EBook Gratuito

APPRENDIMENTO google-chrome-extension

Free unaffiliated eBook created from **Stack Overflow contributors.**

#google-chrome-extension

Sommario

Di

Capitolo 1: Iniziare con google-chrome-extension

Osservazioni

TODO: breve descrizione delle estensioni di Chrome

Documentazione ufficiale

- Quali sono le estensioni? (hub di documentazione)
- Guida introduttiva (tutorial di base)
- Panoramica
- API JavaScript (elenco completo di `chrome.*` API)

Ulteriori letture

TODO: popolare con collegamenti ad argomenti di panoramica importanti

Examples

Esempio minimo assoluto

Qualsiasi estensione di Chrome inizia come *un'estensione decompressa* : una cartella contenente i file dell'estensione.

Un file che deve contenere è `manifest.json` , che descrive le proprietà di base dell'estensione. Molte delle proprietà in quel file sono facoltative, ma qui è un file `manifest.json` minimo assoluto:

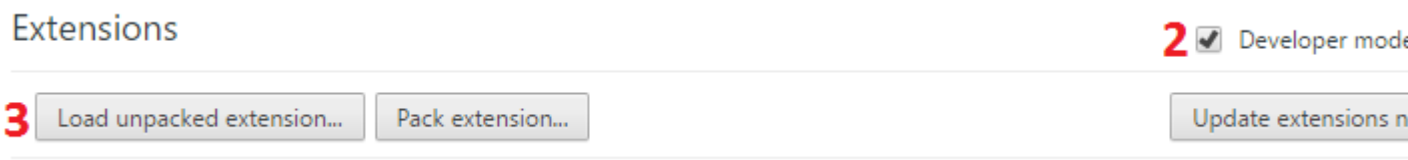
```
{
  "manifest_version": 2,
  "name": "My Extension",
  "version": "1.0"
}
```

Crea una cartella (per esempio, `myExtension`) da qualche parte, aggiungi `manifest.json` come elencato sopra ad essa.

Quindi, devi caricare l'estensione in Chrome.

1. Apri `chrome://extensions/` pagina, accessibile tramite **Menu> Altri strumenti> Estensioni** .

2. Abilita la **Modalità Sviluppatore** con una casella di controllo in alto a destra, se non è già abilitata.
3. Fai clic sul pulsante **Carica estensione scompattata ...** e seleziona la cartella `myExtension` creata.



Questo è tutto! La tua prima estensione è caricata da Chrome:

Certo, non fa ancora niente, quindi è un buon momento per leggere una panoramica dell'architettura di estensione per iniziare ad aggiungere le parti di cui hai bisogno.

Importante: quando si apportano modifiche all'estensione, non dimenticare di tornare a `chrome://extensions/` e premere il link **Ricarica** per l'interno dopo aver apportato le modifiche. In caso di script di contenuto, ricaricare anche la pagina di destinazione.

Pagina di sfondo

Le pagine di sfondo sono pagine implicite che contengono script in background. Uno script in background è un singolo script a esecuzione prolungata per gestire alcune attività o stati. Esiste per tutta la durata della tua estensione e solo una sua istanza alla volta è attiva.

Puoi dichiararlo così nel tuo `manifest.json` :

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

Una pagina di sfondo verrà generata dal sistema di estensione che include ciascuno dei file elencati nella proprietà degli script.

Hai accesso a tutte `chrome.*` API `chrome.*` consentite.

Esistono due tipi di pagine in background: pagine di sfondo **persistenti** sempre aperte e **pagine di eventi** aperte e chiuse secondo necessità.

Se vuoi che la tua pagina di sfondo non sia persistente, devi semplicemente impostare `persistent` -flag su false:

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

Questo script in background è attivo solo se viene attivato un evento su cui è registrato un listener. In generale si utilizza un `addListener` per la registrazione.

Esempio: l'app o l'estensione viene prima installata.

```
chrome.runtime.onInstalled.addListener(function() {
   console.log("The Extension is installed!");
});
```

Script di contenuti

Uno **script di contenuto** è un codice di estensione che viene eseguito insieme a una pagina normale.

Hanno accesso completo al DOM della pagina Web (e sono, di fatto, **l'unica parte dell'estensione che può accedere al DOM di una pagina**), ma il codice JavaScript è isolato, un concetto chiamato Mondo isolato . Ogni estensione ha il proprio contesto JavaScript script di script invisibile agli altri e alla pagina, evitando conflitti di codice.

Definizione di esempio in `manifest.json` :

```
  "content_scripts": [
    {
      "matches": ["http://www.stackoverflow.com/*"],
      "css": ["style.css"],
      "js": ["jquery.js", "myscript.js"]
    }
  ]
```

Gli attributi hanno il seguente significato:

Attributo	Descrizione
fiammiferi	Specifica in quali pagine verrà inserito questo script di contenuto. Segue il formato Match Pattern .
css	Elenco di file CSS da inserire nelle pagine corrispondenti.
js	Elenco di file JS da iniettare in pagine corrispondenti. **Eseguito nell'ordine elencato.**

Gli script di contenuto possono anche essere iniettati su richiesta usando `chrome.tabs.executeScript` , che si chiama Iniezione programmatica .

