کلید ذخیره سازی و مقدار آن و پیام را به آن پاس می‌کنیم.

کد تابع SaveDateAndShowMesage

```
function SaveDateAndShowMessage(DateField,DateValue,Message)
{
var params={
}
params[DateField]=DateValue;

chrome.storage.local.set( params,function(){

var options={
  type: "basic",
   title: Messages.SiteUpdated,
   message: Message,
   iconUrl: "icon.png"
}
chrome.notifications.create("",options,function(){
chrome.notifications.onClicked.addListener(function(){
chrome.tabs.create({'url': WebLinks.Home}, function(tab) {
});
});
});
});
}
```

خطوط اول مربوط به ذخیره تاریخ است و دومین نکته نحوه‌ی ساخت نوتیفکیشن است. اجرای یک notification نیاز به مجوز "notifications " دارد که مجوز آن در manifest به شرح زیر است:

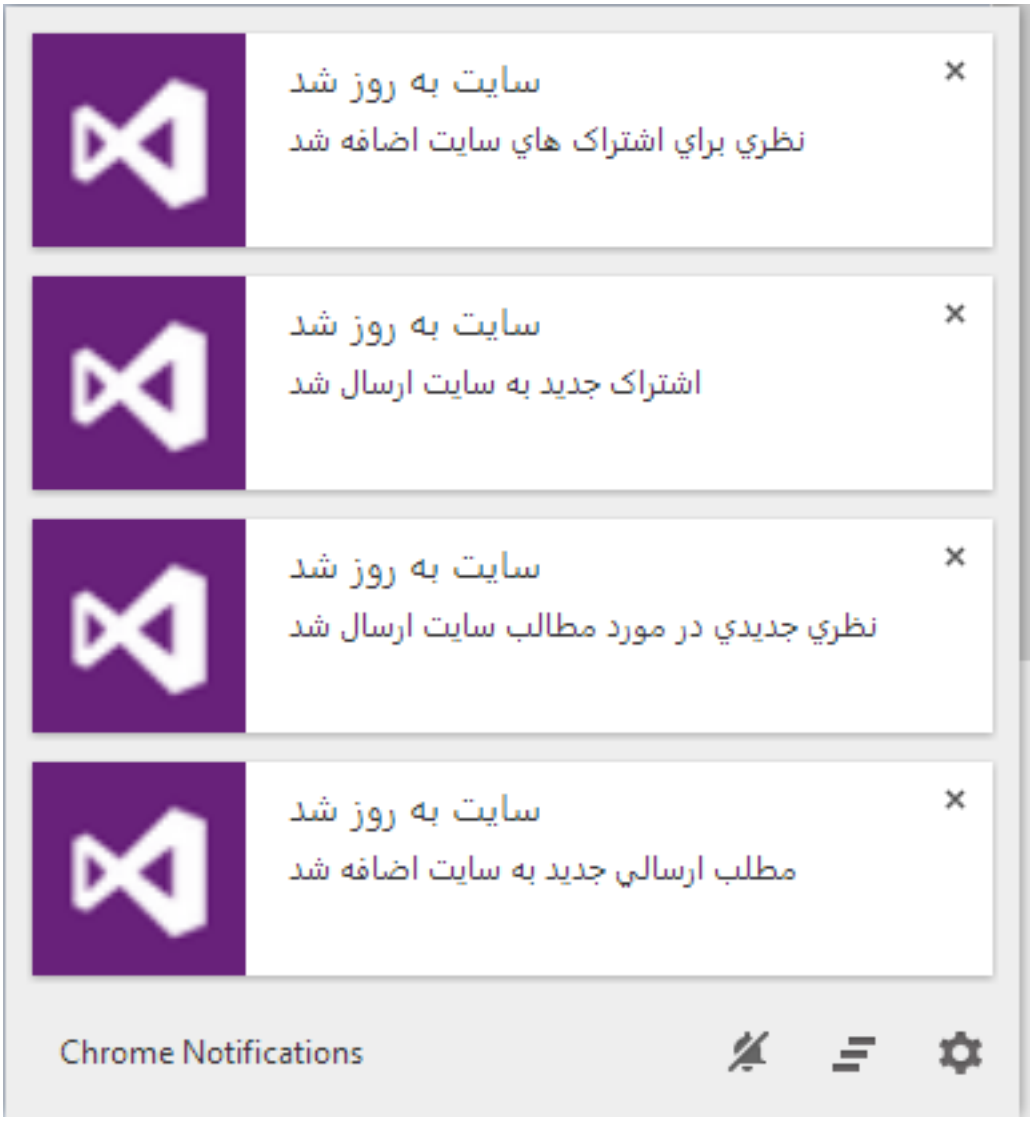
```
"permissions": [
    "storage",
     "tabs",
 "alarms",
 "notifications"
  ]
```

