



Développement par composants (d'extension) Web

Intervenant : Chouki TIBERMACHINE

Bureau : LIRMM (E.311)

Tél. : 04.67.14.97.24

Mél. : Chouki.Tibermachine@lirmm.fr

Web : <https://github.com/ctiber/composants/>

Plan du cours

- Mécanisme des WebExtensions
- Quelques APIs fournies par les navigateurs

Plan du cours

- Mécanisme des WebExtensions
- Quelques APIs fournies par les navigateurs

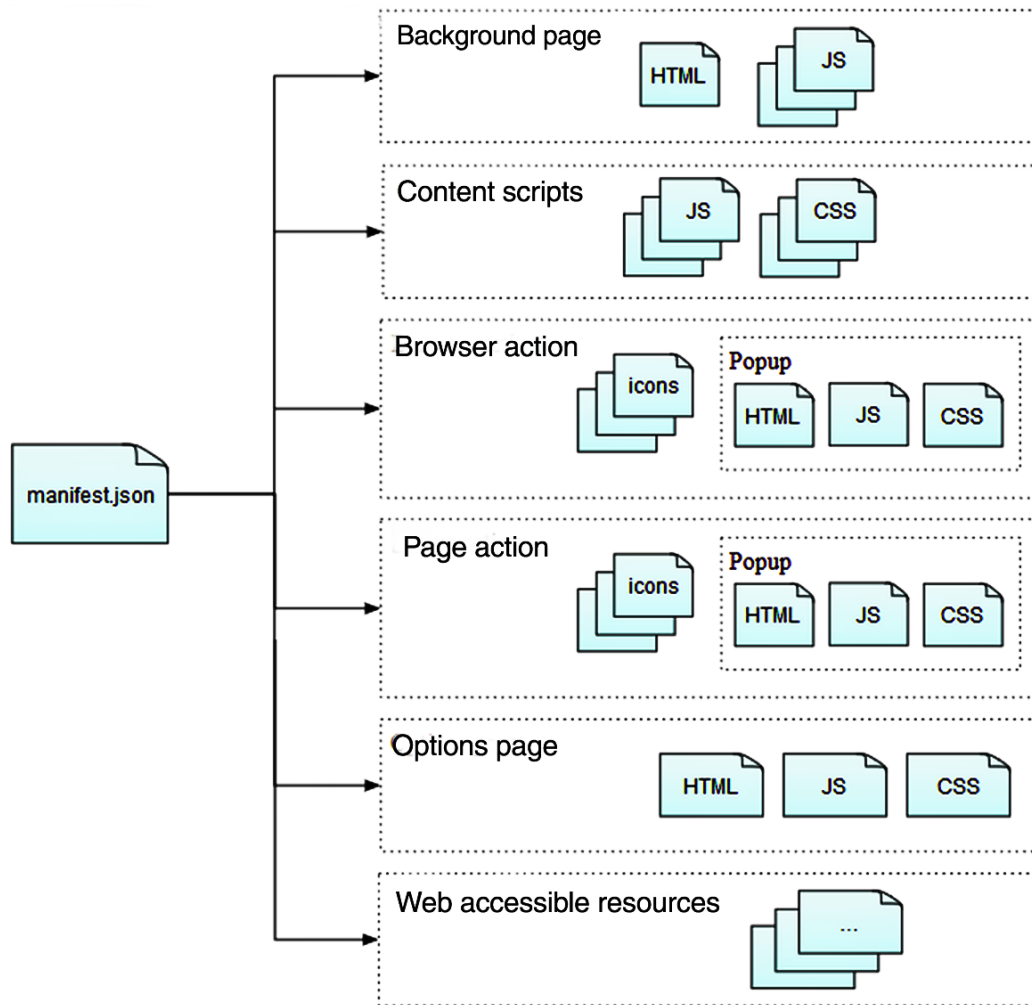
Principe général

- Le mécanisme des extensions Web (*WebExtensions*) est un système multi-navigateurs pour développer des plugins pour enrichir les fonctionnalités des navigateurs
- Théoriquement, un même plugin (composant) devrait fonctionner sur Firefox, Chrome et Opera, et sur Microsoft Edge (moyennant quelques petits changements)
- Ces plugins sont écrits avec les standards du Web : HTML, CSS et JavaScript (avant, il y avait des langages dédiés)