Guarda anche

- Documentazione ufficiale: contenuti script
- Documentazione Overflow dello stack: script di contenuto

Pagina delle opzioni

Le pagine delle opzioni vengono utilizzate per dare all'utente la possibilità di mantenere le impostazioni per il proprio interno.

Versione 2

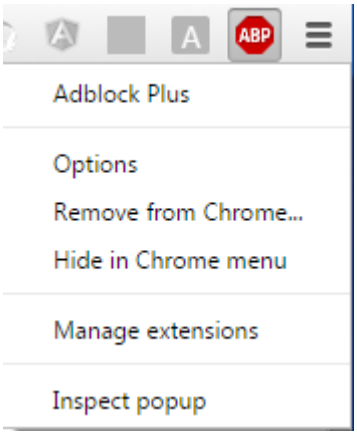
Da Chrome 40 c'è la possibilità di avere la pagina delle opzioni come un dialogo predefinito su *chrome: // extensions* .

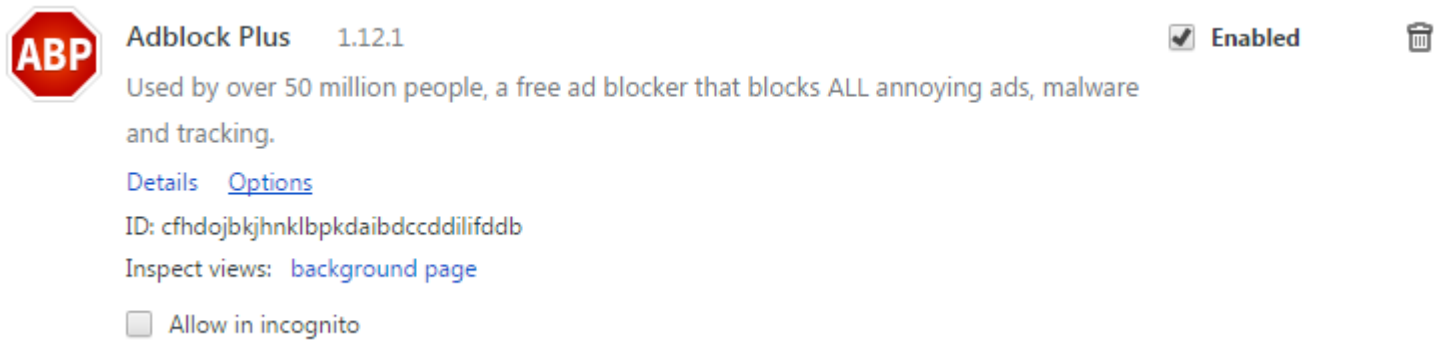
Il modo per definire una pagina di opzioni in `manifest.json` è come il seguente:

```
"options_ui": {
    "page": "options.html",
    "chrome_style": true
}
```

Questa pagina delle opzioni si comporterà come un dialogo, si aprirà come un popup, dove verrà visualizzato il file **options.html** . `chrome_style` applicherà un foglio di stile Chrome per motivi di coerenza dello stile alla pagina delle opzioni.

Le opzioni verranno automaticamente esposte tramite il menu di scelta rapida del **pulsante di estensione** o la pagina **chrome: // extensions** .





Puoi anche aprire la pagina delle opzioni a livello di programmazione , ad esempio da un'interfaccia utente popup:

```
chrome.runtime.openOptionsPage();
```

Versione 1 (deprecata)

Definizione di esempio in `manifest.json` :

```
"options_page": "options.html"
```

Si consiglia di utilizzare la Versione 2 poiché il comportamento `options_ui` verrà presto applicato alle pagine delle opzioni della Versione 1.

Conservazione

Normalmente le impostazioni devono persistere, quindi si consiglia l'uso dell'API `chrome.storage` . Le autorizzazioni possono essere dichiarate in questo modo in `manifest.json` :

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

Documentazione ufficiale

- Pagina Opzioni - Versione 1
- Pagina Opzioni - Versione 2
- API di archiviazione

Crea una nuova scheda

Nel codice dell'estensione puoi utilizzare qualsiasi `chrome.*` API se hai decalcificato le autorizzazioni richieste. Inoltre, alcune API funzionano solo da pagine in background e alcune API funzionano solo da script di contenuto.

