Scalars

The GraphQL includes the following default types: `Int`, `Float`, `String`, `Boolean` and `ID`. However, sometimes you may need to support custom atomic data types (e.g. `Date`).

Schema first

In order to define a custom scalar (read more about scalars **here**), we have to create a type definition and a dedicated resolver as well. Here (as in the official documentation), we'll take the `graphql-type-json` package for demonstration purposes. This npm package defines a `JSON` GraphQL scalar type. Firstly, let's install the package:

```
$ npm i --save graphql-type-json
```

Once the package is installed, we have to pass a custom resolver to the `forRoot()` method:

```
import * as GraphQLJSON from 'graphql-type-json';

@Module({
  imports: [
    GraphQLModule.forRoot({
      typePaths: ['./**/*.graphql'],
      resolvers: { JSON: GraphQLJSON },
    }),
  ],
})
export class ApplicationModule {}
```

Now we can use `JSON` scalar in our type definitions:

```
scalar JSON

type Foo {
  field: JSON
}
```

Another form of defining the scalar type is to create a simple class. Let's say that we would like to enhance our schema with

the `Date` type.

```
import { Scalar, CustomScalar } from '@nestjs/graphql';
import { Kind, ValueNode } from 'graphql';

@Scalar('Date')
export class DateScalar implements CustomScalar<number, Date> {
  description = 'Date custom scalar type';

  parseValue(value: number): Date {
    return new Date(value); // value from the client
  }

  serialize(value: Date): number {
    return value.getTime(); // value sent to the client
  }

  parseLiteral(ast: ValueNode): Date {
    if (ast.kind === Kind.INT) {
      return new Date(ast.value);
    }
    return null;
  }
}
```

Afterward, we need to register `DateScalar` as a provider.

```
@Module({
  providers: [DateScalar],
})
export class CommonModule {}
```

And now we are able to use `Date` scalar in our type definitions.

```
scalar Date
```

Code first

In order to create a `Date` scalar, simply create a new class.

```
import { Scalar, CustomScalar } from '@nestjs/graphql';
import { Kind, ValueNode } from 'graphql';

@Scalar('Date', type => Date)
export class DateScalar implements CustomScalar<number, Date> {
  description = 'Date custom scalar type';

  parseValue(value: number): Date {
    return new Date(value); // value from the client
  }

  serialize(value: Date): number {
    return value.getTime(); // value sent to the client
  }

  parseLiteral(ast: ValueNode): Date {
    if (ast.kind === Kind.INT) {
      return new Date(ast.value);
    }
    return null;
  }
}
```

Once it's ready, register `DateScalar` as a provider.

```
@Module({
  providers: [DateScalar],
})
export class CommonModule {}
```

Now you can use `Date` type in your classes.

```
@Field()
creationDate: Date;
```

Support us

Nest is an MIT-licensed open source project. It can grow thanks to the support by these awesome people. If you'd like to

Nest is an MIT-licensed open source project. It can grow thanks to the support by these awesome people. If you'd like to join them, please read more here.

Principal Sponsor

Sponsors / Partners

Designed by Jakub Staroń, hosted by Netlify