در خطوط بالا سایر مجوزهایی که در طول این دوره به کار اضافه شده است را هم می‌بینید.
برای ساخت نوتیفکیشن از کد chrome.notifications.create استفاده می‌کنیم که پارامتر اول آن کد یکتا یا همان ID جهت ساخت نوتیفیکیشن هست که میتوان خالی گذاشت و دومی تنظیمات ساخت آن است؛ از قبیل عنوان و آیکن و ... که در بالا به اسم options معرفی کرده ایم و در آگومان دوم آن را معرفی کرده ایم و آرگومان سوم هم یک تابع callback است که نوشتن آن اجباری است. options شامل عنوان، پیام، آیکن و نوع notification می‌باشد که در اینجا basic انتخاب کرده‌ایم. برای دسترسی به دیگر خصوصیت‌های options به اینجا و برای داشتن notificationهای زیباتر به عنوان rich notification به اینجا مراجعه کنید. برای اینکه این امکان باشد که کاربر با کلیک روی notification به سایت هدایت شود باید در تابع callback مربوط به notifications.create این کد اضافه گردد که در صورت کلیک یک تب جدید با آدرس سایت ساخته شود:

```
chrome.notifications.create("",options,function(){
chrome.notifications.onClicked.addListener(function(){
chrome.tabs.create({'url': WebLinks.Home}, function(tab) {
});});
});
```

نکته مهم: پیشتر معرفی آیکن به صورت بالا کفایت میکرد ولی بعد از این باگ کد زیر هم باید جداگانه به manifest اضافه شود:

```
"web_accessible_resources": [
    "icon.png"
  ]
```



خوب؛ کار افزونه تمام شده است ولی اجازه دهید این بار امکانات افزونه را بسط دهیم:
من می‌خواهم برای افزونه نیز قسمت تنظیمات داشته باشم. برای دسترسی به options میتوان از قسمت مدیریت افزونه‌ها در مرورگر یا حتی با راست کلیک روی آیکن browser action عمل کرد. در اصل این قسمت برای تنظیمات افزونه است ولی ما به خاطر آموزش و هم اینکه افزونه ما UI خاصی نداشت تنظیمات را از طریق browser action پیاده سازی کردیم و گرنه در صورتی که افزونه شما شامل UI خاصی مثلا نمایش فید مطالب باشد، بهترین مکان تنظیمات، options است. برای تعریف options در manifest.json به روش زیر اقدام کنید:

```
"options_page": "popup.html"
```

همان صفحه popup را در این بخش نشان میدهم و اینبار یک کار اضافه‌تر دیگر که نیاز به آموزش ندارد اضافه کردن input با Type=number است که برای تغییر interval به کار می‌رود و نحوه ذخیره و بازیابی آن را در طول دوره یاد گرفته اید.

جایزگزینی صفحات یا Override Pages

بعضی صفحات مانند بوک مارک و تاریخچه فعالیت‌ها History و همینطور newtab را می‌توانید جایگزین کنید. البته یک اکستنشن میتواند فقط یکی از صفحات را جایگزین کند. برای تعیین جایگزین در manifest اینگونه عمل می‌کنیم:

```
"chrome_url_overrides": {
    "newtab": "newtab.htm"
  }
```

ایجاد یک تب اختصاصی در Developer Tools

تکه کدی که باید manifest اضافه شود:

```
"devtools_page": "devtools.htm"
```

شاید فکر کنید کد بالا الان شامل مباحث ui و ... می‌شود و بعد به مرورگر اعمال خواهد شد؛ در صورتی که اینگونه نیست و نیاز دارد چند خط کدی نوشته شود. ولی مسئله اینست که کد بالا تنها صفحات html را پشتیبانی می‌کند و مستقیما نمی‌تواند فایل js را بخواند. پس صفحه بالا را ساخته و کد زیر را داخلش می‌گذاریم:

```
<script src="devtools.js"></script>
```