Puoi utilizzare la maggior parte dei metodi `chrome.tabs` dichiarando qualsiasi autorizzazione. Ora ci concentriamo su `chrome.tabs.create`

Nota: la nuova scheda verrà aperta senza alcun avviso `popup` .

```
chrome.tabs.create({
       url:"http://stackoverflow.com",
       selected:false  // We open the tab in the background
})
```

Puoi saperne di più su tab object, nello sviluppatore ufficiale di Chrome

Leggi Iniziare con google-chrome-extension online: https://riptutorial.com/it/google-chrome-extension/topic/787/iniziare-con-google-chrome-extension

Capitolo 2: Debug delle estensioni di Chrome

Examples

Utilizzo degli strumenti di sviluppo per eseguire il debug dell'estensione

Un'estensione cromata è separata in un massimo di 4 parti:

- la pagina di sfondo
- la pagina popup
- uno o più script di contenuto
- la pagina delle opzioni

Ogni parte, dal momento che sono innatamente separate, richiede il debug individuale.

Tieni presente che queste pagine sono separate, il che significa che le variabili non sono condivise direttamente tra loro e che un `console.log()` in una di queste pagine non sarà visibile nei registri di altre parti.

Utilizzando gli strumenti di sviluppo chrome:

Le estensioni di Chrome sono simili a debug di altre app web e pagine web. Il debugging viene eseguito più spesso con l'uso dell'ispector di chrome devtools aperto usando la scorciatoia da tastiera rispettivamente per windows e mac: `ctrl` + `shift` + `i` e `cmd` + `shift` + `i` o facendo clic con il tasto destro sulla pagina e selezionando inspect.

Dall'ispettore uno sviluppatore può controllare gli elementi html e il modo in cui il css li influenza, oppure utilizzare la console per controllare i valori delle variabili javascript e leggere gli output da qualsiasi `console.log()` s lo sviluppatore (i) impostato.

Ulteriori informazioni sull'utilizzo dell'ispettore sono disponibili su Chrome Devtools .

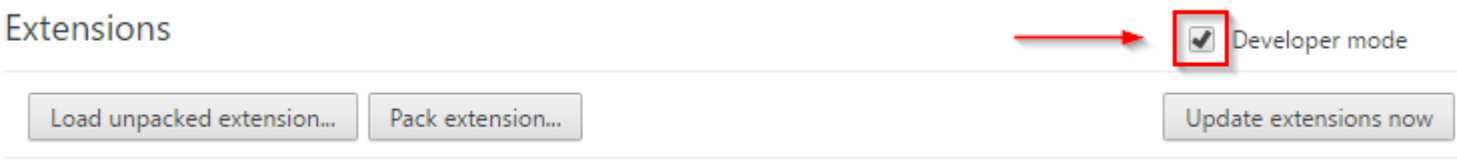
Ispezione del popup, della pagina delle opzioni e di altre pagine accessibili tramite chrome: //.....yourExtensionId.../:

È possibile accedere alla *pagina popup* e alla *pagina delle opzioni* semplicemente controllandole quando sono aperte.

Ulteriori pagine html che fanno parte dell'estensione, ma non sono né il popup né la pagina delle opzioni sono anch'esse debuggate allo stesso modo.

Ispezionando la pagina di sfondo:

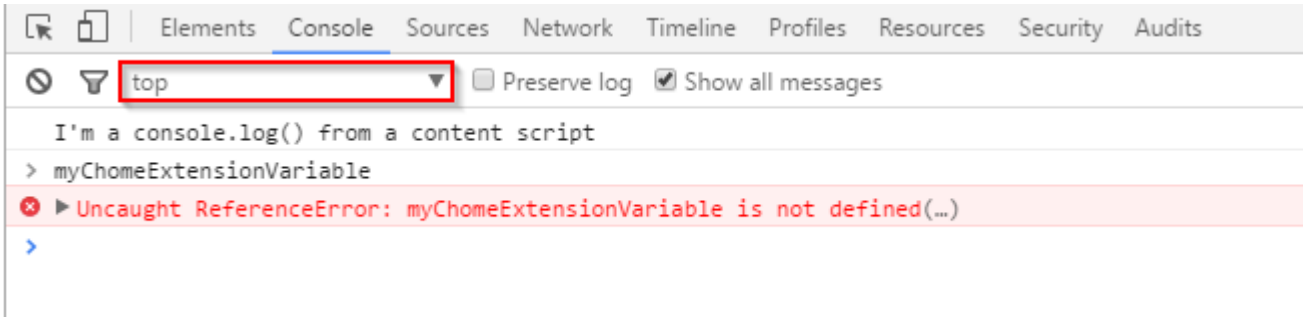
Per accedere alla tua *pagina di sfondo* devi prima accedere alla pagina delle estensioni di Chrome su chrome: // extensions / . Assicurati che il *segno di* spunta *"Modalità sviluppatore"* sia abilitato.



