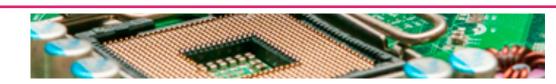


Servicios y Aplicaciones Distribuidas

2014

Seminario 7: Promesas







- En algunas ocasiones el uso de "callbacks" sobre operaciones asincrónicas provocará problemas:
 - A veces, se anidan sucesivos "callbacks".
 - Estos pueden generar excepciones.
 - Cuando una excepción no es tratada, se propaga a la operación desde la que se invocó.
 - Si el tratamiento no es uniforme en todas las operaciones, algunas excepciones podrían "perderse" o llegar a ser manejadas en operaciones en las que no se esperaban.
 - o Resulta difícil seguir el código.
 - El orden de ejecución no siempre es intuitivo.







- La programación con promesas mantiene las signaturas de las operaciones con un formato "tradicional":
 - Una operación recibe una serie de parámetros y devuelve un resultado.
 - Ninguno de los parámetros es un "callback".
 - El resultado es un "objeto" especial: la promesa ("promise" en inglés).
 - Esa "promesa":
 - Se encuentra en alguno de los estados siguientes:
 - "Pendiente": La operación todavía no ha concluido y su resultado se desconoce.
 - "Resuelta": La operación ha terminado. Ya es posible preguntar por el resultado. Dos subestados posibles:
 - » "Rechazada": La operación ha terminado con error.
 - » "Satisfecha": La operación ha terminado con éxito.
 - Es otra forma de modelar la ejecución asincrónica.
 - Su sintaxis es más intuitiva que la basada en "callbacks".
 - Las promesas se ejecutarán cuando el hilo "principal" termine o quede bloqueado por algún motivo.
 - » Quedan pendientes como eventos futuros.







- En JavaScript...
 - No hay ninguna especificación oficial para las promesas.
 - El estándar ECMAscript no las incluye.
 - Propuestas (Según http://wiki.commonjs.org/wiki/Promises):
 - Promises/A.
 - Promises/B.
 - Promises/KISS.
 - Promises/D.
 - Promises/A+.
 - o El módulo Q será el utilizado en esta asignatura.
 - Implanta parcialmente las propuestas "Promises/A" y "Promises/A+".
 - Fácilmente utilizable desde Node.js.
 - Disponible en:
 - » https://github.com/kriskowal/q (repositorio y tutorial breve).
 - » https://github.com/kriskowal/q/wiki/API-Reference (referencia).
 - Se puede instalar con la orden: npm install q







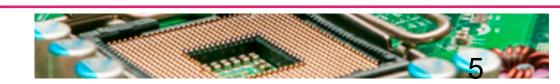
- Ilustraremos cómo funcionan las promesas con un ejemplo...
 - Imaginemos una función que realiza alguna operación matemática sobre sus dos argumentos enteros, retornando el resultado mediante un "callback" asincrónico:

```
// Declaramos la función.
function operaciones(a,b,c) {...};
// Para después invocarla con el "callback".
operaciones(45,67, function(x) {
      console.log("El resultado es: " + x);
});
```

Empleando promesas tendría esta forma:

```
// Declaramos la función.
function operaciones(a,b) {...};
// Para después obtener la "promesa".
var promesa = operaciones(45,67);
promesa.then( function(x) {
    console.log("El resultado es: " + x);
});
```







- Para operar con promesas se utiliza:
 - promise.then(onFulfilled [, onRejected [, onProgress]])
 - Permite establecer funciones manejadoras para los posibles estados de una promesa:
 - Satisfecha: Se ejecutará la función "onFulfilled".
 - Rechazada: Se ejecutará la función "onRejected".
 - Pendiente: Se ejecutará la función "onProgress".
 - La invocación de estas funciones es asincrónica.
 - Por ejemplo, la función "onFulfilled" se ejecutará cuando la promesa esté satisfecha.
 - No hay forma de asegurar cuánto tardará en empezar una vez ya esté satisfecha.
 - » Depende de la gestión de eventos realizada por el intérprete.
 - No depende de cuándo se invocó a then().