Applications possibles

- Changer l'apparence de certains sites Web (les adapter aux utilisateurs ayant un handicap – l'accessibilité, par exemple)
- Changer le contenu de certains sites Web (supprimer les messages publicitaires, par exemple, comme Adblock)
- Archiver toutes les navigations effectuées par l'utilisateur et les ré-exécuter automatiquement à sa demande
- Télécharger les médias (vidéos, par exemple) qui sont visionnés sur le navigateur (comme Download Helper)
- ...

Structure d'un plugin



Implémentent la logique métier du plugin

Scriptent le comportement et l'apparence des pages Web visitées

Ajoutent des boutons à la barre d'outils du navigateur (actions permanentes)

Ajoutent des boutons à la barre d'adresses du navigateur (actions propres à une page)

Ajoutent une page pour permettre aux utilisateurs de paramétrer le plugin

Grouper les ressources (images, par ex) utilisées par le plugin

Le manifest d'un plugin

- C'est le seul fichier obligatoire
- Il décrit :
 - le nom et la version du plugin
 - les permissions demandées au (requis du) navigateur
- Il pointe vers les autres fichiers et répertoires qui composent le plugin
- "manifest_version", "version", et "name" sont obligatoires

Exemple de manifest (developer.mozilla.org)

```
{
  "manifest_version": 2,
  "name": "Borderify",
  "version": "1.0",

  "description": "Adds a solid red border to all webpages matching mozilla.org.",
  "icons": {
    "48": "icons/border-48.png"
  },

  "content_scripts": [
    {
      "matches": ["*://*.mozilla.org/*"],
      "js": ["borderify.js"]
    }
  ]
}
```

L'icône qui va apparaître dans la liste des plugins du navigateur

Le script qui va s'exécuter lorsque le motif indiqué dans *matches* correspond à l'URL de la page visitée par l'utilisateur

Les script JS dans content_scripts

- Le code du script de l'exemple précédent (très simple) :
`document.body.style.border = "5px solid red";`
- Ce sont des scripts qui sont chargés dans le contexte de la page affichée par le navigateur
- Ils permettent de scripter le contenu ces pages Web
Dans l'exemple : changer la bordure du body du document

Démo du premier plugin

- Il faut avoir cette arborescence

borderify/manifest.json

borderify/borderify.js

borderify/icons/border-48.png

- Sur Firefox :

Ouvrir l'URL : about:debugging

Cliquer sur « Charger un module temporaire »

Sélectionner un fichier dans le répertoire du plugin (le manifest par ex.)

- Sur Chrome :

Aller dans le menu Outils (Tools) > Extensions

Cliquer sur « Charger l'extension non empaquetée ... »

Sélectionner le répertoire où se trouve le plugin

- Ouvrir un nouvel onglet et consulter la page : developer.mozilla.org

- Pour dés-installer un plugin sur Firefox, aller dans Outils > Modules complémentaires > Supprimer ou Désactiver

Les content scripts

- Ils agissent comme les scripts `<script>` placés dans la page mais :
 - N'ont pas un accès direct aux autres scripts de la page (leurs variables et fonctions, les bibliothèques qu'ils chargent, ...)
- Contrairement aux scripts de la page :
 - Ils peuvent faire des requêtes XHR (cross-origin)
 - Ils peuvent utiliser certaines API WebExtension (voir plus loin)
 - Ils peuvent échanger des messages avec les background scripts
- Noter qu'il y a 3 sortes de scripts : page, content, background
- Il est possible de mettre des fichiers CSS à la place des scripts JS dans les `content_scripts`