فایل devtools.js هم شامل کد زیر می‌شود:

```
chrome.devtools.panels.create(
    "Dotnettips Updater Tools",
    "icon.png",
    "devtoolsui.htm",
    function(panel) {
    }
);
```

خط chrome.devtools.panels.create یک پنل یا همان تب را ساخته و در پارامترهای بالا به ترتیب عنوان، آیکن و صفحه‌ای که باید در آن رندر شود را دریافت می‌کند و پس از ایجاد یک callback اجرا می‌شود. اطلاعات بیشتر

APIها

برای دیدن لیست کاملی از APIها می‌توانید به مستندات آن رجوع کنید و این مورد را به یاد داشته باشید که ممکن است بعضی apiها در بعضی موارد پاسخ ندهند. به عنوان مثال در content scripts نمی‌توانید به chrome.devtools.panels دسترسی داشته باشید یا اینکه در DeveloperTools دسترسی به DOM میسر نیست. پس این مورد را به خاطر داشته باشید. همچنین بعضی apiها از نسخه‌ی خاصی به بعد اضافه شده‌اند مثلا همین مثال قبلی devtools از نسخه 18 به بعد اضافه شده است و به این معنی است با خیال راحت می‌توانید از آن استفاده کنید. یا آلارم‌ها از نسخه 22 به بعد اضافه شده‌اند. البته خوشبختانه امروزه با دسترسی آسانتر به اینترنت و آپدیت خودکار مرورگرها این مشکلات دیگر آن چنان رخ نمی‌دهند.

Messaging

همانطور که در بالا اشاره شد شما نمی‌توانید بعضی از apiها را در بعضی جاها استفاده کنید. برای حل این مشکل می‌توان از messaging استفاده کرد که دو نوع تبادلات پیغامی داریم:

One-Time Requests یا درخواست‌های تک مرتبه‌ای

Long-Lived Connections یا اتصالات بلند مدت یا مصر

درخواست‌های تک مرتبه ای

این درخواست‌ها همانطور که از نامش پیداست تنها یک مرتبه رخ می‌دهد؛ درخواست را ارسال کرده و منتظر پاسخ می‌ماند. به عنوان مثال به کد زیر که در content script است دقت کنید:

```
window.addEventListener("load", function() {
    chrome.extension.sendMessage({
        type: "dom-loaded",
        data: {
            myProperty   : "value"
        }
    });
}, true);
```

کد بالا یک ارسال کننده پیام است. موقعی که سایتی باز می‌شود، یک کلید با مقدارش را ارسال می‌کند و کد زیر در background گوش می‌ایستد تا اگر درخواستی آمد آن را دریافت کند:

```
chrome.extension.onMessage.addListener(function(request, sender, sendResponse) {
    switch(request.type) {
        case "dom-loaded":
            alert(request.data.myProperty   );
        break;
    }
    return true;
});
```

اتصالات بلند مدت یا مصر

اگر نیاز به یک کانال ارتباطی مصر و همیشگی دارید کدها را به شکل زیر تغییر دهید *contentscripts*

```
var port = chrome.runtime.connect({name: "my-channel"});
port.postMessage({myProperty: "value"});
port.onMessage.addListener(function(msg) {
    // do some stuff here
});
```

