INSTITUTO TECNOLÓGICO DE PABELLÓN DE ARTEAGA

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

SEP
Institutos Tecnológicos

INSTITUTO TECNOLÓGICO®
de Pabellón de Arteaga
TEC

Lectura 11

Desarrollo de Aplicación y Consultorías de Sistemas de Información

PRESENTA: ***Ricardo Montoya Gómez***

DOCENTE:

Eduardo Flores Gallegos

PABELLÓN DE ARTEAGA, AGS., 13 DE DICIEMBRE DE 2018

¿Por qué todo desarrollador de aplicaciones para usuario principiante debe conocer el patrón de publicación-suscripción?

Todo comienza cuando te encuentras en uno de los puntos más emocionantes de tu viaje de desarrollo web: En este mismo momento, empiezas a ver que cuando se trata de JS, es mucho más que unos simples trucos y efectos visuales de jQuery.

Pero no estás desanimado: surgen nuevas ideas constantemente, escribes cada vez más código.

Entonces, empiezas a sentir una pequeña picazón.

Hace una hora tenía 200 líneas de código, ahora supera las 500 líneas. Has leído sobre código limpio y mantenible, y para lograrlo, comienzas a separar tu lógica en archivos, bloques y componentes. Las cosas comienzan a verse hermosas otra vez. Te sientes bien, porque varios archivos se nombran correctamente y se colocan en un directorio determinado. El código se vuelve modular y más mantenible.

De la nada, empiezas a sentir otra picazón, pero la causa aún no está clara.

Las aplicaciones deben responder adecuadamente a los eventos de la red, las interacciones del usuario, los mecanismos de sincronización y varias acciones diferidas.

Usted podría tener su aplicación bellamente dividida en bloques de construcción. Los componentes de navegación y contenido se pueden colocar ordenadamente en directorios apropiados, los archivos de script de ayuda más pequeños podrían contener código repetitivo que realiza tareas mundanas. Todo puede gestionarse mediante el archivo de código app.js de una sola entrada, donde todo comienza. Ordenado.

Al final del día, su cabeza adolorida está llena de varias palabras de moda. Tu mente explota a partir de una gran cantidad de enfoques posibles.

Pero es de esperar que te permita pensar en un código asíncrono un poco más fácil, ya que todo se basa en un principio bastante básico.

Después de todo, son solo máquinas tontas como una calculadora que son capaces de almacenar valores en varios cuadros y cambiar su flujo debido a algunos ifs o llamadas a funciones. Como lenguaje imperativo y ligeramente orientado a objetos, JavaScript no es una especie muy diferente aquí.

Considere una aplicación simple, una muy simple. Por ejemplo, una pequeña aplicación para marcar nuestros lugares favoritos en un mapa. Nada especial allí: solo una vista de mapa a la derecha y una barra lateral simple a la izquierda. Al hacer clic en un mapa se debe guardar un nuevo marcador en un mapa.

Por supuesto ahora queremos que recuerde la lista de lugares que utilizan el almacenamiento local.

Ahora, basándonos en esa descripción, podemos dibujar un diagrama de flujo de acción básico para nuestra aplicación.

Analicemos rápidamente este:

La función init () inicializa el elemento del mapa utilizando la API de Google Maps.

addPlace () maneja el clic del mapa, luego agrega un nuevo lugar a la lista e invoca la representación del marcador.

renderMarkers () recorre los lugares de la matriz y, después de borrar el mapa, coloca los marcadores en él.

En primer lugar, queremos tener una lista de lugares marcados en la barra lateral. En segundo lugar, queremos buscar nombres de ciudades por la API de Google, y aquí es donde se introducen los mecanismos asíncronos.

Hay una cierta característica de obtener el nombre de la ciudad desde la API de Google Debe llamar al servicio adecuado en la biblioteca de JavaScript de Google, y la respuesta tardará un tiempo.

Volvamos a la interfaz de usuario Hay dos áreas de interfaz supuestamente separadas: la barra lateral y el área de contenido principal. La razón es obvia: ¿qué pasaría si en el futuro tengamos cuatro componentes? ¿O seis? ¿O 100? Necesitamos tener nuestro código dividido en partes, de esta manera, tendremos dos archivos JavaScript separados. la pregunta es: ¿cuál debería mantener la matriz con los lugares?

podemos mover el almacenamiento y la lógica a otro archivo de código, que tratará de forma centralizada con los datos. Este archivo, el servicio, será responsable de los problemas y mecanismos, como la sincronización con el almacenamiento local.

El código está perfectamente organizado en cajones adecuados de nuevo.

Después de realizar cualquier acción, la interfaz no reacciona.

¿Por qué? Bueno, no implementamos ningún medio de sincronización. Ni siquiera podemos poner el método getPlaces () en la siguiente línea después de invocar addPlace (), porque la búsqueda de la ciudad es asíncrona y requiere tiempo.

Las cosas suceden en segundo plano, pero nuestra interfaz no es consciente de sus resultados. Una idea bastante simple es encuestar nuestro servicio de vez en cuando.

Ahora, tratemos de imitar "dejar nuestros datos de contacto" en el JavaScript.

JavaScript es un lenguaje muy elegante, una de sus características peculiares es tratar las funciones como cualquier otro valor. Esto significa que cualquier función se puede asignar a una variable o pasar como un parámetro a otra función.

Esta característica es fundamental en escenarios asíncronos.

Podemos definir una función que actualizaría nuestra interfaz de usuario, y luego pasarla a otra parte del código, donde se llamará. Usando este mecanismo, podemos tomar nuestro renderCities y de alguna manera pasarlo al servicio de datos. Allí, se invocará cuando sea necesario

Hay muchas más posibilidades aquí: podemos crear diferentes temas (o canales) para diferentes tipos de acciones. Además, podríamos extraer métodos de publicación y suscripción para separar completamente el archivo de código y usarlo desde allí. Pero por ahora, estamos bien.

Volviendo al título: ¿por qué es tan importante esta cosa? Después de todo, a largo plazo, tiene poco sentido mantener el JavaScript de vainilla y modificar el DOM manualmente, lo mismo ocurre con los mecanismos manuales para pasar y recibir eventos.

Bueno, la verdad es que todos ellos usan alguna variación del patrón de publicación-suscripción.

Como ya dijimos, los oyentes de eventos DOM no son más que suscribirse a acciones de IU de publicación

¿Qué es una Promesa? Desde cierto punto de vista, es solo un mecanismo que nos permite suscribirnos para completar una determinada acción diferida, luego publica algunos datos cuando está listo.

¿Reaccionar y cambiar de estado? Los mecanismos de actualización de los componentes están suscritos a los cambios.

¿Websocket está en ()? Fetch API? Permiten suscribirse a determinadas acciones de la red. Redux? Permite suscribirse a los cambios en la tienda. Y RxJS? Es un patrón de suscripción grande y descarado.
Es el mismo principio. No hay unicornios mágicos bajo el capó. Es como el final del episodio de Scooby-Doo.

No es un gran descubrimiento. Pero es importante saber:

No importa qué método de resolución de problemas asíncronos usará, siempre habrá una variación del mismo principio: algo se suscribe, algo se publica.