- El segundo manejador ("onRejected") facilita la gestión de errores.
 - Resultará especialmente útil cuando se aniden promesas.
 - Es una mejora respecto a los "callbacks".
 - Estos no siempre tienen un parámetro para notificar la ocurrencia de errores y permitir su gestión.
 - Cuando lo hay, complica bastante la lectura del código resultante.

```
// Without error management.
getUser("Someone", function (user) {
   getBestFriend(user, function (friend) {
     ui.showBestFriend(friend);
   });
   });
   **Someone", function (err, user) {
     if (err) { ui.error(err); }
        else {
        getBestFriend(user, function(err, friend) {
        if (err) { ui.error(err); }
        else { ui.showBestFriend(friend); }
     })
     }; });
```







- Podemos convertir cualquier función "normal" en una que devuelva promesas empleando fbind().
- Ejemplo:

```
var Q = require("q");
                                                    var p_4 = eventualAdd(3,6);
// fbind() generates a function B that eventually
                                                    // Checks whether p1 is still pending.
// executes the function received as a parameter,
                                                    function message() {
// returning a promise for B.
                                                     console.log( "Is p1 still pending? "
var eventualAdd = Q.fbind(function(a,b) {
                                                      + p1.isPending());
  return a+b;
});
// Prints its argument on screen.
                                                    message();
function print(a) {
                                                     p1.then(print);
 console.log("Result: " + a);
                                                    message();
                                                    p4.then(print);
                                                     p3.then(print);
// p1, p2, p3 and p4 are promises.
var p1 = eventualAdd(1,1);
                                                    p2.then(print);
var p2 = eventualAdd(1,3);
                                                     console.log("Intermediate message.");
var p_3 = eventualAdd(2,5);
                                                    setTimeout( message, 100 );
```





- Si la función utilizada como "onFulfilled" en un then() retorna algún objeto, ese valor retornado será una nueva promesa.
 - Se podrá anidar una nueva llamada a then().
 - Esto permite encadenar una secuencia de funciones con ejecución asincrónica.

```
var Q = require("q");
                                                 function print2(a) {
                                                  vari = a*a;
                                                  console.log("The square of " + a +
var eventualAdd = Q.fbind(function(a,b) {
                                                   " is " + i);
  return a+b;
                                                  return i;
});
// Prints its argument on screen.
// Returns also the printed argument.
                                                 // p1 and p2 are promises.
function print(a) {
                                                 var p1 = eventualAdd(2,1);
 console.log("Result: " + a);
                                                 var p2 = eventualAdd(1,6);
 return a;
                                                 p1.then(print).then(print2);
                                                 p2.then(print2).then(print);
// Prints the square of its argument.
// Returns also that result.
                                                 console.log("A message. When is it printed??");
```





- Conviene utilizar el parámetro "onRejected" de "then()" cuando las funciones puedan generar excepciones o algún tipo de error.
 - En caso de anidamiento, la gestión puede retrasarse al último "then()".

```
var Q = require("q");
                                                     // Prints the square of its argument.
                                                     // Returns also that result.
var evDivide = Q.fbind(function(a,b) {
 var i=a/b;
                                                     function print2(a) {
  if (b==0) throw new Error("Divisor is zero!");
                                                      vari = a*a;
  return i;
                                                      console.log("The square of " + a +
                                                       " is " + i);
});
// Prints its argument on screen.
                                                      return i;
// Returns also the printed argument.
function print(a) {
                                                     // p1 and p2 are promises.
 console.log("Result: " + a);
                                                     var p1 = evDivide(2,1);
                                                     var p2 = evDivide(6,0);
 return a;
// Rejection handler.
                                                     p1.then(print).then(print2);
function on Error(a) {
                                                     p2.then(print2).then(print,onError);
 console.log("Exception raised: " + a);
                                                     console.log("Message. When is it printed??");
```





Trabajo en seminario

Realizar las 2 actividades disponibles en PoliformaT

Actividad 1: Entender el funcionamiento de los callbacks y las promesas

Actividad 2: Trabajar con promesas en un caso práctico.

