breaking thought barriers

Angular Training

Session -23

Outlines

Http Client

HTTP Client

http://

Outlines :

- ❑ Features of HTTP
- ❑ Using HttpClient methods
- ❑ Working with HTTP

- Angular provides a simplistic HTTP API for performing HTTP operations in angular applications.

- The front-end applications need to communicate with a server over the HTTP protocol to download data, upload data, and access other back-end services.

Features of HTTP

It offers features like:

- Error handling
- Request and response interception.
- Typed response objects
- Stateless
- Text-Based
- URI (Uniform Resource Identifier)
- Methods
- Headers
- Status Codes

HTTP Methods :

- In Angular, you can use the HttpClient service to make various HTTP requests to interact with remote servers or APIs.

- HttpClient provides methods for different HTTP methods like GET, POST, PUT, DELETE, and more.

1. GET Request:

To make a GET request using HttpClient, you can use the get() method.

```
import { HttpClient } from '@angular/common/http';

constructor(private http: HttpClient) { }

ngOnInit(): void {
  this.http.get('https://api.example.com/data').subscribe((data) => {
    // Handle the response data
    console.log(data);
  });
}
```

2. POST Request:

To make a POST request with data, you can use the post() method.

```
import { HttpClient } from '@angular/common/http';

constructor(private http: HttpClient) { }

submitData(data: any): void {
  this.http.post('https://api.example.com/submit', data).subscribe((response) => {
    // Handle the response
    console.log(response);
  });
}
```

3.PUT Request:

For PUT requests to update data on the server, use the put() method:

```
import { HttpClient } from '@angular/common/http';

constructor(private http: HttpClient) { }

updateData(data: any): void {
  this.http.put('https://api.example.com/update', data).subscribe((response) => {
    // Handle the response
    console.log(response);
  });
}
```

4. **DELETE Request:**

To send a DELETE request to remove data on the server, use the delete() method:

```
import { HttpClient } from '@angular/common/http';

constructor(private http: HttpClient) { }

deleteData(id: number): void {
  this.http.delete(`https://api.example.com/delete/${id}`).subscribe((response) => {
    // Handle the response
    console.log(response);
  });
}
```

5.Setting Headers and Query Parameters:

You can set custom headers and **query parameters** using the HttpHeaders and **HttpParams** classes:

```
import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';
constructor(private http: HttpClient) { }
getDataWithHeadersAndParams(): void {
  const headers = new HttpHeaders({
    'Authorization': 'Bearer your-token',
    'Custom-Header': 'custom-value'
  });
  const params = new HttpParams()
    .set('param1', 'value1')
    .set('param2', 'value2');
  const options = { headers: headers, params: params };
  this.http.get('https://api.example.com/data', options).subscribe((data) => {
    // Handle the response data
    console.log(data);
  });
}
```

Working with HTTP in angular

1. Import HttpClientModule

```
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  imports: [HttpClientModule],
  // ...
})
export class AppModule { }
```

2. Inject HttpClient

```
import { HttpClient } from '@angular/common/http';
constructor(private http: HttpClient) { }
```

3. Making HTTP Requests

```
// Making a GET request
this.http.get('/api/data').subscribe((data) => {
  // Handle the response data
  console.log(data);
});
```

4. **Handling Responses:**

When you make an HTTP request, you typically subscribe to the Observable returned by the HTTP method. You can then handle the response data or errors in the subscription's callback.

```
this.http.get('/api/data').subscribe(
  (data) => {
    // Handle successful response
    console.log(data);
  },
  (error) => {
    // Handle errors
    console.error('An error occurred:', error);
  }
);
```

5. **Sending Data in POST Requests:**

```
const postData = { name: 'John', age: 30 };

this.http.post('/api/postData', postData).subscribe(
  (response) => {
    // Handle successful response
    console.log(response);
  },
  (error) => {
    // Handle errors
    console.error('An error occurred:', error);
  }
);
```

6. Handling Headers and Options:

You can also set HTTP headers and options for your requests. This can be done using the HttpHeaders class and the HttpParams class for query parameters.

```
const headers = new HttpHeaders({
  'Content-Type': 'application/json',
  'Authorization': 'Bearer your-token-here'
});
const options = {
  headers: headers,
  params: new HttpParams().set('param1', 'value1')
};
this.http.get('/api/data', options).subscribe((data) => {
  // Handle the response data
  console.log(data);
});
```

7. **Using Observables:**

Angular's HTTP client returns Observables. You can leverage RxJS operators to manipulate and transform the data received from HTTP requests.

```
import { map } from 'rxjs/operators';

this.http.get('/api/data').pipe(
  map((data) => {
    // Transform data as needed
    return data;
  })
).subscribe((transformedData) => {
  // Handle the transformed data
  console.log(transformedData);
});
```

8. Error Handling and Interceptors:

You can implement error handling strategies and use interceptors to intercept HTTP requests and responses globally. This allows you to apply common error handling or headers to multiple requests.

```
// Error handling example
catchError((error) => {
  console.error('An error occurred:', error);
  return throwError('Something went wrong, please try again later.');
});
```

Performing CRUD Operation Using Angular

https://jsonplaceholder.typicode.com/users

What is Observable?

- Observable help us to manage async data. You can think of Observables as an array of items, which arrive asynchronously over time.
- The observables implement the observer design pattern, where observables maintain a list of dependents. We call these dependents as observers.
- The observable notifies them automatically of any state changes, usually by calling one of their methods.
- Observer subscribes to an Observable. The observer reacts when the value of the Observable changes.
- An Observable can have multiple subscribers and all the subscribers are notified when the state of the Observable changes.

- When an Observer subscribes to an observable, it needs to pass (optional) the three callbacks. next(), error() & complete(). The observable invokes the next() callback, when it receives an value. When the observable completes it invokes the complete() callback. And when the error occurs it invokes the error() callback with details of error and subscriber finishes.

- The Observables are used extensively in Angular. The new HTTPClient Module and Event system are all Observable based.

- The Observables are proposed feature for the next version of Javascript. The Angular uses a Third-party library called Reactive Extensions or RxJs to implement the Observables.

Observables Operators

- Operators are methods that operate on an Observable and return a new observable. Each Operator modifies the value it receives. These operators are applied one after the other in a chain.

- The RxJs provides several Operators, which allows you to filter, select, transform, combine and compose Observables.

- Examples of Operators are map, filter, take, merge, etc

How to use RxJs

- The RxJs is a very large library. Hence Angular exposes a stripped-down version of Observables.

- You can import it using the following import statement

```
import { Observable } from 'rxjs';
```

The above import imports only the necessary features. It does not include any of the Operators.

- To use observables operators, you need to import them. The following code imports the map & catchError operators.

```
import { map, catchError } from 'rxjs/operators';
```

Thank You