





First We will see syntax

import { Component, OnInit } from '@angular/core';

import { Observable } from 'rxjs';

@Component({

  selector: 'app-new',

  templateUrl: './new.component.html',

  styleUrls: ['./new.component.css']

})

export class NewComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {

     // promise

    const promise = new Promise(resolve=>

      {

        setTimeout(()=>

        {resolve("Promise solved");

        }, 1000)

      })

      promise.then(res=> console.log(res) );

      // Observable

 const observable = new Observable(sub=>

  {

setTimeout(()=>

{

sub.next("Onservable working");

},1000)

  })

   observable.subscribe(res=> {console.log(res) });

  }

}

Differences

Promise are eager

Observable are lazy

import { Component, OnInit } from '@angular/core';

import { Observable } from 'rxjs';

@Component({

  selector: 'app-new',

  templateUrl: './new.component.html',

  styleUrls: ['./new.component.css']

})

export class NewComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {

     // promise

    const promise = new Promise(resolve=>

      {

        console.log("promise called");

        setTimeout(()=>

        {resolve("Promise solved");

        }, 1000)

      })

     // promise.then(res=> console.log(res) );

      // Observable

 const observable = new Observable(sub=>

  { console.log("observable called");

setTimeout(()=>

{

sub.next("Onservable working");

},1000)

  })

  //  observable.subscribe(res=> {console.log(res) });

  }

}

2. Promise can emit single values

Observable can return multiple values

import { Component, OnInit } from '@angular/core';

import { Observable } from 'rxjs';

@Component({

  selector: 'app-new',

  templateUrl: './new.component.html',

  styleUrls: ['./new.component.css']

})

export class NewComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {

     // promise

    const promise = new Promise(resolve=>

      {

        console.log("promise called");

        setTimeout(()=>

        {resolve("Promise solved");

        resolve("Promise solved1");

        resolve("Promise solved2");

        resolve("Promise solved3");

        resolve("Promise solved4");

        }, 1000)

      })

     promise.then(res=> console.log(res) );

      // Observable

 const observable = new Observable(sub=>

  { console.log("observable called");

setTimeout(()=>

{

sub.next("Onservable working");

sub.next("Onservable working1");

sub.next("Onservable working2");

sub.next("Onservable working3");

},1000)

  })

   observable.subscribe(res=> {console.log(res) });

  }

}

3. Observable acts like as an Array so we can use operators on the Observable object to apply

Whenever you want to apply operator, we have to apply it before subscribe

import { Component, OnInit } from '@angular/core';

import { Observable } from 'rxjs';

import {filter} from 'rxjs/operators';

@Component({

  selector: 'app-new',

  templateUrl: './new.component.html',

  styleUrls: ['./new.component.css']

})

export class NewComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {

     // promise

    const promise = new Promise(resolve=>

      {

        console.log("promise called");

        setTimeout(()=>

        {resolve("Promise solved");

        resolve("Promise solved1");

        resolve("Promise solved2");

        resolve("Promise solved3");

        resolve("Promise solved4");

        }, 1000)

      })

     promise.then(res=> console.log(res) );

      // Observable

 const observable = new Observable(sub=>

  { console.log("observable called");

setTimeout(()=>

{

sub.next("Onservable working");

sub.next("Onservable working1");

sub.next("Onservable working2");

sub.next("Onservable working3");

},1000)

  })

  observable.pipe(

    filter(d=> d==='Onservable working1'),

      )

    .subscribe(res=> {console.log(res) });

  }

}

4. Cancelling the subscription

We will see why do we need to unsubscribe

import { Component, OnInit } from '@angular/core';

import { Observable } from 'rxjs';

import {filter} from 'rxjs/operators';

@Component({

  selector: 'app-new',

  templateUrl: './new.component.html',

  styleUrls: ['./new.component.css']

})

export class NewComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {

     // promise

    const promise = new Promise(resolve=>

      {

        console.log("promise called");

        setTimeout(()=>

        {resolve("Promise solved");

        resolve("Promise solved1");

        resolve("Promise solved2");

        resolve("Promise solved3");

        resolve("Promise solved4");

        }, 1000)

      })

    //  promise.then(res=> console.log(res) );

      // Observable

 const observable = new Observable(sub=>

  { console.log("observable called");

  let x= 1;

setInterval(()=>

{ x=x+1;

  sub.next(x);

  },1000);

});

  // observable.pipe(

  //   filter(d=> d==='Onservable working1'),

  //     )

   observable.subscribe(res=> {console.log(res) });

  }

}

import { Component, OnDestroy, OnInit } from '@angular/core';

import { Observable, SubscribableOrPromise, Subscription } from 'rxjs';

import {filter} from 'rxjs/operators';

@Component({

  selector: 'app-new',

  templateUrl: './new.component.html',

  styleUrls: ['./new.component.css']

})

export class NewComponent implements OnInit , OnDestroy{

  constructor() { }

  private mySubscription : Subscription;

  ngOnInit(): void {

     // promise

    const promise = new Promise(resolve=>

      {

        console.log("promise called");

        setTimeout(()=>

        {resolve("Promise solved");

        resolve("Promise solved1");

        resolve("Promise solved2");

        resolve("Promise solved3");

        resolve("Promise solved4");

        }, 1000)

      })

    //  promise.then(res=> console.log(res) );

      // Observable

 const observable = new Observable(sub=>

  { console.log("observable called");

  let x= 1;

setInterval(()=>

{ x=x+1;

  sub.next(x);

  },1000);

});

  // observable.pipe(

  //   filter(d=> d==='Onservable working1'),

  //     )

    this.mySubscription = observable.subscribe(res=> {console.log(res) });

  }

ngOnDestroy()

{

  this.mySubscription.unsubscribe();

}

}

