InstanceWrapper

August 17, 2026

InstanceWrapper

Kind: Class

Source: `packages/core/injector/instance-wrapper.ts`

Part of: Core

`InstanceWrapper` is the internal dependency-injection record for a provider, storing its token, metatype, scope, and resolved instances. It manages provider instances per request context or transient inquirer, while also retaining constructor and property dependency metadata used during resolution.

Methods

Method	Signature	Returns
`getInstanceByContextId`	`getInstanceByContextId(contextId: ContextId, inquirerId: string)`	`InstancePerContext<T>`
`getInstanceByInquirerId`	`getInstanceByInquirerId(contextId: ContextId, inquirerId: string)`	`InstancePerContext<T>`
`setInstanceByContextId`	`setInstanceByContextId(contextId: ContextId, value: InstancePerContext<T>, inquirerId: string)`	`void`
`setInstanceByInquirerId`	`setInstanceByInquirerId(contextId: ContextId, inquirerId: string, value: InstancePerContext<T>)`	`void`
`removeInstanceByContextId`	`removeInstanceByContextId(contextId: ContextId, inquirerId: string)`	`void`
`removeInstanceByInquirerId`	`removeInstanceByInquirerId(contextId: ContextId, inquirerId: string)`	`void`
`addCtorMetadata`	`addCtorMetadata(index: number, wrapper: InstanceWrapper)`	`void`

Method	Signature	Returns
`getCtorMetadata`	`getCtorMetadata()`	`InstanceWrapper[]`
`addPropertiesMetadata`	`addPropertiesMetadata(key: symbol	string, wrapper: InstanceWrapper)`
`getPropertiesMetadata`	`getPropertiesMetadata()`	`PropertyMetadata[]`
`addEnhancerMetadata`	`addEnhancerMetadata(wrapper: InstanceWrapper)`	`void`
`getEnhancersMetadata`	`getEnhancersMetadata()`	`InstanceWrapper[]`
`isDependencyTreeDurable`	`isDependencyTreeDurable(lookupRegistry: string[])`	`boolean`
`introspectDepsAttribute`	`introspectDepsAttribute(callback: ( collection: InstanceWrapper[], lookupRegistry: string[], ) => boolean, lookupRegistry: string[])`	`boolean`
`isDependencyTreeStatic`	`isDependencyTreeStatic(lookupRegistry: string[])`	`boolean`
`cloneStaticInstance`	`cloneStaticInstance(contextId: ContextId)`	`InstancePerContext<T>`
`cloneTransientInstance`	`cloneTransientInstance(contextId: ContextId, inquirerId: string)`	`InstancePerContext<T>`
`createPrototype`	`createPrototype(contextId: ContextId)`	`void`
`isInRequestScope`	`isInRequestScope(contextId: ContextId, inquirer: InstanceWrapper)`	`boolean`
`isLazyTransient`	`isLazyTransient(contextId: ContextId, inquirer: InstanceWrapper	undefined)`
`isExplicitlyRequested`	`isExplicitlyRequested(contextId: ContextId, inquirer: InstanceWrapper)`	`boolean`
`isStatic`	`isStatic(contextId: ContextId, inquirer: InstanceWrapper	undefined)`
`attachRootInquirer`	`attachRootInquirer(inquirer: InstanceWrapper)`	`void`
`getRootInquirer`	`getRootInquirer()`	`InstanceWrapper	undefined`
`getStaticTransientInstances`	`getStaticTransientInstances()`	`void`
`mergeWith`	`mergeWith(provider: Provider)`	`void`

Properties

Property	Type
`name`	`any`
`token`	`InjectionToken`
`async`	`boolean`
`host`	`Module`
`isAlias`	`boolean`
`subtype`	`EnhancerSubtype`
`scope`	`Scope`
`metatype`	`Type
`inject`	`FactoryProvider['inject']
`forwardRef`	`boolean`
`durable`	`boolean`
`initTime`	`number`
`settlementSignal`	`SettlementSignal`

Where it refuses work

- `InstanceWrapper` stops the work with an early return when `this.scope === Scope.TRANSIENT && inquirerId`, in 3 places.
- `InstanceWrapper` stops the work with an early return when `!this.isTransient`, in 2 places.
- `InstanceWrapper` stops the work with an early return when `!collection`.
- `InstanceWrapper` stops the work with an early return when `!isUndefined(this.isTreeDurable)`.
- `InstanceWrapper` stops the work with an early return when `isStatic`.
- `InstanceWrapper` stops the work with an early return when `lookupRegistry.includes(this[INSTANCE_ID_SYMBOL])`.

Diagram

```
graph LR
  Injector[Injector / Resolver] --> Wrapper[InstanceWrapper]
  Wrapper --> Metadata[Constructor & Property Metadata]
```

```
  Wrapper --> ContextStore[Context ID Instance Store]
  Wrapper --> TransientStore[Inquirer ID Instance Store]
  ContextStore --> RequestInstance[InstancePerContext]
  TransientStore --> TransientInstance[InstancePerContext]
```

Usage

```ts
import { InstanceWrapper } from '@nestjs/core/injector/instance-wrapper';
import { ContextIdFactory } from '@nestjs/core/helpers/context-id-factory';

class LoggerService {
  log(message: string) {
    console.log(message);
  }
}

const loggerWrapper = new InstanceWrapper<LoggerService>({
  token: LoggerService,
  name: LoggerService.name,
  metatype: LoggerService,
});

const contextId = ContextIdFactory.create();

loggerWrapper.setInstanceByContextId(contextId, {
  instance: new LoggerService(),
  isResolved: true,
});

const instanceRecord = loggerWrapper.getInstanceByContextId(contextId);

instanceRecord.instance.log('Resolved from the current context');

// Remove request-scoped data when the context is no longer needed.
loggerWrapper.removeInstanceByContextId(contextId);
```

AI Coding Instructions

- Treat `InstanceWrapper` as DI-container infrastructure; application code should normally consume providers through injection instead of creating wrappers directly.
- Use `getInstanceByContextId()` and `setInstanceByContextId()` for request- or context-specific provider state.
- For transient providers, preserve the inquirer ID when reading or storing instances so each consumer receives the correct instance.
- Register constructor dependencies with `addCtorMetadata()` and property dependencies with `addPropertiesMetadata()` using the resolved dependency

wrappers.

- Remove context-bound instances after a request or custom context completes to avoid retaining unnecessary scoped instances.

Used by

5 references from 5 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (5)

- `ExceptionFiltersContext` — `packages/microservices/context/exception-filters-context.ts`:16
- `ListenersController` — `packages/microservices/listeners-controller.ts`:45
- `MicroservicesModule` — `packages/microservices/microservices-module.ts`:23
- `TestingInjector` — `packages/testing/testing-injector.ts`:23
- `SocketModule` — `packages/websockets/socket-module.ts`:27