Les background scripts

- Ils effectuent un traitement sur le long terme (ne se limitant pas à une page ou un contenu particulier visité par l'utilisateur)
- Ils s'exécutent dès que le plugin est installé
- Ils sont indiqués dans le manifest :
`// manifest.json`
`"background": { "scripts": ["background-script.js"] }`
- L'environnement de ces scripts est une page d'arrière plan, accessible via l'objet window et son API DOM
- Il est possible de préciser une page précise comme background :
`"background": { "page": "background-page.html" }`

Les background scripts -suite-


- Ces scripts ont accès à de nombreuses APIs WebExtension
- Ils peuvent faire des requêtes XHR
- Si dans une extension, on définit une action dans le navigateur (bouton dans la barre d'outils), et si on n'a pas défini de popup, il est possible d'écouter les clics sur ce bouton :
`chrome.browserAction.onClicked.addListener(handleClick);`
- Les background scripts n'ont pas un accès direct aux contenus des pages visités par les utilisateurs, mais ils peuvent échanger des messages avec les content scripts et leur demander de s'exécuter

Exemple de plugin avec background script

- Le manifest.json :

```
{  
  "description": "Demonstrating webRequests",  
  "manifest_version": 2,  
  "name": "webRequest-demo",  
  "version": "1.0",  
  "permissions": ["webRequest", "webRequestBlocking"],  
  "background": {"scripts": ["background.js"]}  
}
```

Permissions requises
au navigateur



Le background script (site MDN)

- Remplace toutes les images dans les pages dont l'URL match avec la pattern (urls) par l'image suivante : 

```
// add the listener, passing the filter argument and "blocking"
chrome.webRequest.onBeforeRequest.addListener(
  redirect, // callback
  {urls:["http://www.lirmm.fr/*"], types:["image"]}, // filter
  ["blocking"] // extraInfoSpec
);
// redirect function
// returns an object with a property 'redirectURL' set to the new URL
function redirect(requestDetails) {
  console.log("Redirecting: " + requestDetails.url);
  return {
    redirectUrl: "http://image.shutterstock.com/z/stock-vector-vector-skull-danger-sign-120892354.jpg"
  };
}
```

Actions avec popups

- Lorsqu'on a une action pour laquelle on définit un popup, l'événement onClicked n'est pas propagé (click = afficher ou masquer le popup)

- Exemple :

```
"browser_action": {  
  "default_icon": {  
    "19": "button/geo-19.png",  
    "38": "button/geo-38.png"  
  },  
  "default_title": "Whereami?",  
  "default_popup": "popup/geo.html"  
}
```

Différents formats d'icônes selon l'affichage du bouton (dans la barre d'outil ou dans le menu hamburger) et selon la taille et la résolution de l'écran

Une page Web ordinaire qui peut contenir du CSS et du JavaScript, qui écoute les événements qui se produisent sur cette page

Communication entre content et background scripts

- Deux manières différentes : envoi de message simple ou connexion durable
- Envoi de messages simples :

	In content script	In background script
Send a message	<code>chrome.runtime.sendMessage()</code>	<code>chrome.tabs.sendMessage()</code>
Receive a message	<code>chrome.runtime.onMessage</code>	<code>chrome.runtime.onMessage</code>

- Exemple : extension sur GitHub qui notifie les clicks sur les liens

```
// content-script.js
window.addEventListener("click", notifyExtension);
function notifyExtension(e) {
  if (e.target.tagName !== "A") {
    return;
  }
  chrome.runtime.sendMessage({"url": e.target.href});
}
```

Communication entre content et background scripts

■ Exemple -suite- :

```
// background-script.js
```

```
chrome.runtime.onMessage.addListener(notify);
```

```
function notify(message) {  
  chrome.notifications.create({  
    "type": "basic",  
    "iconUrl": chrome.extension.getURL("icons/link-48.png"),  
    "title": "You clicked a link!",  
    "message": message.url  
  });  
}
```

■ Extrait du manifest :

```
"permissions": ["notifications"],  
"background": {"scripts": ["background-script.js"]},  
"content_scripts": [  
  { "matches": ["<all_urls>"] , "js": ["content-script.js"]} ]
```

