Circular dependency

The circular dependency occurs when for example, class A needs class B, and class B needs class A as well. Nest permits creating **circular dependencies** between both providers and modules, but we advise you to avoid whenever it's possible. Sometimes it's really difficult to avoid this kind of the relationships, that's why we have provided some ways to deal with this issue.

Forward reference

The **forward reference** allows Nest referring to references which aren't defined so far. When `CatsService` and `CommonService` depend on each other, both sides of the relationship need to use `@Inject()` and the `forwardRef()` utility, otherwise Nest won't instantiate them because all of the essential metadata won't be available. Let's see the following snippet:

cats.service.ts JS

```
@Injectable()
export class CatsService {
  constructor(
    @Inject(forwardRef(() => CommonService))
    private readonly commonService: CommonService,
  ) {}
}
```

> **HINT**
> The `forwardRef()` function is imported from the `@nestjs/common` package.

Here's the first side of the relationship. Now let's do the same with the `CommonService`:

common.service.ts JS

```
@Injectable()
export class CommonService {
  constructor(
    @Inject(forwardRef(() => CatsService))
    private readonly catsService: CatsService,
  ) {}
}
```

> **WARNING**
> You don't have guarantee which constructor will be called first.

In order to create circular dependencies between modules you have to use the same `forwardRef()` utility on both parts of the modules association:

common.module.ts JS

```
@Module({
  imports: [forwardRef(() => CatsModule)],
})
export class CommonModule {}
```

Module reference

Nest provides the `ModuleRef` class that can be simply injected into any component.

cats.service.ts JS

```
@Injectable()
export class CatsService implements OnModuleInit {
  private service: Service;
  constructor(private readonly moduleRef: ModuleRef) {}

  onModuleInit() {
    this.service = this.moduleRef.get(Service);
  }
}
```

> **HINT**
> The `ModuleRef` class is imported from the `@nestjs/core` package.

The module reference has a `get()` method which allows retrieving a provider available in the current module. Additionally, you can switch to a non-strict mode, which enables you to pick any existing provider among the entire application.

```
this.moduleRef.get(Service, { strict: false });
```

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 join them, please read more here.

Principal Sponsor

Sponsors / Partners