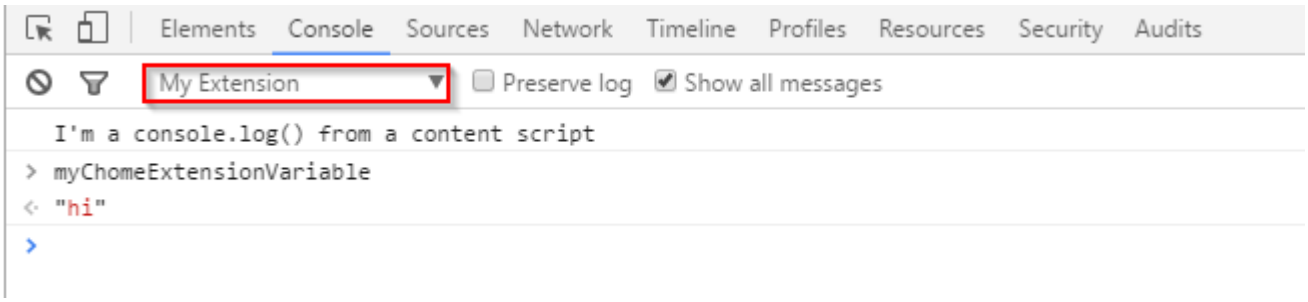
Quindi fai clic sullo script di sfondo accanto a *"Ispeziona viste"* per ispezionare la tua pagina di sfondo.

Ispezione degli script di contenuto:

Gli script di contenuto corrono lungo i siti Web in cui sono stati inseriti. È possibile esaminare lo script del contenuto ispezionando prima il sito Web in cui è inserito lo script del contenuto. Nella console sarà possibile visualizzare tutti i file `console.log()` emessi dall'estensione, ma non sarà possibile modificare o ispezionare le variabili dello script del contenuto.



Per risolvere questo problema, devi fare clic sul menu a discesa solitamente impostato su `'top'` e selezionare la tua estensione dall'elenco di estensioni.



Da lì avrai accesso alle variabili all'interno della tua estensione.

Leggi Debug delle estensioni di Chrome online: https://riptutorial.com/it/google-chrome-extension/topic/5730/debug-delle-estensioni-di-chrome

Capitolo 3: Integrazione degli strumenti per sviluppatori

Examples

Suggerimento programmatico del punto di interruzione

Aggiungi l'istruzione debugger nello script di contenuto

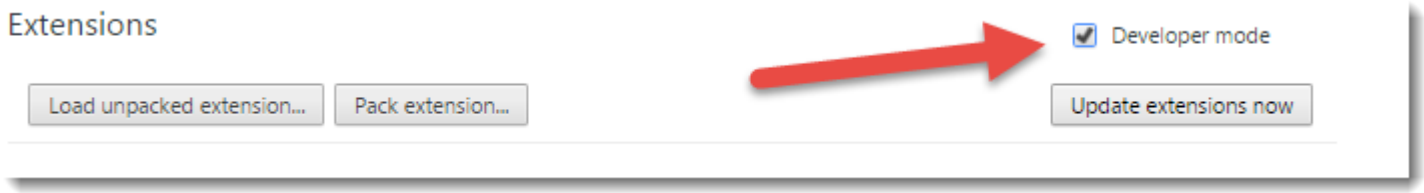
```
var foo = 1;
debugger;

foo = 2;
```

Aprire lo strumento di sviluppo nella pagina Web in cui viene iniettato lo script di contenuto per vedere la pausa dell'esecuzione del codice su tali righe.

Debug della pagina di sfondo / script

Lo script in background è come qualsiasi altro codice JavaScript. Puoi eseguire il debug utilizzando gli stessi strumenti che esegui il debug di altri codici JavaScript in Chrome.

Per aprire gli Strumenti per sviluppatori di Chrome, vai su `chrome://extensions` e attiva la **modalità sviluppatore** :



Ora puoi eseguire il debug di qualsiasi estensione con una pagina di sfondo o uno script. Basta scorrere fino all'estensione che si desidera eseguire il debug e fare clic sul collegamento della **pagina di sfondo** per ispezionarlo.

Suggerimento: per ricaricare l'estensione, puoi premere `F5` all'interno della finestra degli strumenti

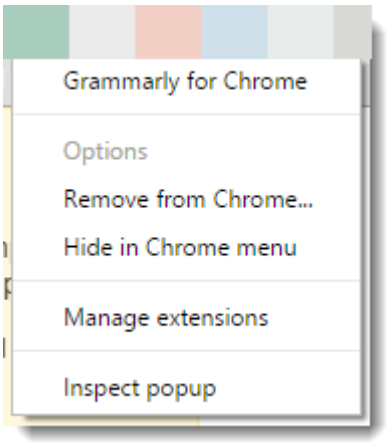
di sviluppo. È possibile inserire punti di interruzione nel codice di inizializzazione prima di ricaricarlo.

Suggerimento: facendo clic con il pulsante destro del mouse sul pulsante di azione dell'estensione e selezionando "Gestisci estensioni" si aprirà la pagina `chrome://extensions` spostata su tale estensione.