Connexion durable entre scripts

- Un content et un background script peuvent communiquer via des objets runtime.Port
- Pour obtenir cet objet, un content script peut invoquer la méthode `runtime.connect` (le background script fait `tabs.connect`)
- En utilisant la méthode `postMessage` de l'objet Port, les deux scripts peuvent s'envoyer des messages
- Exemple : // content-script.js

```
var myPort = chrome.runtime.connect({name:"port-from-cs"});  
myPort.postMessage({greeting: "Hello from content script"});
```

Connexion durable entre scripts -suite-

- Pour accepter des connexions, il faut définir un écouteur d'événements `runtime.onConnect`, qui reçoit en paramètre le Port
Exemple : // background-script.js

```
chrome.runtime.onConnect.addListener(function connected(p) {  
  p.postMessage({greeting: "hi there content script!"});  
});
```
- Pour recevoir des messages, il faudra définir un écouteur d'événements `onMessage` sur l'objet Port (le message est dans le paramètre de la callback : `m`)
Exemple : // suite de background-script.js

```
p.onMessage.addListener(function(m) {  
  console.log("In background script, received message from content script");  
  console.log(m.greeting);  
});
```

Connexion durable entre scripts -suite-

- Le content-script écoute les messages envoyés par l'autre script :

```
// content-script.js
myPort.onMessage.addListener(function(m) {
  console.log("In content script, received message from
              background script:");
  console.log(m.greeting);
});
```

- Possibilité de communiquer avec les scripts embarqués dans les pages Web visitées par l'utilisateur avec `window.postMessage` et `window.addEventListener("message", ...)`

Permissions requises par les plugins

- Les plugins peuvent demander des permissions au navigateur pour avoir accès à :
 - certains **hôtes** : pour leur envoyer des requêtes XHR par ex Définis par des motifs (patterns)
 - certaines **API** : pour accéder à des fonctionnalités précises (voir deuxième partie du cours)
 - « **activeTab** » : pour donner des privilèges à l'extension sur l'onglet actif, comme exécuter des scripts JS ou charger des feuilles de style CSS
- Le navigateur peut demander des autorisations à l'utilisateur lors de l'installation du plugin (pour valider les permissions)

Permissions requises par les plugins -suite-

- Dans le manifest, ajouter une clé « permissions » qui a comme valeur un tableau de chaînes de caractères
- Exemples :
"permissions": ["webRequest","webRequestBlocking"]

"permissions": ["*://developer.mozilla.org/*","webRequest"]

