**Observables vs Promises**

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 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 subcribes. The subscribe() executes the defined behavior once, and it can be called again. Each subscription has its own computation. Resubscription causes recomputation of values.

// declare [a](https://v5.angular.io/api/router/RouterLinkWithHref) publishing operation

new Observable((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.

// initiate execution

new Promise((resolve, reject) => { executer\_fn });

// handle return value

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.

observable.map((v) => 2\*v);

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

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.

const sub = obs.subscribe(...); sub.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.

obs.subscribe(() => {

throw Error('my error');

});

* Promises push errors to the child promises.

promise.then(() => {

throw Error('my error');

});

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.map((value) => value \* 2 ); | promise.then((value) => value \* 2); |
| Subscribe | sub = obs.subscribe((value) => {  console.log(value)  }); | promise.then((value) => {  console.log(value);  }); |
| Unsubscribe | sub.unsubscribe(); | Implied by promise resolution. |