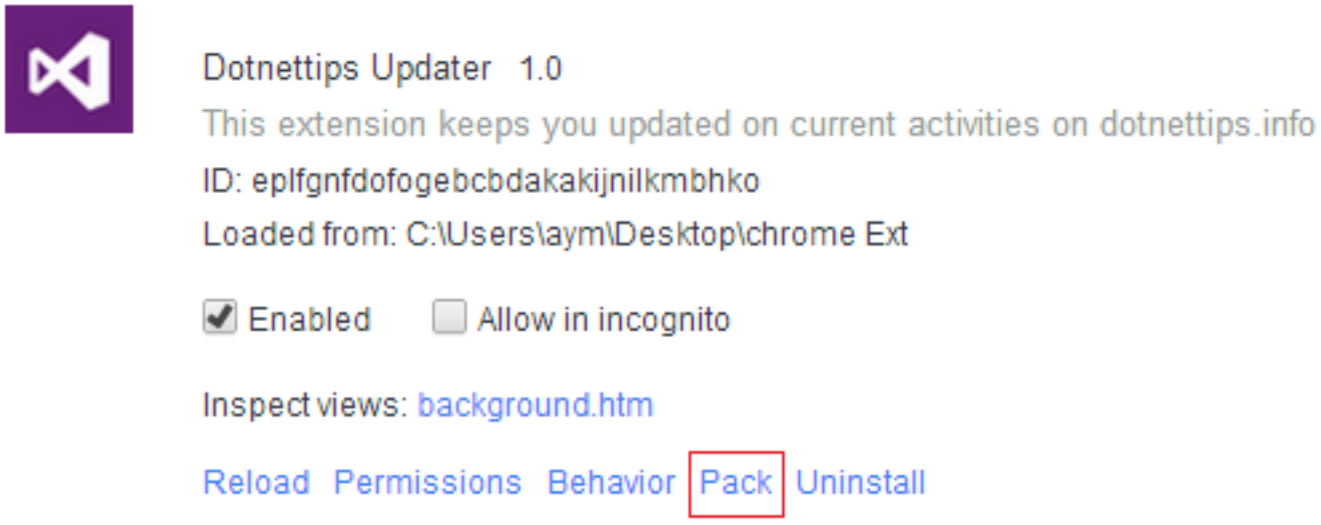
background

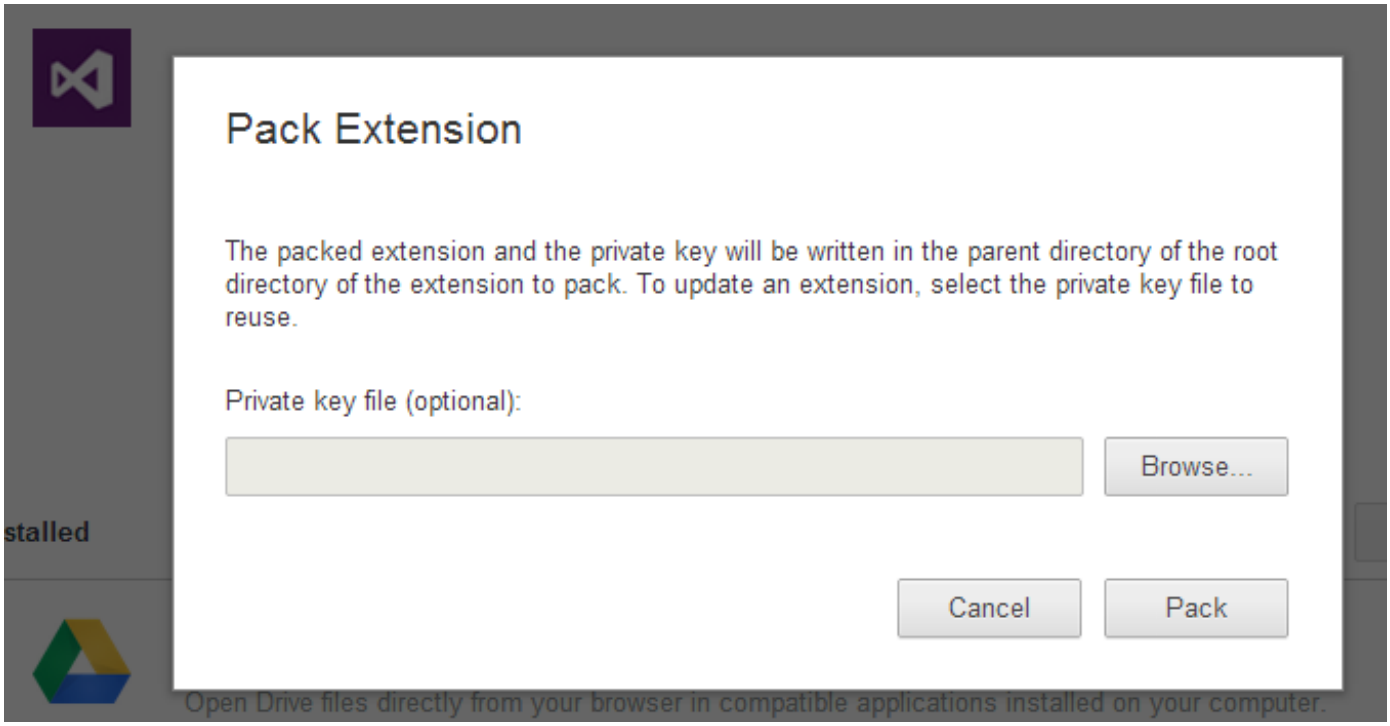
```
chrome.runtime.onConnect.addListener(function(port) {
    if(port.name == "my-channel"){
        port.onMessage.addListener(function(msg) {
            // do some stuff here
        });
    }
});
```

