Développement d'Applications Web

Cours 6 : Solution des exercices

Vincent Simonet, 2014-2015

Exercise 1: AJAX

Question 1.1

```
function ajax(url, fn) {
  var xhr = new XMLHttpRequest();
  xhr.open('get', url);

  xhr.onreadystatechange = function() {
    if (xhr.readyState === 4) {
      if (xhr.status === 200) {
        fn(JSON.parse(xhr.responseText));
    }
  }

  xhr.send(null);
}
```

Question 1.2

```
function ajax_2_parallel(url1, url2, fn) {
  var response1 = undefined;
  var response2 = undefined;
  function finalize() {
    if (response1 && response2) {
      fn(response1, response2);
    }
  }
  ajax(url1, function(response) {
    response1 = response;
    finalize();
  });
  ajax(url2, function(response) {
    response2 = response;
    finalize();
  });
}
```

La fonction `finalize` est appelée par chacun des deux *callbacks*. Lors du premier appel, l'une des variables `response1` et `response2` n'a pas encore été définie, la fonction retourne donc immédiatement. Lors du second appel, les deux variables sont définies, et `fn` est appelée.

Question 1.3

```
function ajax_2_series(url1, url2, fn) {
  ajax(url1, function(response1) {
    ajax(url2, function(response2) {
      response2 = response;
      fn(response1, response2);
    });
  });
}
```

Pour une exécution en série, il suffit de faire le deuxième appel asynchrone dans le *callback* du premier appel.

Question 1.4

```
function ajax_n_parallel(urls, fn) {
  var count = urls.length;
  var responses = [];
  function finalize() {
    --count;
    if (count == 0) {
      fn(responses);
    }
  }
  for (var i = 0; i < urls.length; ++i) {
    ajax(urls[i], function(response) {
      responses[i] = response;
      finalize();
    });
  }
}
```

Cette fonction généralise le principe de la fonction `ajax_2_parallel`. Les réponses sont stockées dans un tableau. La variable `count` permet de compter le nombre de rèponses reçues et de déterminer lorsque toutes les réponses ont été reçues. À ce moment là, la fonction `fn` peut être appelée.

Question 1.5

```
function ajax_n_parallel(urls, fn) {
  var responses = [];
  var iter = function(i) {
    if (i == urls.length) {
      fn(responses);
    } else {
      ajax(urls[i], function(response_i) {
        responses.push(response_i);
        iter(i + 1);
    }
  }
  iter(0);
```

```
}
```

Pour "chaîner" les appels, ont définit une fonction récursive. L'appel récursif est effectué dans le callback de l'appel asynchrone.

Exercise 2: JSONP

Question 2.1

```
var global_callback = null;

function jsonp(url, fn) {
  global_callback = fn;
  var script = document.createElement('script');
  script.setAttribute('src', url + '&callback=global_callback');
  document.body.appendChild(script);
}
```

La variable `global_callback` est définie en dehors du périmètre de la fonction `jsonp`, puisque le script reçu en réponse à l'appel JSONP sera interprété en dehors de ce périmètre.

Question 2.2

La fonction définie à la question 2.1 ne gère pas correctement les appels parallèles, puisqu'elle utilise une variable globale partagée (`global_callback`). Pour contourner ce problème, nous allons utiliser un compteur (`counter`) qui sera incrémenté à chaque appel à la fonction `jsonp`, de manière à utiliser un nom de callback différent. L'objet `window` est l'objet global en JavaScript dans un navigateur. L'instruction `window['x'] = ...` permet d'affecter la variable globale `x` (avec une indirection, puisque le nom `x` est donné dans une chaîne de caractères).

```
var counter = 0;

function jsonp(url, fn) {
  var cbName = 'jsonp_callback_' + counter;
  ++counter;
  window[cbName] = function(response) {
    window[cbName] = null;
    fn(response);
  };
  var script = document.createElement('script');
  script.setAttribute('src', url + '&callback=' + cbName);
  document.body.appendChild(script);
}
```

Question 2.3

La fonction `jsonp` abstrait un RPC utilisant le protocole JSONP avec le même prototype que la fonction `ajax` qui utlisait le protocole AJAX. On peut donc reprendre les fonctions écrites aux questions 1.2 à 1.5 en remplaçant les appels à la fonction `ajax` par un appel à la fonction `jsonp`.