Manage Events with Background Scripts

Background.js 文件说明

Extensions are event based programs used to modify or enhance the Chrome browsing experience. Events are browser triggers, such as navigating to a new page, removing a bookmark, or closing a tab. Extensions monitor these events in their background script, then react with specified instructions.

A background page is loaded when it is needed, and unloaded when it goes idle. Some examples of events include:

- The extension is first installed or updated to a new version.
- The background page was listening for an event, and the event is dispatched.
- A content script or other extension **sends a message.**
- Another view in the extension, such as a popup, calls `runtime.getBackgroundPage`.

Once it has been loaded, a background page will stay running as long as it is performing an action, such as calling a Chrome API or issuing a network request. Additionally, the background page will not unload until all visible views and all message ports are closed. Note that opening a view does not cause the event page to load, but only prevents it from closing once loaded.

Effective background scripts stay dormant until an event they are listening for fires, react with specified instructions, then unload.

Register Background Scripts

Background scripts are registered in the **manifest** under the `"background"` field. They are listed in an array after the `"scripts"` key, and `"persistent"` should be specified as false.

```
{
  "name": "Awesome Test Extension",
  ...
  "background": {
    "scripts": ["background.js"],
    "persistent": false
  },
  ...
}
```

Multiple background scripts can be registered for modularized code.

```
  {
     "name": "Awesome Test Extension",
     ...
     "background": {
       "scripts": [
         "backgroundContextMenus.js",
         "backgroundOmniBox.js",
         "backgroundOauth.js"
       ],
       "persistent": false
     },
     ...
   }
```

The only occasion to keep a background script persistently active is if the extension uses **chrome.webRequest** API to block or modify network requests. The webRequest API is incompatible with non-persistent background pages.

If an extension currently uses a persistent background page, refer to **Background Migration Guide** for instruction on how to switch to a non-persistent model.

Initialize the Extension

Listen to the **`runtime.onInstalled`** event to initialize an extension on installation. Use this event to set a state or for one-time initialization, such as a **context menu**.

```
chrome.runtime.onInstalled.addListener(function() {
  chrome.contextMenus.create({
    "id": "sampleContextMenu",
    "title": "Sample Context Menu",
    "contexts": ["selection"]
  });
});
```

Set Up Listeners

Structure background scripts around events the extension depends on. Defining functionally relevant events allows background scripts to lie dormant until those events are fired and prevents the extension from missing important triggers.

Listeners must be registered synchronously from the start of the page.

```
chrome.runtime.onInstalled.addListener(function() {
  chrome.contextMenus.create({
    "id": "sampleContextMenu",
    "title": "Sample Context Menu",
    "contexts": ["selection"]
  });
});

// This will run when a bookmark is created.
chrome.bookmarks.onCreated.addListener(function() {
  // do something
});
```

Do not register listeners asynchronously, as they will not be properly triggered.

```
chrome.runtime.onInstalled.addListener(function() {
  // ERROR! Events must be registered synchronously from the start of
```

```
  // the page.
  chrome.bookmarks.onCreated.addListener(function() {
    // do something
  });
});
```

Extensions can remove listeners from their background scripts by calling `removeListener`. If all listeners for an event are removed, Chrome will no longer load the extension's background script for that event.

```
chrome.runtime.onMessage.addListener(function(message, sender, reply) {
    chrome.runtime.onMessage.removeListener(event);
});
```

Filter Events

Use APIs that support **event filters** to restrict listeners to the cases the extension cares about. If an extension is listening for the **`tabs.onUpdated`** event, try using the **`webNavigation.onCompleted`** event with filters instead, as the tabs API does not support filters.

```
chrome.webNavigation.onCompleted.addListener(function() {
    alert("This is my favorite website!");
}, {url: [{urlMatches : 'https://www.google.com/'}]});
```

React to Listeners

Listeners exist to trigger functionality once an event has fired. To react to an event, structure the desired reaction inside of the listener event.

```
chrome.runtime.onMessage.addListener(function(message, callback) {
  if (message.data == “setAlarm”) {
    chrome.alarms.create({delayInMinutes: 5})
  } else if (message.data == “runLogic”) {
    chrome.tabs.executeScript({file: 'logic.js'});
  } else if (message.data == “changeColor”) {
    chrome.tabs.executeScript(
        {code: 'document.body.style.backgroundColor="orange"'});
  };
});
```

Unload Background Scripts

Data should be persisted periodically so that important information is not lost if an extension crashes without receiving onSuspend. Use the **storage** API to assist with this.

