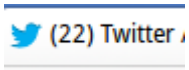
How to detect page title change in Google Chrome from an extension?

Asked 10 years, 8 months ago Modified 2 years, 10 months ago Viewed 8k times

11

I'm creating a Google Chrome extension and I need to detect when a page's title changes. The page's title is changed like in Twitter: `(num) Twitter` (see the screenshot below) - when a new tweet is posted, the number increments. Example:



I'm trying to detect the title changes of a URL that's loaded in one of my tabs and play a beep sound whenever there's a difference. This check is to be done in a repeated interval and I think that can be accomplished using `setTimeOut()` function.

I've created a `manifest.json` as follows:

```
{
  "manifest_version": 2,

  "name": "Detect Page Title Changes",
  "description": "Blah",
  "version": "1.0",

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

However, I'm clueless about the rest. I've searched through the docs [1] [2] and tried the solutions on similar Stack Overflow threads such as this one I but couldn't find anything that suits my requirements.

Do you have any suggestions? Please include an example, if possible.

javascript google-chrome google-chrome-extension

Share Edit Follow

edited May 23, 2017 at 12:10
Community Bot
1 • 1

asked Jul 27, 2013 at 5:28

Amal Murali
76.2k • 18 • 130 • 151

Do you need events if you're polling? Can't you just store the last tab title and compare it to the current? – nrabinowitz Jul 27, 2013 at 6:01

I think it'd be a good idea to use an event (if one is available). But if there isn't one, your idea can be used, as well. – Amal Murali Jul 27, 2013 at 6:06

I'm baffled by the downvotes; it would be nice for the downvoter to explain why he/she thinks this is an unhelpful and/or incorrect response so I can, perhaps, improve it :) – Amal Murali Mar 10, 2014 at 13:44

possible duplicate of Detect change in document title via Javascript, and the accepted answer there is better. – Xan Apr 10, 2014 at 0:18

@Xan: I don't know where you got the impression that's a duplicate. Maybe judging by the title? It's ***not*** a duplicate. Please read the whole question again. – Amal Murali Apr 10, 2014 at 4:59

4 Answers

Sorted by: Highest score (default)

18

Instead of arguing in comments that a certain approach is better, let me be more constructive and add an answer by showing a particular implementation I co-wrote myself, **and explain some gotchas** you may run into. **Code snippets refer to a service different from Twitter, but the goal was the same.** In fact, this code's goal is to report the exact number of unread messages, so yours might be simpler.

My approach is based on an answer here on SO, and instead of being **polling-driven** (check condition at fixed intervals) is **event-driven** (be notified of potential changes in condition).

Advantages include immediate detection of a change (which would otherwise not be detected until the next poll) and not wasting resources on polls while the condition does not change. Admittedly, the second argument hardly applies here, but the first one still stands.

+100

Architecture at a glance:

1. Inject a content script into the page in question.
2. Analyze initial state of the title, report to background page via `sendMessage`.
3. Register a handler for a title change event.
4. Whenever the event fires and the handler is called, analyze the new state of the title, report to background page via `sendMessage`.

Already step 1 has a gotcha to it. Normal content script injection mechanism, when the content script is defined in the manifest, will inject it in pages upon navigation to a page that matches the URL.

```
"content_scripts": [
  {
    "matches": [
      "*://theoldreader.com/*"
    ],
    "js": ["observer.js"],
    "run_at": "document_idle"
  }
]
```

This works pretty well, *until your extension is reloaded*. This can happen in development as you're applying changes you've made, or in deployed instances as it is auto-updated. What happens then is that content scripts are **not** re-injected in existing open pages (until navigation happens, like a reload). Therefore, if you rely on manifest-based injection, you should also consider including programmatic injection into already-open tabs when extension initializes:

```
function startupInject() {
  chrome.tabs.query(
    {url: "*://theoldreader.com/*"},
    function (tabs) {
      for (var i in tabs) {
        chrome.tabs.executeScript(tabs[i].id, {file: "observer.js"});
      }
    }
  );
}
```

On the other end, content script instances that were active at the time of extension reload **are not terminated, but are orphaned**: any `sendMessage` or similar request will fail. It is, therefore, recommended to always check for exceptions when trying to communicate with the parent extension, and self-terminate (by removing handlers) if it fails:

```
try {
  chrome.runtime.sendMessage({'count' : count});
} catch(e) { // Happens when parent extension is no longer available or was reloaded
  console.warn("Could not communicate with parent extension, deregistering observer");
  observer.disconnect();
}
```

Step 2 also has a gotcha to it, though it depends on the specifics of the service you're watching. Some pages inside the scope of the content script will not show the number of unread items, but it does not mean that there are no new messages.

After observing how the web service works, I concluded that if the title changes to something without navigation, it's safe to assume the new value if correct, but for the initial title "no new items" should be ignored as unreliable.

So, the analysis code accounts for whether it's the initial reading or handling an update:

```
function notify(title, changed) {
  // ...
  var match = /^\((\d+)\)/.exec(title);
  var match_zero = /^The Old Reader$/.exec(title);

  if (match && match[1]) {
    count = match[1];
  } else if (match_zero && changed) {
    count = 0;
  }
  // else, consider that we don't know the count
  //...
}
```

It is called with the initial title and `changed` = `false` in step 2.

Steps 3 & 4 are the main answer to "how to watch for title changes" (in an event-driven way).

```
var target = document.querySelector('head > title');

var observer = new window.MutationObserver(
  function(mutations) {
    mutations.forEach(
      function(mutation){
        notify(mutation.target.textContent, true);
      }
    );
  }
);

observer.observe(target, { subtree: true, characterData: true, childList: true });
```

For specifics as to why certain options of `observer.observe` are set, see the original answer.

Note that `notify` is called with `changed` = `true`, so going from "(1) The Old Reader" to "The Old Reader" without navigation is considered to be a "true" change to zero unread messages.

Share Edit Follow

edited May 23, 2017 at 10:31
Community Bot
1 • 1

answered Apr 10, 2014 at 13:29
Xan
76.1k • 16 • 187 • 209

@AmalMurali `observer.js` is almost fine as it is, just modify URLs and title match patterns to fit your web service. Forget for now about `storage.js` and how the options page works, cut them away, and you can throw away large parts of background code (`background.js` and `functions.js`) that deal with things other than `onMessage` . The rest - you'll need to implement your own logic for dealing with reported count, and adapt `manifest.json` . That's a starting point, good luck. – Xan Apr 10, 2014 at 15:58

1 Nice answer, one suggestion: Use the unprefixed `window.MutationObserver` . You only need to add `window.MutationObserver=window.MutationObserver||window.WebKitMutationObserver;` if you want to support Chrome 26 or earlier, but given that the Chrome Web Store now refuses to install extensions in Chrome 30 or earlier, you're probably fine with only using the unprefixed API. – Rob W Jun 1, 2014 at 8:38

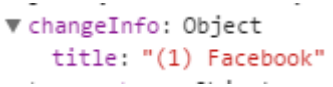
@RobW nice catch. In fact, I noticed it, fixed it in the codebase this was taken from, and then forgot to update here. – Xan Jun 1, 2014 at 11:15

11

Put `chrome.tabs.onUpdated.addListener` in your background script:

```
chrome.tabs.onUpdated.addListener(function(tabId, changeInfo, tab) {
   console.log(changeInfo);
});
```

`changeInfo` is an object which includes title changes, e.g. here:



Can then filter on the object so that an action only occurs if `changeInfo` includes a title change. For additional manipulation, e.g. responding to page title changes with page content / actions, you can send a message to content script from inside the listener after whatever conditions are met.

Share Edit Follow

edited Jun 6, 2021 at 4:50
andromeda947
1,043 ● 12 ● 11

answered Oct 20, 2016 at 11:55
lou1989
337 ● 2 ● 8

6 This is the best answer by far. Very underrated. – Daniel van der Merwe Nov 29, 2016 at 16:57

9

1. Create an event page.
2. Create a content script that gets injected into a webpage when a webpage loads.
3. Within the content script, use `setInterval` to poll the page to see if `window.document.title` changes.
4. If the title has changed, use `chrome.runtime.sendMessage` to send a message to your event page.
5. On your event page, listen for messages with `chrome.runtime.onMessage` and play a sound.

Share Edit Follow

edited May 23, 2017 at 12:10
Community Bot
1 ● 1

answered Jul 27, 2013 at 8:18
Chris McFarland
6,119 ● 5 ● 44 ● 63

7

After researching Chrome's tabs API, it doesn't look like anything stands out to help you directly. However, you should be able to attach an event listener to the title node of the tab(s) you're interested in. The DOMSubtreeModified mutation event works in Chrome, and a quick test in a normal html document proves to work for me - should be no different from within an extension.

```
var title = document.getElementsByTagName('title')[0];

if (title) {
    title.addEventListener('DOMSubtreeModified', function (e) {
        // title changed
    }, false);
}
```

Share Edit Follow

answered Jul 27, 2013 at 16:24

user2609094

1 It's an old answer; since then, `DOMSubtreeModified` events are considered deprecated. – Xan Jun 1, 2014 at 11:16