TestingInstanceLoader

August 17, 2026

TestingInstanceLoader

Kind: Class

Source: `packages/testing/testing-instance-loader.ts`

Part of: Testing

`TestingInstanceLoader` creates and initializes providers, controllers, and other dependency instances for Nest testing modules. It extends Nest's core instance-loading flow with testing-specific injector behavior, including support for mocked dependencies during `TestingModule` compilation.

Extends: `InstanceLoader`

Methods

Method	Signature	Returns
`createInstancesOfDependencies`	`createInstancesOfDependencies(modules: Map<string, Module>, mocker: MockFactory)`	`Promise<void>`

Diagram

```
graph LR
  A[TestingModuleBuilder] --> B[TestingInstanceLoader]
  B --> C[TestingInjector]
  B --> D[NestContainer Modules]
  C --> E[Mock Factory / Overrides]
  B --> F[InstanceLoader]
  F --> G[Providers and Controllers Instantiated]
```

Usage

```ts
import { Test } from '@nestjs/testing';
import { UsersService } from './users.service';
import { UsersRepository } from './users.repository';

describe('UsersService', () => {
  it('creates service dependencies with testing overrides', async () => {
    const moduleRef = await Test.createTestingModule({
      providers: [UsersService, UsersRepository],
    })
      .overrideProvider(UsersRepository)
      .useValue({
        findById: jest.fn(),
      })
      .compile();

    // TestingInstanceLoader runs internally during compile().
    const usersService = moduleRef.get(UsersService);

    expect(usersService).toBeDefined();
  });
});
```

AI Coding Instructions

- Preserve the delegation to Nest's core `InstanceLoader`; testing behavior should augment instance creation rather than replace it.
- Ensure the testing injector is associated with the current Nest container before resolving dependencies.
- Keep mocker and provider-override behavior scoped to the testing injector so production dependency resolution is unaffected.
- Prefer using `Test.createTestingModule(...).compile()` instead of constructing `TestingInstanceLoader` directly in application tests.

How it works

`TestingInstanceLoader`

`TestingInstanceLoader` is an exported testing-specific subclass of `InstanceLoader` whose injector type is `TestingInjector`. [packages/testing/testing-instance-loader.ts:6]

Its `createInstancesOfDependencies` method:

- Accepts an optional module map and an optional `MockFactory`; the factory has the type `(token?: InjectionToken) => any`. [packages/testing/testing-instance-loader.ts:7-

10] [packages/testing/interfaces/mock-factory.ts:1-3]

- Assigns the loader’s container to the `TestingInjector` before dependency creation. [packages/testing/testing-instance-loader.ts:11] [packages/testing/testing-injector.ts:31-33]
- Assigns the mocker to that injector only when the `mocker` argument is truthy. [packages/testing/testing-instance-loader.ts:12] [packages/testing/testing-injector.ts:27-29]
- Calls the inherited loader with no arguments, rather than forwarding its `modules` parameter. Consequently, the parent method uses `this.container.getModules()` as its default module map. [packages/testing/testing-instance-loader.ts:8] [packages/testing/testing-instance-loader.ts:13] [packages/core/injector/instance-loader.ts:25-28]

The inherited loading sequence creates provider, injectable, and controller prototypes for every module, then asynchronously loads their instances. [packages/core/injector/instance-loader.ts:28-31] [packages/core/injector/instance-loader.ts:40-45] [packages/core/injector/instance-loader.ts:62-76] [packages/core/injector/instance-loader.ts:80-113] It also inspects modules after successful loading; if instance creation fails, it inspects modules, records the partial graph with the error, and rethrows that error. [packages/core/injector/instance-loader.ts:30-38]

Because this loader uses `TestingInjector`, dependency-resolution errors can be handled by the configured mocker: `TestingInjector` first delegates resolution to `Injector`, then invokes its mock path if that call throws. [packages/testing/testing-injector.ts:35-55] [packages/testing/testing-injector.ts:58-77] If no mocker is set, or if the mocker returns a falsy value, the original resolution error is rethrown. [packages/testing/testing-injector.ts:80-93] When a truthy mock is returned, the injector creates a resolved wrapper for it and registers it as a value provider and export in the container’s internal core module. [packages/testing/testing-injector.ts:94-118] If that internal core module reference is absent, it throws `Error('Expected to have internal core module reference at this point.')`. [packages/testing/testing-injector.ts:103-108]

`TestingModuleBuilder` constructs this loader with its container, a newly created `TestingInjector`, and a graph inspector, then calls this method with the container’s modules and the builder’s optional mocker. [packages/testing/testing-module.builder.ts:176-192] The builder stores a mocker through `useMocker(mocker)`.

[packages/testing/testing-module.builder.ts:47] [packages/testing/testing-module.builder.ts:67-70]

Relationships

- IMPORTS → `InstanceLoader`
- IMPORTS → `Module`