Debug della finestra popup

Hai 2 modi per eseguire il debug della finestra popup. Entrambi i modi sono utilizzando Chrome DevTools.

Opzione 1: fare clic con il pulsante destro del mouse sul pulsante di azione dell'estensione e selezionare **Controlla popup**



Opzione 2: apri la finestra popup, direttamente nel browser come una scheda.

Ad esempio, se l'ID dell'estensione è `abcdefghijkmnop` e il tuo file html è `popup.html` . Vai all'indirizzo e vai a:

```
chrome-extension://abcdefghijklmnop/popup.html
```

Ora vedi il poup nella scheda normale. E puoi premere `F12` per aprire gli strumenti di sviluppo.

Leggi Integrazione degli strumenti per sviluppatori online: https://riptutorial.com/it/google-chrome-extension/topic/5938/integrazione-degli-strumenti-per-sviluppatori

Capitolo 4: manifest.json

Osservazioni

Documentazione ufficiale

Formato file manifest

Formato

Il file manifest è scritto nel formato JSON (JavaScript Object Notation).

Questo formato differisce dalle regole più sciolte di scrittura di letterali oggetto nel codice JavaScript. Tra le differenze importanti:

- Ogni nome chiave e stringa letterale **deve essere racchiuso tra virgolette** .
  - Corretto: `"key": "value"`
  - Sbagliato: `key: "value"` , `'key': 'value'`
- **Nessun commento** è consentito dal formato.
  - Sbagliato: `"key": "value" // This controls feature foo`
- Regole virgola rigorose: **elementi separati da virgole, senza virgolette** .
  - Corretta:

```
{
  "foo": "bar",
  "baz": "qux"
}
```

  - Sbagliato (manca la virgola):

```
{
  "foo": "bar"
  "baz": "qux"
}
```

  - Wrong (virgola ciondolante):

```
{
  "foo": "bar",
  "baz": "qux",
```

```
}
```

Examples

Minimo assoluto manifest.json

`manifest.json` fornisce informazioni sull'estensione, come i file più importanti e le funzionalità che l'estensione potrebbe utilizzare. Tra i campi manifest manifestati per le estensioni, sono necessari i **tre** seguenti.

```
{
    "manifest_version": 2,
    "name": "My Extension",
    "version": "1.0"
}
```

Ottenere manifest dal codice di estensione

`chrome.runtime.getManifest()` restituisce manifest dell'estensione sotto forma di oggetto analizzato.

Questo metodo funziona sia su script di contenuto che su tutte le pagine di estensione, non richiede autorizzazioni,

Esempio, ottenendo la stringa della versione dell'estensione:

```
var version = chrome.runtime.getManifest().version;
```

Leggi manifest.json online: https://riptutorial.com/it/google-chrome-extension/topic/948/manifest-json

Capitolo 5: Messaggio in corso

Osservazioni

Documentazione ufficiale

- Messaggio in corso
- Messaggistica nativa
- `chrome.runtime` API (la maggior parte delle funzioni di messaggistica e tutti gli eventi di messaggistica)

Examples

Invia una risposta in modo asincrono

Nel tentativo di inviare una risposta in modo asincrono dal callback `chrome.runtime.onMessage` potremmo provare questo **codice errato** :

```
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
    $.ajax({
        url: 'https://www.google.com',
        method: 'GET',
        success: function(data) {
            // data won't be sent
            sendResponse(data);
        },
    });
});
```

Tuttavia, troveremmo che i `data` non sono mai stati inviati. Ciò accade perché abbiamo inserito `sendResponse` all'interno di una chiamata asincrona ajax, quando il metodo di `success` è stato eseguito, il canale del messaggio è stato chiuso.

La soluzione sarebbe semplice, purché `return true;` esplicitamente `return true;` alla fine del callback, che indica che desideriamo inviare una risposta in modo asincrono, quindi il canale del messaggio sarà tenuto aperto all'altro capo (chiamante) fino a quando `sendResponse` verrà eseguito.

```
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
    $.ajax({
        url: 'https://www.google.com',
        method: 'GET',
        success: function(data) {
            // data would be sent successfully
            sendResponse(data);
        },
    });

    return true; // keeps the message channel open until `sendResponse` is executed
```

```
});
```

Naturalmente, si applica anche a un `return` esplicito dal callback onMessage:

```
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
    if (request.action == 'get') {
        $.ajax({
            url: 'https://www.google.com',
            method: 'GET',
            success: function(data) {
                // data would be sent successfully
                sendResponse(data);
            },
        });

        return true; // keeps the message channel open until `sendResponse` is executed
    }

    // do something synchronous, use sendResponse

    // normal exit closes the message channel
});
```

Leggi Messaggio in corso online: https://riptutorial.com/it/google-chrome-extension/topic/2185/messaggio-in-corso

Capitolo 6: Pagine di sfondo

Examples

Dichiarazione della pagina di sfondo nel manifest

Esistono due modi per registrare una pagina di sfondo nel manifest dell'estensione.