Plan du cours

- Mécanisme des WebExtensions
- Quelques APIs fournies par les navigateurs

Interception des requêtes et réponses HTTP

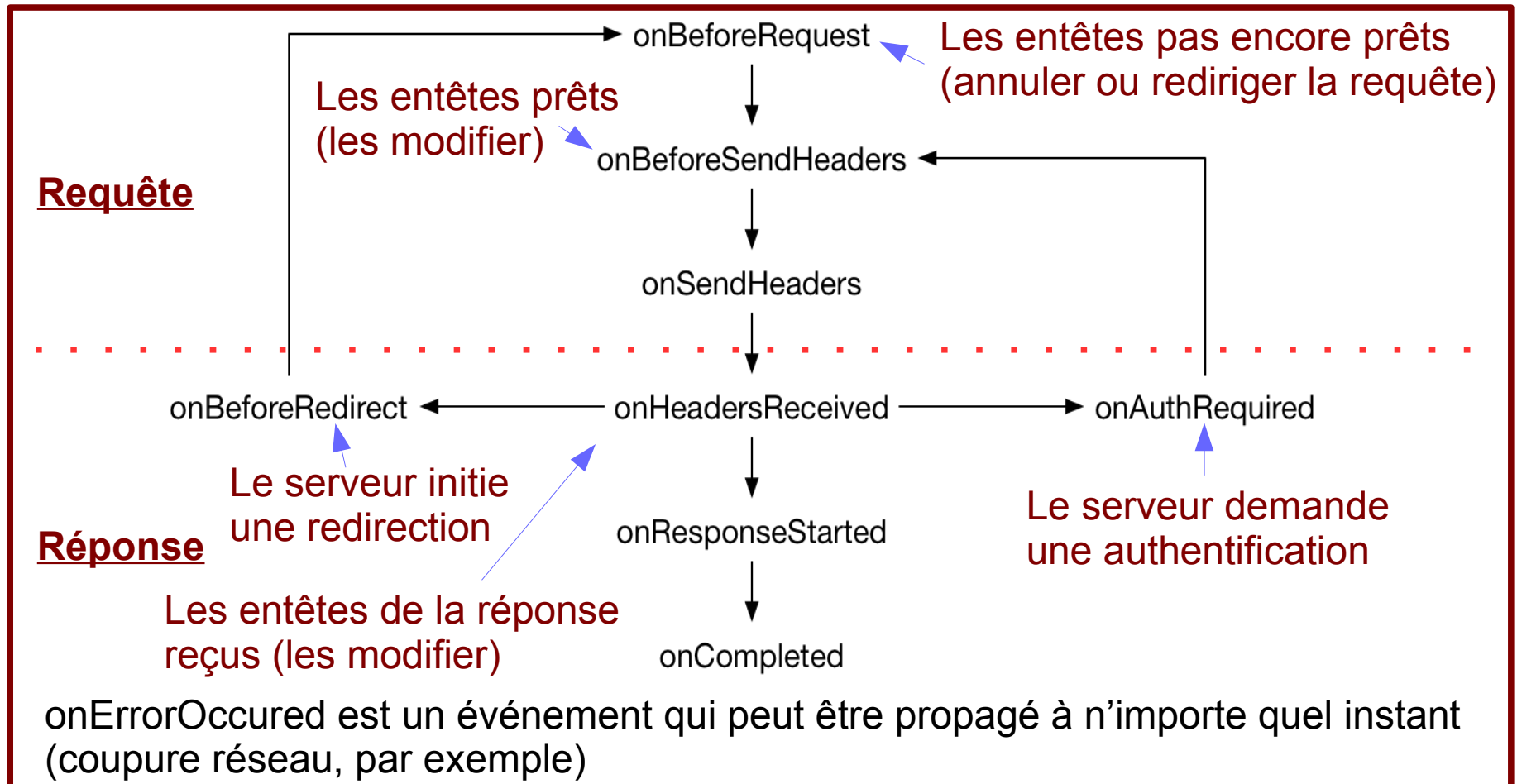
- Nous devons demander une permission API au navigateur :
"permissions": ["webRequest"],
- Ceci permet de définir des écouteurs d'événements à différentes étapes du cycle de vie des requêtes et réponses HTTP (côté client)
- Dans les écouteurs, nous pouvons :
 - Accéder aux entêtes et corps des requêtes et des réponses
 - Annuler et rediriger des requêtes
 - Modifier les entêtes des requêtes et des réponses

Exemple d'interception de requêtes HTTP

■ Exemple : journaliser les requêtes HTTP

```
// manifest.json
{
  "description": "Demonstrating webRequests",
  "manifest_version": 2, "name": "webRequest-demo", "version": "1.0",
  "permissions": ["webRequest", "<all_urls>"],
  "background": {"scripts": ["background.js"]}
}
// background.js
function logURL(requestDetails) {
  console.log("Loading: " + requestDetails.url);
}
chrome.webRequest.onBeforeRequest.addListener(
  logURL, {urls: ["<all_urls>"]}
);
```

Événements « interceptables » par un plugin



Écouteurs d'événements liés à l'interception

- Les méthodes d'ajout d'écouteurs d'événements peuvent généralement prendre trois arguments :
 - La fonction callback (l'écouteur)
 - Un objet filtre pour choisir les URL ou les ressources pour lesquels nous souhaitons faire l'interception
 - Un tableau optionnel pour indiquer des infos supplémentaires (extraInfoSpec)
- Exemple :
`chrome.webRequest.onBeforeRequest.addListener(
 logURL, {urls: ["<all_urls>"] });`
- La fonction callback reçoit en paramètre un objet qui contient tous les détails de la requête ou de la réponse HTTP