نمونه کد نمونه کدهایی که در سایت گوگل موجود هست می‌توانند کمک بسیاری خوبی باشند ولی اینگونه که پیداست اکثر مثال‌ها مربوط به نسخه‌ی یک manifest است که دیگر توسط مرورگرها پشتیبانی نمی‌شوند و مشکلاتی چون اسکریپت inline و CSP که در بالا اشاره کردیم را دارند و گوگل کدها را به روز نکرده است.

دیباگ کردن و پک کردن فایل‌ها برای تبدیل به فایل افزونه Debugging and packing

برای دیباگ کردن کدها می‌توان از دو نمونه console.log و alert برای گرفتن خروجی استفاده کرد و همچنین ابزار Chrome Apps & Extensions Developer Tool هم نسبتا امکانات خوبی دارد که البته میتوان از آن برای پک کردن اکستنشن نهایی هم استفاده کرد. برای پک کردن روی گزینه pack کلیک کرده و در کادر باز شده گزینه‌ی pack را بزنید. برای شما دو نوع فایل را در مسیر والد دایرکتوری extension نوشته شده درست خواهد کرد که یکی پسوند crx دارد که می‌شود همان فایل نهایی افزونه و دیگری هم پسوند pem دارد که یک کلید اختصاصی است و باید برای آپدیت‌های آینده افزونه آن را نگاه دارید. در صورتی که افزونه را تغییر دادید و خواستید آن را به روز رسانی کنید موقعی که اولین گزینه pack را می‌زنید و صفحه باز می‌شود قبل از اینکه دومین گزینه pack را بزنید، از شما می‌خواهد اگر دارید عملیات به روز رسانی را انجام می‌دهید، کلید اختصاصی آن را وارد نمایید و بعد از آن گزینه pack را بزنید:





آپلود نهایی کار در Google web store

برای آپلود نهایی کار به google web store که در آن تمامی برنامه‌ها و افزونه‌های کروم قرار دارند بروید. سمت راست آیکن تنظیمات را بزنید و گزینه developer dashboard را انتخاب کنید تا صفحه‌ی آپلود کار برای شما باز شود. دایرکتوری محتویات اکستنشن را zip کرده و آپلود نمایید. توجه داشته باشید که محتویات و سورس خود را باید آپلود کنید نه فایل crx را. بعد از آپلود موفقیت آمیز، صفحه‌ای ظاهر می‌شود که از شما آیکن افزونه را در اندازه 128 پیکسل میخواهد بعلاوه توضیحاتی در مورد افزونه، قیمت گذاری که به طور پیش فرض به صورت رایگان تنظیم شده است، لینک وب سایت مرتبط، لینک محل پرسش و پاسخ برای افزونه، اگر لینک یوتیوبی در مورد افزونه دارید، یک شات تصویری از افزونه و همینطور چند تصویر برای اسلایدشو سازی که در همان صفحه استاندارد آن‌ها را توضیح می‌دهد و در نهایت گزینه‌ی جالب‌تر هم اینکه اکستنشن شما برای چه مناطقی تهیه شده است که متاسفانه ایران را ندیدم که می‌توان همه موارد را انتخاب کرد. به خصوص در مورد ایران که آی پی‌ها هم صحیح نیست، انتخاب ایران چنان تاثیری ندارد و در نهایت گزینه‌ی publish را می‌زنید که متاسفانه بعد از این صفحه درخواست می‌کند برای اولین بار باید 5 دلار آمریکا پرداخت شود که برای بسیاری از ما این گزینه ممکن نیست.

سورس پروژه را می‌توانید از اینجا ببینید و خود افزونه را از اینجا دریافت کنید.