Observables compared to other techniques

Contents ›

•••

You can often use observables instead of promises to deliver values asynchronously. Similarly, observables can take the place of event handlers. Finally, because observables deliver multiple values, you can use them where you might otherwise build and operate on arrays.

Observables behave somewhat differently from the alternative techniques in each of these situations, but offer some significant advantages. Here are detailed comparisons of the differences.

Observables compared to promises

Observables are often compared to promises. Here are some key differences:

- Observables are declarative; computation does not start until subscription. Promises execute immediately on creation. This makes observables useful for defining recipes that can be run whenever you need the result.
- Observables provide many values. Promises provide one. This makes observables useful for getting multiple values over time.
- Observables differentiate between chaining and subscription. Promises only have `.then()` clauses. This makes observables useful for creating complex transformation recipes to be used by other part of the system, without causing the work to be executed.
- Observables `subscribe()` is responsible for handling errors. Promises push errors to the child promises. This makes observables useful for centralized and predictable error handling.

Creation and subscription

- Observables are not executed until a consumer subscribes. The `subscribe()` executes the defined behavior once, and it can be called again. Each subscription has its own computation. Resubscription causes recomputation of values.

src/observables.ts (observable)

```
// declare a publishing operation
const observable = new Observable<number>(observer => {
  // Subscriber fn...
});
```

```
// initiate execution
observable.subscribe(() => {
  // observer handles notifications
});
```

- Promises execute immediately, and just once. The computation of the result is initiated when the promise is created. There is no way to restart work. All `then` clauses (subscriptions) share the same computation.

src/promises.ts (promise)

```
// initiate execution
const promise = new Promise<number>((resolve, reject) => {
  // Executer fn...
});

promise.then(value => {
  // handle result here
});
```

Chaining

- Observables differentiate between transformation function such as a map and subscription. Only subscription activates the subscriber function to start computing the values.

src/observables.ts (chain)

```
observable.pipe(map(v => 2 * v));
```

- Promises do not differentiate between the last `.then` clauses (equivalent to subscription) and intermediate `.then` clauses (equivalent to map).

src/promises.ts (chain)

```
promise.then(v => 2 * v);
```

Cancellation

- Observable subscriptions are cancellable. Unsubscribing removes the listener from receiving further values, and notifies the subscriber function to cancel work.

src/observables.ts (unsubcribe)

```
const subscription = observable.subscribe(() => {
  // observer handles notifications
});

subscription.unsubscribe();
```

- Promises are not cancellable.

Error handling

- Observable execution errors are delivered to the subscriber's error handler, and the subscriber automatically unsubscribes from the observable.

src/observables.ts (error)

```
observable.subscribe(() => {
  throw Error('my error');
});
```

- Promises push errors to the child promises.

src/promises.ts (error)

```
promise.then(() => {
  throw Error('my error');
});
```

Cheat sheet

The following code snippets illustrate how the same kind of operation is defined using observables and promises.

OPERATION	OBSERVABLE	PROMISE

Creation	`new Observable((observer) => {`
	`  observer.next(123);`
	`});`
		`new Promise((resolve, reject) => {`
		`  resolve(123);`
		`});`

Transform	`obs.pipe(map((value) => value * 2));`	`promise.then((value) => value * 2);`
Subscribe	`sub = obs.subscribe((value) => {`	`promise.then((value) => {`

	`  console.log(value)`
	`});`
		`  console.log(value);`
		`});`

Unsubscribe	`sub.unsubscribe();`	Implied by promise resolution.

Observables compared to events API

Observables are very similar to event handlers that use the events API. Both techniques define notification handlers, and use them to process multiple values delivered over time. Subscribing to an observable is equivalent to adding an event listener. One significant difference is that you can configure an observable to transform an event before passing the event to the handler.

Using observables to handle events and asynchronous operations can have the advantage of greater consistency in contexts such as HTTP requests.

Here are some code samples that illustrate how the same kind of operation is defined using observables and the events API.

	Observable	**Events API**

Creation & cancellation	`// Setup`
	`const clicks$ = fromEvent(buttonEl, ‘click’);`
	`// Begin listening`
	`const subscription = clicks$`
	`  .subscribe(e => console.log(‘Clicked’, e))`
	`// Stop listening`
	`subscription.unsubscribe();`
		`function handler(e) {`
		`  console.log(‘Clicked’, e);`
		`}`
		`// Setup & begin listening`
		`button.addEventListener(‘click’, handler`
		`// Stop listening`
		`button.removeEventListener(‘click’, hand`

Subscription	`observable.subscribe(() => {`
	`  // notification handlers here`
	`});`
		`element.addEventListener(eventName, (eve`
		`  // notification handler here`
		`});`

Configuration	Listen for keystrokes, but provide a stream representing the value in the input.
	`fromEvent(inputEl, 'keydown').pipe(`
	`  map(e => e.target.value)`
	`);`
		Does not support configuration.
		`element.addEventListener(eventName, (eve`
		`  // Cannot change the passed Event into`
		`  // value before it gets to the handler`
		`});`

Observables compared to arrays

An observable produces values over time. An array is created as a static set of values. In a sense, observables are asynchronous where arrays are synchronous. In the following examples, → implies asynchronous value delivery.

	Observable	**Array**

Given	`obs: →1→2→3→5→7`
	`obsB: →'a'→'b'→'c'`
		`arr: [1, 2, 3, 5, 7]`
		`arrB: ['a', 'b', 'c']`

`concat()`	`concat(obs, obsB)`
	`→1→2→3→5→7→'a'→'b'→'c'`
		`arr.concat(arrB)`
		`[1,2,3,5,7,'a','b','c']`

`filter()`	`obs.pipe(filter((v) => v>3))`
	`→5→7`
		`arr.filter((v) => v>3)`
		`[5, 7]`

`find()`	`obs.pipe(find((v) => v>3))`
	`→5`
		`arr.find((v) => v>3)`
		`5`

`findIndex()`	`obs.pipe(findIndex((v) => v>3))`
	`→3`
		`arr.findIndex((v) => v>3)`
		`3`

`forEach()`	`obs.pipe(tap((v) => {`
	`  console.log(v);`
	`}))`
	`1`
	`2`
	`3`
	`5`
	`7`
		`arr.forEach((v) => {`
		`  console.log(v);`
		`})`
		`1`
		`2`
		`3`
		`5`
		`7`

`map()`	`obs.pipe(map((v) => -v))`
	`→-1→-2→-3→-5→-7`
		`arr.map((v) => -v)`
		`[-1, -2, -3, -5, -7]`

`reduce()`	`obs.pipe(reduce((s,v)=> s+v, 0))`	`arr.reduce((s,v) => s+v, 0)`

→18	18

RESOURCES	HELP	COMMUNITY	LANGUAGES
About	Stack Overflow	Events	简体中文版
Resource Listing	Gitter	Meetups	正體中文版
Press Kit	Report Issues	Twitter	日本語版
Blog	Code of Conduct	GitHub	한국어
Usage Analytics		Contribute	