Annuler ou rediriger une requête

- D'abord, dans les permissions, ajouter « webRequestBloquing »
- Indiquer le blocage de la requête en passant la chaîne « blocking » dans le tableau, troisième paramètre de addListener
- Ensuite, dans la fonction callback retourner un objet BlockingResponse avec l'une des deux propriétés suivantes :
 - cancel avec comme valeur true : annulation de la requête
 - redirectUrl avec comme valeur l'URL de la page vers laquelle rediriger

Exemple de redirection

- L'ajout d'un écouteur :

```
chrome.webRequest.onBeforeRequest.addListener(  
  redirect,  
  {urls:["http://www.lirmm.fr/*"], types:["image"]},  
  ["blocking"]  
);
```

- La fonction callback redirect :

```
function redirect(requestDetails) {  
  console.log("Redirecting: " + requestDetails.url);  
  return {  
    redirectUrl: "http://image.shutterstock.com/z/stock-vector-vector-skull-danger-sign-120892354.jpg"  
  };  
}
```

- Rappel : (dans le manifest)

```
"permissions": ["webRequest", "webRequestBlocking"]
```

Types de ressources

- Dans l'ajout d'un écouteur, on filtre le type de ressources
`chrome.webRequest.onBeforeRequest.addListener(
 redirect,
 {urls:["http://www.lirmm.fr/*"], types:["image"]},
 ["blocking"]
);`
- Quelques valeurs possibles :
 - "main_frame"
 - "stylesheet"
 - "script"
 - "object"
 - "xmlhttprequest"
 - "font"
 - "other"

Paramètre de la callback

- Le paramètre de la fonction callback lors de la redirection :

```
function redirect(requestDetails) {  
    console.log("Redirecting: " + requestDetails.url);  
    ...  
}
```

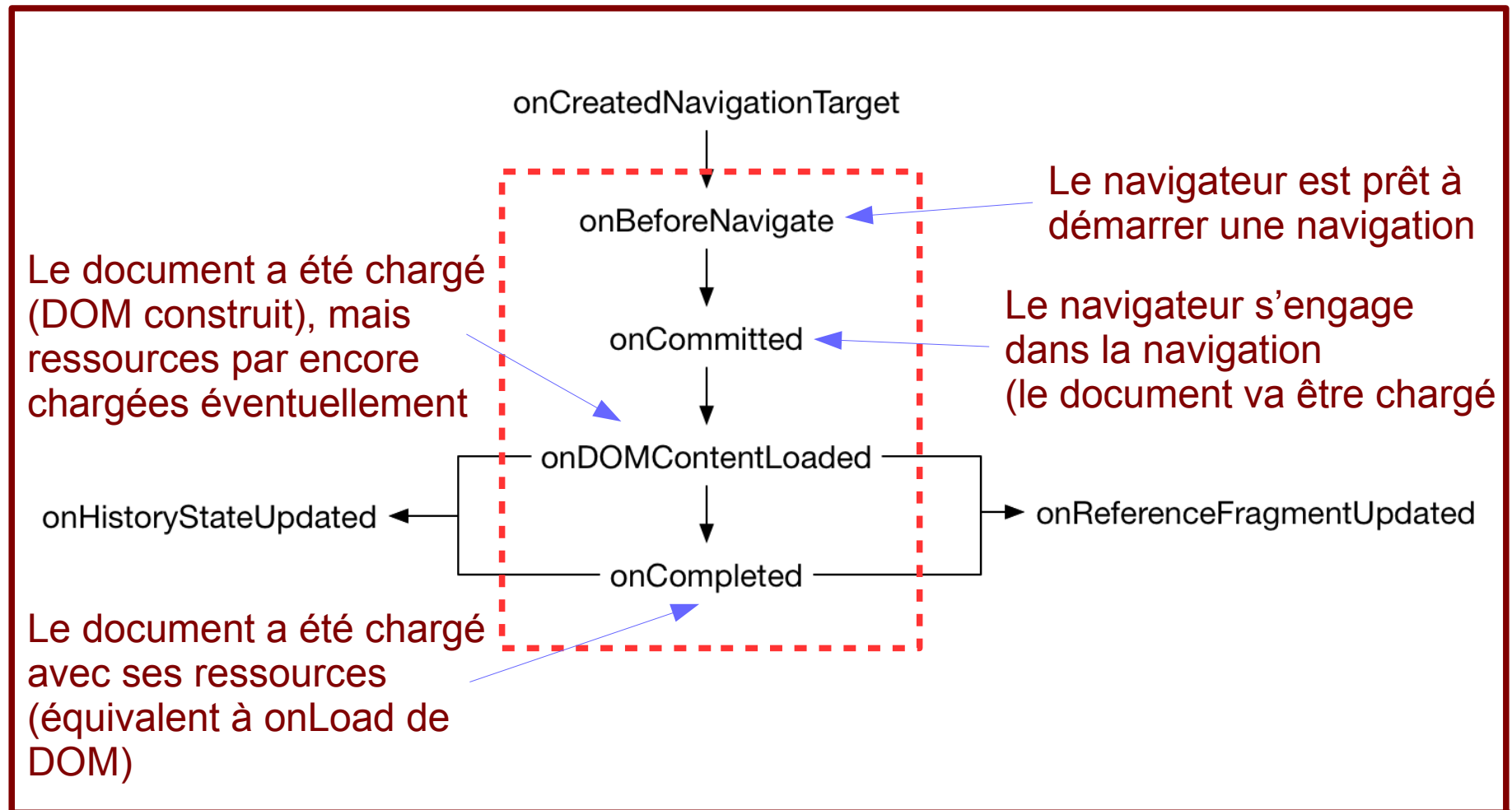
- Le paramètre de la callback pour changer l'entête d'une requête :

```
var targetPage = "http://useragentstring.com/*";  
var ua = "Opera/9.80 (X11; Linux i686; Ubuntu/14.10) Presto/2.12.388 Version/12.16";  
function rewriteUserAgentHeader(e) {  
    for (var header in e.requestHeaders) {  
        if (header.name == "User-Agent") {header.value = ua;}  
    }  
    return {requestHeaders: e.requestHeaders};  
}  
chrome.webRequest.onBeforeSendHeaders.addListener(  
    rewriteUserAgentHeader,  
    {urls: [targetPage],["blocking", "requestHeaders"]});
```