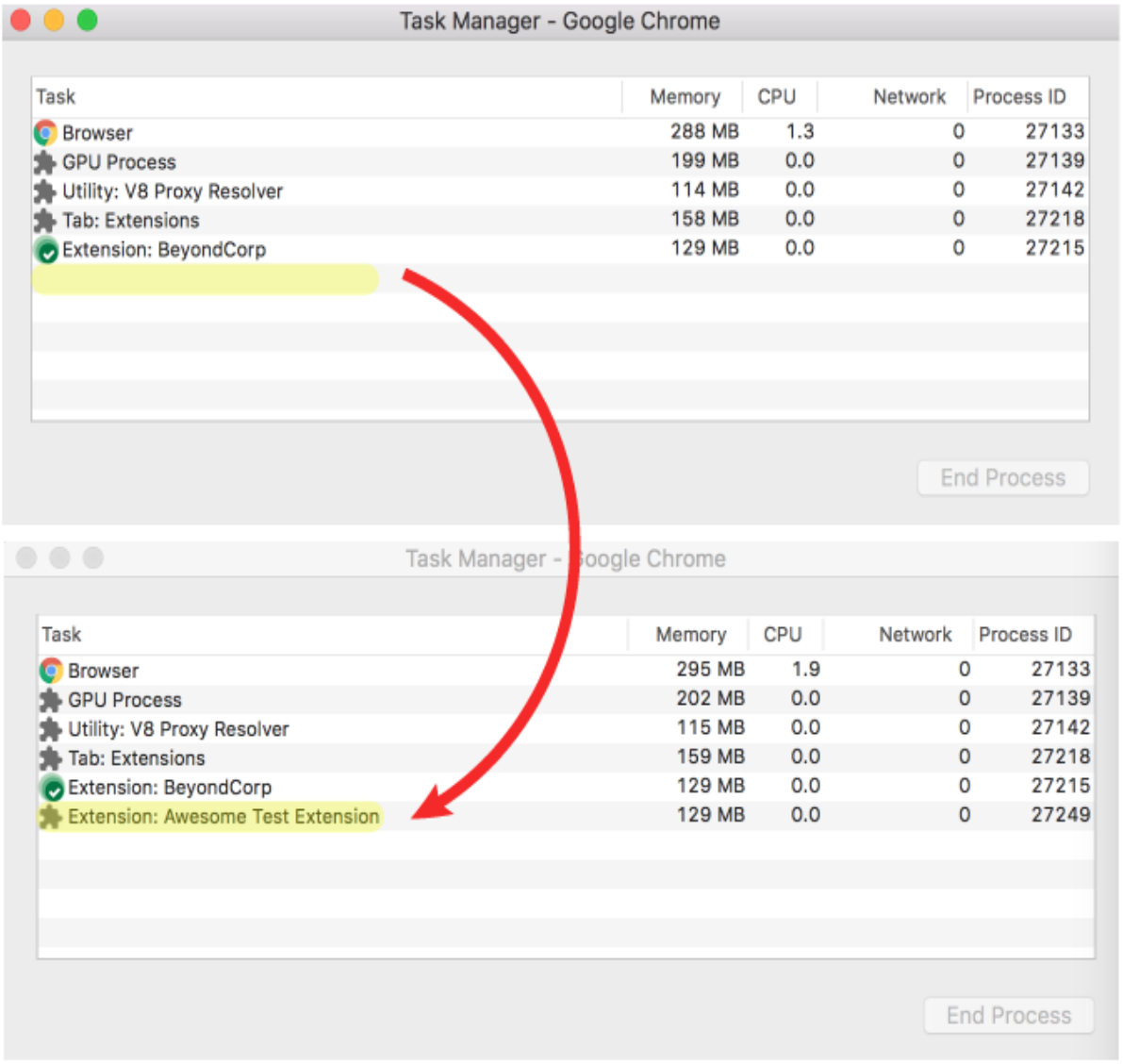
```
chrome.storage.local.set({variable: variableInformation});
```

If an extension uses **message passing**, ensure all ports are closed. The background script will not unload until all message ports have shut. Listening to the **runtime.Port.onDisconnect** event will give insight to when open ports are closing. Manually close them with **runtime.Port.disconnect**.

```
chrome.runtime.onMessage.addListener(function(message, callback) {
  if (message == 'hello') {
    sendResponse({greeting: 'welcome!'})
  } else if (message == 'goodbye') {

    chrome.runtime.Port.disconnect();
  }
});
```

The lifetime of a background script is observable by monitoring when an entry for the extension appears and disappears from Chrome's task manager.



Open the task manager by clicking the Chrome Menu, hovering over more tools and selecting "Task Manager".

Background scripts unload on their own after a few seconds of inactivity. If any last minute cleanup is required, listen to the `runtime.onSuspend` event.

```
chrome.runtime.onSuspend.addListener(function() {
  console.log("Unloading.");
  chrome.browserAction.setBadgeText({text: ""});
});
```

However, persisting data should be prefered over relying on `runtime.onSuspend`. It doesn't allow for as much cleanup as may be needed and will not help in case of a crash.