1. La proprietà degli `scripts`

   Nel caso comune, una pagina di sfondo non richiede alcun markup HTML. Possiamo registrare questo tipo di pagine in background usando la proprietà `scripts` .

   In questo caso, verrà generata una pagina in background dal sistema di estensione che include ciascuno dei file elencati nella proprietà degli `scripts` .

   ```
   {
     ...
     "background": {
       "scripts": ["background1.js", "background2.js"],
       "persistent": true
     },
     ...
   }
   ```

2. La proprietà della `page`

   In alcuni casi, potremmo voler specificare l'HTML nella pagina di sfondo, possiamo farlo usando la proprietà della `page` .

   ```
   {
     ...
     "background": {
       "page": "background.html",
       "persistent": true
     },
     ...
   }
   ```

`scripts` VS `page`

È difficile dire quale sia il migliore. potremmo usare la proprietà della `page` e avere alcuni elementi dichiarati nella pagina HTML per uso futuro. Potremmo anche creare dinamicamente tali elementi negli script senza dichiarare esplicitamente la pagina HTML. Tutto dipende dai bisogni reali.

Leggi Pagine di sfondo online: https://riptutorial.com/it/google-chrome-extension/topic/4066/pagine-di-sfondo

Capitolo 7: Porting to / from Firefox

Osservazioni

Se stai utilizzando una versione di *Firefox* prima di 48, avrai anche bisogno di una chiave aggiuntiva in `manifest.json` chiamata applicazioni:

```
"applications": {
  "gecko": {
    "id": "borderify@example.com",
    "strict_min_version": "42.0",
    "strict_max_version": "50.*",
    "update_url": "https://example.com/updates.json"
  }
}
```

applicazioni

Nota:

Firma estensione :

Con il rilascio di Firefox 48, la firma dell'estensione non può più essere disabilitata nelle versioni di rilascio e beta channel utilizzando una preferenza. Come delineato quando è stata annunciata la firma dell'estensione, pubblichiamo build specializzati che supportano questa preferenza in modo che gli sviluppatori possano continuare a testare il codice generato da beta e release build.

Stato di `WebExtensions` :

WebExtensions sono attualmente in uno stato alfa sperimentale. Da Firefox 46, puoi pubblicare WebExtensions per gli utenti di Firefox, proprio come qualsiasi altro componente aggiuntivo. Puntiamo a una prima versione stabile in Firefox 48.

UPD : *Firefox* 48 rilasciato il 02.08.2016.

---

link:

Stato di supporto dell'API - L'elenco delle API e il loro stato.

Incompatibilità Chrome

WebExtensions - API JavaScript, chiavi di manifest.json, tutorial, ecc.

Examples

Porting attraverso WebExtensions

Prima di parlare del porting delle estensioni di *Firefox* da / a, si dovrebbe sapere cosa sono `WebExtensions` .

`WebExtensions` - è una piattaforma che rappresenta un'API per la creazione di estensioni di *Firefox* .

Utilizza la stessa architettura di estensione di *Chromium* , di conseguenza, questa API è compatibile in molti modi con l'API in *Google Chrome* e *Opera* (Opera basata su Chromium). In molti casi, le estensioni sviluppate per questi browser funzioneranno in *Firefox* con alcune modifiche o addirittura senza di esse.

MDN consiglia di utilizzare `WebExtension` per le nuove estensioni:

> In futuro, WebExtensions sarà il modo consigliato per sviluppare componenti aggiuntivi di Firefox e altri sistemi saranno deprecati.

In considerazione di quanto sopra, se si desidera effettuare il porting di estensioni a *Firefox* , è necessario sapere come è stata scritta l'estensione.

Le estensioni per *Firefox* possono essere basate su `WebExtension` , `Add-on SDK` o `XUL` .

Estensioni compatibili basate su WebExtension

Quando si utilizza `WebExtension` , si deve esaminare l'elenco delle incompatibilità , poiché alcune funzioni sono supportate completamente o parzialmente, ovvero in altre parole, si dovrebbe controllare `manifest.json` .

Consente inoltre di utilizzare lo stesso spazio dei nomi :

> Al momento, tutte le API sono accessibili tramite lo spazio dei nomi chrome. *. Quando iniziamo ad aggiungere le nostre API, ci aspettiamo di aggiungerle al browser. * Namespace. Gli sviluppatori saranno in grado di utilizzare il rilevamento delle funzioni per determinare se un'API è disponibile nel browser. *.