Interception des navigations

- Une navigation est une transition d'un URL à un autre, suite à une action de l'utilisateur (ou pas, parfois) comme le clique sur un lien
- Il faudrait d'abord demander une permission « webNavigation »
- Il est possible de définir ensuite des écouteurs d'événement sur chaque étape du cycle de vie d'une navigation
- API d'un niveau plus élevé comparé à celui de webRequest (traite de l'interface utilisateur du navigateur et non de HTTP)

Événements « interceptables » par un plugin



Navigations et frames

- Chaque navigation est une transition d'URL dans un frame du navigateur : onglet ou iframe
- Chaque frame est identifié par un id de frame (frameId) et un id d'onglet (tabId)
- Les paramètres des fonctions callback des écouteurs d'événement ont des propriétés : frameId, tabId, url, ...
- Il est possible de définir des filtres : des URL vers lesquels nous souhaitons intercepter les navigations
C'est le deuxième paramètre des méthodes `addEventListener`

Types de transitions

- Le paramètre de la fonction callback de l'événement `onCommitted` comporte une propriété qui s'appelle `transitionType`
- Les valeurs possibles de cette propriété :
 - `link` : l'utilisateur a cliqué sur un lien
 - `typed` : l'utilisateur a saisi un URL dans la barre d'adresse
 - `auto_bookmark` : l'utilisateur a cliqué sur un favori/marque-page
 - `reload` : rechargement de la même page par l'utilisateur
 - ...

Quelques références

- **Mozilla Developer Network.**

<https://developer.mozilla.org/en-US/Add-ons/WebExtensions>

- **Documentation Chrome.**

<https://developer.chrome.com/extensions>

- **Documentation Microsoft Edge.**

<https://docs.microsoft.com/en-us/microsoft-edge/extensions>

Questions

