PROMESSES ET ANGULARJS

LES PROMESSES

LES PROMESSES

QU'EST-CE QU'UNE PROMESSE (PROMISE) ?

+ « *Une interface permettant de manipuler un objet représentant le résultat différé d'une opération asynchrone, qui peut ou non se terminer à n'importe quel moment* »

+ Résolue de façon asynchrone, via des fonctions de callback, pour gérer la valeur de retour ou un cas d'erreur, via une méthode **then(successCallback, errorCallback, notifyCallback)**

+ Plusieurs propositions de standardisation :

+ Cas d'utilisations : travailler avec du code dont on ne maitrise pas ou qu'on ne souhaite pas maitriser la durée d'exécution : appels au backend / calcul métier coté frontend trop long pour attendre...

LES PROMESSES

ENCHAINEMENT DE PROMESSES

- L'enchainement de callback dans les opérations asynchrones peut rapidement impliquer un grand nombre d'imbrications, avec une gestion des erreurs à chaque niveau
- La fonction **then()** d'une promesse renvoie à son tour une promesse
  - permet de chaîner les promesses
  - les fonctions de callback sont optionnelles : permet de gérer les cas d'erreur à la fin d'un enchainement de promesses

```
promise.then(step1)
       .then(step2)
       .then(step3)
       .then(step4)
       .then(null, function (error) {
           // Handle any error from step1 through step4
       });
```

LES PROMESSES

SERVICE $Q

- Implémentation des promesses au sein d'Angular
- Inspiré par la librairie Q de Kris Kowal (https://github.com/kriskowal/q)
- Création d'un **promise manager** via la fonction **$q.defer()**
  - Propriété **promise** : la promesse, qui dispose de la fonction *then()*
  - Méthode **resolve(value)** : résoud la promesse avec la valeur de retour en paramètre
    la valeur de retour peut être une promesse : dans ce cas la promesse associée au promise manager ne sera résolue que lorsque la promesse en paramètre sera à son tour résolue
  - Méthode **reject(reason)** : résoud la promesse en erreur, avec la raison en paramètre
  - Méthode **notify(value)** : notifie d'une mise à jour du statut d'exécution de la promesse ; peut-être appelée plusieurs fois avant la résolution de la promesse

LES PROMESSES

SERVICE $Q

```
function asyncGreet(name) {
    var deferred = $q.defer();

     $timeout(function() {
        deferred.notify('About to greet ' + name + '.');

        // ...
        // ------ 2 minutes plus tard --------- //
        if (okToGreet(name)) { deferred.resolve('Hello, ' + name + '!'); }
        else { deferred.reject('Greeting ' + name + ' is not allowed.'); }
    }, 0);

    return deferred.promise;
  }

  var promise = asyncGreet('Robin Hood');
  promise.then(function(greeting) { alert('Success: ' + greeting); },
               function(reason) { alert('Failed: ' + reason); },
               function(update) { alert('Got notification: ' + update); });
```

LES PROMESSES

SERVICE $Q

+ Chainage :

```
promiseB = promiseA.then(function(result) {
    return result + 1;
});
```

+ Une promesse peut en cacher une autre...

```
function promesseBaseeSurAutrePromesse() {
    var deferred = $q.defer();

    if (testKO) {
        deferred.reject({ message: 'promesse rejetée car le test a échoué' });
    } else {
        deferred.resolve(fonctionRetournantUneAutrePromesse());
    }

    return deferred.promise;
}
```

LES PROMESSES

SERVICE $Q : PROPRIÉTÉ PROMISE

+ **promise.then(successCallback, errorCallback, notifyCallback)** : retourne une promesse

+ **promise.catch(errorCallback)** : alias de **promise.then(null, errorCallback)**, pour la gestion d'erreur en fin de chaîne

+ **promise.finally(callback)** : permet de réaliser un traitement après résolution de la promesse, même si elle est rejetée

```
promise.then(function (result) { … })
       .then(function (result) { … })
       .catch(function (error) { … })
       .finally(function() { … });
```

LES PROMESSES

SERVICE $Q

+ **$q.when(value)** : encapsule la valeur donnée en paramètre dans une promesse

  Utile quand cette valeur peut ou non être une promesse

+ **$q.all([promise1, promise2, ...])** : créé une promesse qui sera résolue lorsque toutes les promesses passées en paramètre seront résolues

LES PROMESSES

UTILISATION AU SEIN D'ANGULAR

+ Les services **$timeout**, **$interval**, **$http** s'appuient sur le service **$q** et renvoient des promesses

```
$http({method: 'GET', url: '/someUrl'})
    .then(function (response) { … },
          function (error) { … });
```

+ Le service **$resource**, qui s'appuie sur **$http**, peut accéder à la promesse sous-jacente via la propriété **$promise**

```
var User = $resource('/user/:userId', {userId:'@id'});
User.get({userId:123})
    .$promise.then(function(user) {
        $scope.user = user;
     });
```

SQLI GROUP

EXERCICES PRATIQUES

TP16 - PROMESSES

DANS LE DASHBOARD, N'AFFICHER À L'ÉCRAN LES DASHBOARDS QU'À LA FIN D'EXÉCUTION DE 2 PROMESSES (2 SECONDES ET 5 S)

- ✓ 2 promise manager : $q.defer
- ✓ Simuler un long traitement avec $timeout
- ✓ $q.all puis mettre les données dans le scope

FIN