Q module Glossary

Core Promise Methods

promise.**then**(onFulf, onRej, onProg)	permite asignar funciones manejadoras al cumplimiento, fallo o progreso de una promesa
promise.**fail**(onRejected)	captura las excepciones producidas en promise o en sus funciones manejadoras
promise.**progress**(onProgress)	captura los *notify* que indican la evolución de una promesa
promise.**done**()	cualquier excepción se lanza en un evento futuro del bucle de node

.then

```
promiseMeSomething()
.then(function (value) {
}, function (reason) {
});
```

Especifica las funciones handlers de los distintos estados de una promesa (cumplida, rechazada, en progreso). Devuelve una promesa (permite encadenar), si cualquiera de las funciones no se especifica, entonces el valor se pasa a la siguiente promesa.

.fail

```
promiseMeSomething()
.fail(function (value) {
});
```

Permite tratar con el estado *rejected* de una promesa. Si esa promesa se cumple devuelve la misma promesa.

Equivalente a promise.then(undefined, onRejected) o promise.catch(onRejected)

.progress

```
promiseMeSomething()
.progress(function (value) {
});
```

Permite recibir notificaciones (con notify() método disponible en la creación de promesas) **del progreso del cumplimiento de una promesa** (p.ej, el porcentaje de subida de un fichero).

.done

```
promiseMeSomething()
.done(function (value) {
}, function (reason) {
});
```

Similar a .then, pero cualquier excepción que no es tratada, se lanza en un turno futuro del bucle de eventos. Suele utilizarse sin parámetros para **asegurarse de que ninguna excepción se queda sin tratar**.

The Golden Rule of done vs. then usage is: either return your promise to someone else, or if the chain ends with you, call done to terminate it. Terminating with catch is not sufficient because the catch handler may itself throw an error.

Creating Promises Methods

promise.**fapply**(args)	devuelve una promesa para el resultado de llamar a una función pasándole un array con los argumentos
promise.**fcall**(..args)	devuelve una promesa para el resultado de llamar a una función con un número variable de argumentos
promise.**get**(propertyName)	devuelve una promesa por el resultado de acceder a una propiedad de un objeto
promise.**post**(methodName, args)	devuelve una promesa para el resultado de llamar a un método pasándole un array con los argumentos
promise.**invoke**(methodName,..args)	devuelve una promesa para el resultado de llamar a un método con un número variable de argumentos

.**fapply**(args)

Equivalente a:

```
promise.then(function (f) {
    return f.apply(undefined, args);
});
```

Se utiliza para crear una promesa con una función y un vector con sus argumentos.

.**fapply**(args)

Ejemplo:

```
var getUserData = Q.fapply(function (userName) {
    if (!userName) {
        throw new Error("userName must be truthy!");
    }

    if (localCache.has(userName)) {
        return localCache.get(userName);
    }

    return getUserFromCloud(userName);
});
```

.**fcall**(args)

Ejemplo:

```
Q.fcall(function () {
    if (!isConnectedToCloud()) {
        throw new Error("The cloud is down!");
    }
    return syncToCloud();
})
.catch(function (error) {
    console.error("Couldn't sync to the cloud", error);
});
```

Se utiliza para crear una promesa con una función y un número variable de argumentos.

.**get**(propertyName)

Equivalente a:

```
promise.then(function (o) {
    return o[propertyName];
});
```

Se utiliza para crear una promesa para la obtención de una propiedad de un objeto.

Interfacing with Node.js Callbacks

Permiten crear promesas para trabajar con funciones callback (err, result)

Q.**denodeify**(nodeFunc, ...args) aka Q.**nfbind**	devuelve una función que devuelve promesas usando la función estilo callback pasada como parámetro.
Q.**nfapply**(nodeFunc,args)	devuelve una promesa para el resultado de llamar a una función estilo callback con un número variable de argumentos
Q.**nbind**(nodeMethod, thisArg, ...args)	es igual que denodeify pero para métodos de un objeto
Q.**npost**(object, methodName, args) aka Q.**nmapply**	igual que nfapply pero para método de un objeto

.**denodeify**(nodeFunc, ...args)

```
var readFile = Q.denodeify(FS.readFile);

readFile("foo.txt", "utf-8").done(function (text) {

});
```

Se utiliza para crear una función que devuelve promesas usando una función de estilo node-js (con callback(err,data)).

.**nbind**(nodeMethod, thisArg, ...args)

```
var Kitty = mongoose.model("Kitty");
var findKitties = Q.nbind(Kitty.find, Kitty);

findKitties({ cute: true }).done(function (theKitties) {

});
```

Se utiliza para crear una función que devuelve promesas usando un método de estilo node-js (con callback(err,data)).

.**nfapply**(nodeFunc, args)

```
Q.nfapply(FS.readFile, ["foo.txt", "utf-8"]).done(function (text) {
});
```

Devuelve una promesa usando para ello una función de estilo node-js (con callback(err,data)). Los argumentos de esa llamada a la función se pasan en un vector.

.**npost**(object, methodName, args)

```
Q.npost(redisClient, "get", ["user:1:id"]).done(function (user) {
});
```

Devuelve una promesa usando para ello un método de estilo node-js (con callback(err,data)). Los argumentos de esa llamada a la función se pasan en un vector.

Utility functions

promise.**delay**(ms)	devuelve una función que devuelve promesas usando la función estilo callback pasada como parámetro.
promise.**timeout**(ms, msg)	devuelve una promesa para el resultado de llamar a una función estilo callback con un número variable de argumentos
promise.**thenResolve**(value)	es igual que denodeify pero para métodos de un objeto
promise.**thenReject**(reason)	igual que nfapply pero para método de un objeto
promise.**isFulfilled**() promise.**isRejected**(), **isPending**()	comprueban el estado de una promesa y devuelven un booleano

Promise creation

Permiten crear promesas y controlar su estado directamente

Q.**defer()**	Devuelve un objeto que incluye la promesa y métodos para trabajar con la misma
Q **object**	devuelve una promesa a partir de otra promesa, un objeto o un valor.
Q.**reject(reason)**	devuelve una promesa que está en estado rechazado
Q.**Promise(resolver)**	llama de manera síncrónica a resolver y devuelve una promesa cuyo estado es controlado por las funciones que se pasan a resolver