Una semplice estensione che può funzionare in Firefox e Google Chrome

`manifest.json` :

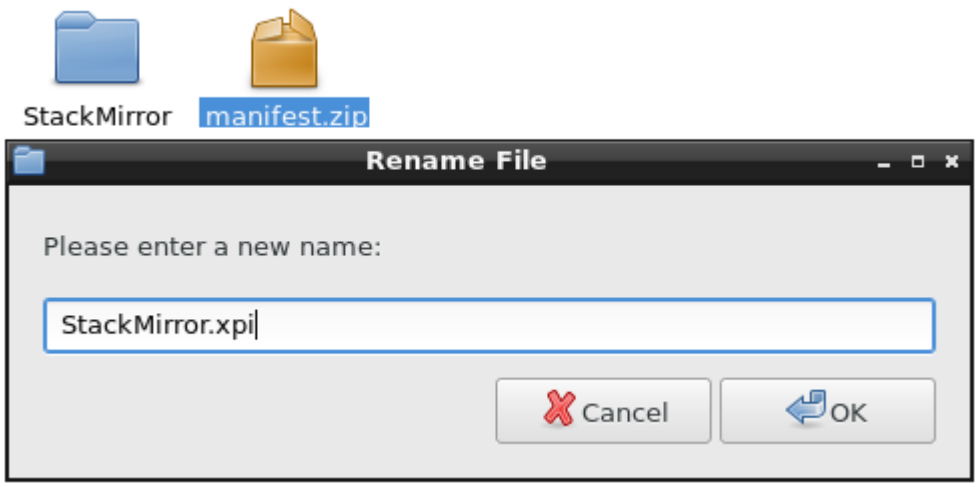
```
{
  "manifest_version": 2,

  "name": "StackMirror",
```

```
  "version": "1.0",

  "description": "Mirror reflection of StackOverflow sites",

  "icons": {
    "48": "icon/myIcon-48.png"
  },

  "page_action": {
    "default_icon": "icon/myIcon-48.png"
  },

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

  "permissions": ["tabs", "*://*.stackoverflow.com/*"]
}
```

sceneggiatura di `background` :

```
function startScript(tabId, changeInfo, tab) {

    if (tab.url.indexOf("stackoverflow.com") > -1) {

        chrome.tabs.executeScript(tabId,

            {code: 'document.body.style.transform = "scaleX(-1)";'}, function () {

            if (!chrome.runtime.lastError) {

                chrome.pageAction.show(tabId);
            }
        });
    }
}

chrome.tabs.onUpdated.addListener(startScript);
```

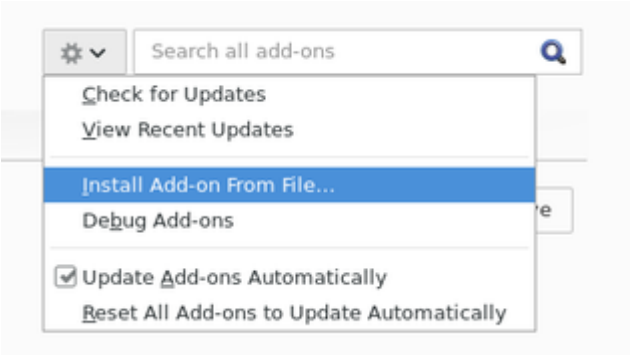
Comprimere il progetto come file `zip` standard, ma con estensioni `.xpi` .



Quindi, devi caricare l'estensione in *Firefox* .

Apri la pagina `about:addons` , accessibile tramite **Menu> Componenti aggiuntivi** .

Fai clic sul pulsante **Strumenti per tutti i componenti aggiuntivi** .



Quando l'estensione è caricata la pagina `about:addons` sarà simile a questa:

Le indicazioni sul caricamento dell'estensione in Google Chrome sono in un altro argomento: come iniziare con Chrome Extensions .

Il risultato dell'operazione di estensione sarà lo stesso in entrambi i browser (*Firefox* / *Google Chrome*):

Se il componente aggiuntivo corrente è basato su Add-on SDK o XUL

Quando l'estensione in fase di porting è basato su `Add-on SDK` si deve guardare attraverso il tavolo di confronto per `Add-on SDK` **=>** `WebExtensions` , perché queste tecnologie hanno caratteristiche simili, ma differiscono per l'attuazione. Ogni sezione della tabella descrive l'equivalente di `Add-on`

`SDK` per `WebExtension` .

Confronto con l'SDK add-on

Un approccio simile e per le estensioni XUL.

Confronto con le estensioni XUL / XPCOM

Leggi Porting to / from Firefox online: https://riptutorial.com/it/google-chrome-extension/topic/5731/porting-to---from-firefox

Capitolo 8: Script di contenuto

Osservazioni

Documentazione ufficiale

- Script di contenuti
- Politica di sicurezza del contenuto> Script di contenuti

Examples

Dichiarare gli script di contenuto nel manifest

Gli script di contenuto possono essere dichiarati in `manifest.json` per essere sempre iniettati in pagine che corrispondono a un set di pattern URL .

