TestingInjector

August 17, 2026

TestingInjector

Kind: Class

Source: `packages/testing/testing-injector.ts`

Part of: Testing

`TestingInjector` extends Nest's dependency injector for testing contexts. It coordinates the test container and mock factory, resolving real providers when available and supplying mocks for unresolved dependencies during testing module compilation.

Extends: `Injector`

Methods

Method	Signature	Returns
`setMocker`	`setMocker(mocker: MockFactory)`	`void`
`setContainer`	`setContainer(container: NestContainer)`	`void`
`resolveComponentWrapper`	`resolveComponentWrapper(moduleRef: Module, name: any, dependencyContext: InjectorDependencyContext, wrapper: InstanceWrapper, resolutionContext: ResolutionContext, keyOrIndex: string`	number)`
`resolveComponentHost`	`resolveComponentHost(moduleRef: Module, instanceWrapper: InstanceWrapper<T>, resolutionContext: ResolutionContext)`	`Promise<InstanceWrapper>`

Properties

Property	Type
`mocker`	`MockFactory`
`container`	`NestContainer`

Where it refuses work

- `TestingInjector` stops the work with `Error` when `!internalCoreModule` — “Expected to have internal core module reference at this point.”.
- `TestingInjector` stops the work with an early return when `!this.mocker`.
- `TestingInjector` stops the work with an early return when `!mockedInstance`.

When something fails

- `TestingInjector` handles failure in 2 places: it turns it into a return value in all 2.

Diagram

```
graph LR
  Builder[TestingModuleBuilder] -->|setMocker()| Injector[TestingInjector]
  Builder -->|setContainer()| Injector
  Injector -->|resolveComponentWrapper()| CoreInjector[Nest Injector]
  CoreInjector -->|Provider found| RealProvider[Resolved provider]
  CoreInjector -->|Provider missing| Injector
  Injector -->|mocker(token)| MockFactory[Mock factory]
  MockFactory --> MockProvider[Mock provider instance]
  MockProvider --> Container[NestContainer]
```

Usage

```
import { Test } from '@nestjs/testing';
import { HttpService } from '@nestjs/axios';
import { UsersService } from './users.service';

const moduleRef = await Test.createTestingModule({
  providers: [UsersService],
})
  // TestingInjector receives and uses this mock factory internally.
  .useMocker((token) => {
    if (token === HttpService) {
      return {
        get: jest.fn().mockResolvedValue({
          data: { id: 'user-1', name: 'Ada' },
```

```
        }),
      };
    }

    // Return undefined for tokens that should not be mocked.
    return undefined;
  })
  .compile();

const usersService = moduleRef.get(UsersService);
```

AI Coding Instructions

- Configure mocks through `TestingModuleBuilder.useMocker()`; `TestingInjector` is an internal integration point used during test module compilation.
- Ensure mock factories return an object or instance for dependencies that cannot be resolved from the testing module.
- Return `undefined` only when a token should be resolved normally or intentionally left unresolved.
- Call `setContainer()` before resolving providers directly so generated mock wrappers can be registered in Nest's internal core module.
- Preserve normal injector behavior by delegating to the base resolution flow before applying mock fallback logic.

Relationships

- IMPORTS → `NestContainer`
- IMPORTS → `STATIC_CONTEXT`
- IMPORTS → `Injector`
- IMPORTS → `InjectorDependencyContext`
- IMPORTS → `ContextId`
- IMPORTS → `InstanceWrapper`
- IMPORTS → `Module`