Esempio minimo

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

Questa voce manifest indica a Chrome di inserire uno script di contenuto `content.js` , insieme al file CSS `content.css` , dopo qualsiasi navigazione verso una pagina che corrisponde al modello di corrispondenza `http://example.com/*`

Le chiavi `js` e `css` sono opzionali: puoi averne solo una o entrambe, a seconda di cosa ti serve.

`content_scripts` chiave `content_scripts` è una matrice e puoi dichiarare diverse definizioni di script del contenuto:

```
"content_scripts" : [
  {
    "js": ["content.js"],
    "matches": ["http://*.example.com/*"]
  },
  {
    "js": ["something_else.js"],
    "matches": ["http://*.example.org/*"]
  }
]
```

Nota che sia `js` che le `matches` sono matrici, anche se hai solo una voce.

Altre opzioni sono disponibili nella documentazione ufficiale e altri esempi.

Nota importante

Gli script di contenuto dichiarati nel manifest **verranno iniettati solo nelle nuove navigazioni dopo il caricamento dell'estensione** . Non verranno iniettati nelle schede esistenti. Questo vale anche per le ricariche degli interni durante lo sviluppo e gli aggiornamenti delle estensioni dopo il rilascio.

Se è necessario assicurarsi che le schede attualmente aperte siano coperte, prendere in considerazione anche l'iniezione programmatica all'avvio.

Iniezione di script di contenuto da una pagina di estensione

Se, invece di avere sempre uno script di contenuto immesso in base all'URL, si desidera controllare direttamente quando viene iniettato uno script di contenuto, è possibile utilizzare l' iniezione programmatica .

Esempio minimo

- JavaScript

```
chrome.tabs.executeScript({file: "content.js"});
```

- CSS

```
chrome.tabs.insertCSS({file: "content.css"});
```

Chiamato da una pagina di estensione (ad esempio sfondo o popup) e assumendo che si abbia il permesso di iniettare, questo eseguirà `content.js` o inserirà `content.css` come script di contenuto nel frame superiore della scheda corrente.

Codice inline

È possibile eseguire il codice inline anziché un file come script di contenuto:

```
var code = "console.log('This code will execute as a content script');";
chrome.tabs.executeScript({code: code});
```

Scegliere la scheda

È possibile fornire un ID di tabulazione (in genere di altri metodi `chrome.tabs` o di messaggistica) da eseguire in una scheda diversa da quella attualmente attiva.

```
chrome.tabs.executeScript({
  tabId: tabId,
  file: "content.js"
});
```

Altre opzioni sono disponibili nella documentazione `chrome.tabs.executeScript()` e in altri esempi.

permessi

L'utilizzo di `chrome.tabs.executeScript()` non richiede l'autorizzazione `"tabs"` , ma richiede le autorizzazioni dell'host per l'URL della pagina.

Controllo degli errori

Se l'iniezione di script fallisce, è possibile prenderla nel callback opzionale:

```
chrome.tabs.executeScript({file: "content.js"}, function() {
  if(chrome.runtime.lastError) {
    console.error("Script injection failed: " + chrome.runtime.lastError.message);
  }
});
```

Più script di contenuto nel manifest

Stesse condizioni, più script

Se è necessario iniettare più file con tutte le altre condizioni uguali, ad esempio per includere una libreria, è possibile elencarli tutti nella matrice `"js"` :

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

Questioni di ordine: `library.js` verrà eseguito prima di `content.js` .

Stessi script, più siti

Se è necessario iniettare gli stessi file in più siti, è possibile fornire più modelli di corrispondenza:

```
"matches": ["http://example.com/*", "http://example.org/*"]
```

Se devi inserire praticamente ogni pagina, puoi utilizzare pattern di corrispondenza generica come

`"*://*/*"` (corrisponde a ogni pagina HTTP (S)) o `"<all_urls>"` (corrisponde a tutte le pagine supportate).

Script diversi o siti diversi

`"content_scripts"` sezione `"content_scripts"` è anche una matrice, quindi è possibile definire più di un blocco di script di contenuto:

```
"content_scripts" : [
  {
    "js": ["content.js"],
    "matches": ["http://*.example.com/*"]
  },
  {
    "js": ["something_else.js"],
    "matches": ["http://*.example.org/*"]
  }
]
```

Leggi Script di contenuto online: https://riptutorial.com/it/google-chrome-extension/topic/2850/script-di-contenuto

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con google-chrome-extension	Aminadav, Community, Deliaz, Haibara Ai, ScientiaEtVeritas, Xan
2	Debug delle estensioni di Chrome	Marc Guiselin
3	Integrazione degli strumenti per sviluppatori	Aminadav, Paul Sweatte, Xan
4	manifest.json	Haibara Ai, Xan
5	Messaggio in corso	Haibara Ai, wOxxOm, Xan
6	Pagine di sfondo	Haibara Ai, Noam Hacker
7	Porting to / from Firefox	Deliaz, UserName
8	Script di contenuto	Haibara Ai, Xan