



CORE

InstanceWrapper

Kind: Class

VERSION
47

UPDATED
7 August 2026

DOCUMENT
reference-subsystem-packages-core/class-instancewrapper

CONTENTS

1	Methods	2	5	Usage	5
2	Properties	4	6	AI Coding Instructions	6
3	Where it refuses work	4	7	Used by	6
4	Diagram	5			

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.

1 Methods

Method	Signature	Returns
<code>getInstanceByContextId</code>	<code>getInstanceByContextId(contextId: ContextId, inquirerId: string)</code>	<code>InstancePerContext<T></code>
<code>getInstanceByInquirerId</code>	<code>getInstanceByInquirerId(contextId: ContextId, inquirerId: string)</code>	<code>InstancePerContext<T></code>
<code>setInstanceByContextId</code>	<code>setInstanceByContextId(contextId: ContextId, value: InstancePerContext<T>, inquirerId: string)</code>	<code>void</code>
<code>setInstanceByInquirerId</code>	<code>setInstanceByInquirerId(contextId: ContextId, inquirerId: string, value: InstancePerContext<T>)</code>	<code>void</code>
<code>removeInstanceByContextId</code>	<code>removeInstanceByContextId(contextId: ContextId, inquirerId: string)</code>	<code>void</code>
<code>removeInstanceByInquirerId</code>	<code>removeInstanceByInquirerId(contextId: ContextId, inquirerId: string)</code>	<code>void</code>
<code>addCtorMetadata</code>	<code>addCtorMetadata(index: number, wrapper: InstanceWrapper)</code>	<code>void</code>
<code>getCtorMetadata</code>	<code>getCtorMetadata()</code>	<code>InstanceWrapper[]</code>
<code>addPropertiesMetadata</code>	<code>`addPropertiesMetadata(key: symbol</code> <code>`</code>	<code>string, wrapper: InstanceWrapper)`</code>
<code>getPropertiesMetadata</code>	<code>getPropertiesMetadata()</code>	<code>PropertyMetadata[]</code>
<code>addEnhancerMetadata</code>	<code>addEnhancerMetadata(wrapper: InstanceWrapper)</code>	<code>void</code>

<code>getEnhancersMetadata</code>	<code>getEnhancersMetadata()</code>	<code>InstanceWrapper[]</code>
<code>isDependencyTreeDurable</code>	<code>isDependencyTreeDurable(lookupRegistry: string[])</code>	<code>boolean</code>
<code>introspectDepsAttribute</code>	<code>introspectDepsAttribute(callback: (collection: InstanceWrapper[], lookupRegistry: string[],) => boolean, lookupRegistry: string[])</code>	<code>boolean</code>
<code>isDependencyTreeStatic</code>	<code>isDependencyTreeStatic(lookupRegistry: string[])</code>	<code>boolean</code>
<code>cloneStaticInstance</code>	<code>cloneStaticInstance(contextId: ContextId)</code>	<code>InstanceProviderContext<T></code>
<code>cloneTransientInstance</code>	<code>cloneTransientInstance(contextId: ContextId, inquirerId: string)</code>	<code>InstanceProviderContext<T></code>
<code>createPrototype</code>	<code>createPrototype(contextId: ContextId)</code>	<code>void</code>
<code>isInRequestScope</code>	<code>isInRequestScope(contextId: ContextId, inquirer: InstanceWrapper)</code>	<code>boolean</code>
<code>isLazyTransient</code>	<code>`isLazyTransient(contextId: ContextId, inquirer: InstanceWrapper</code>	<code>undefined)`</code>
<code>isExplicitlyRequested</code>	<code>isExplicitlyRequested(contextId: ContextId, inquirer: InstanceWrapper)</code>	<code>boolean</code>
<code>isStatic</code>	<code>`isStatic(contextId: ContextId, inquirer: InstanceWrapper</code>	<code>undefined)`</code>
<code>attachRootInquirer</code>	<code>attachRootInquirer(inquirer: InstanceWrapper)</code>	<code>void</code>
<code>getRootInquirer</code>	<code>getRootInquirer()</code>	<code>` InstanceWrapper</code>
<code>getStaticTransientInstances</code>	<code>getStaticTransientInstances()</code>	<code>void</code>
<code>mergeWith</code>	<code>mergeWith(provider: Provider)</code>	<code>void</code>

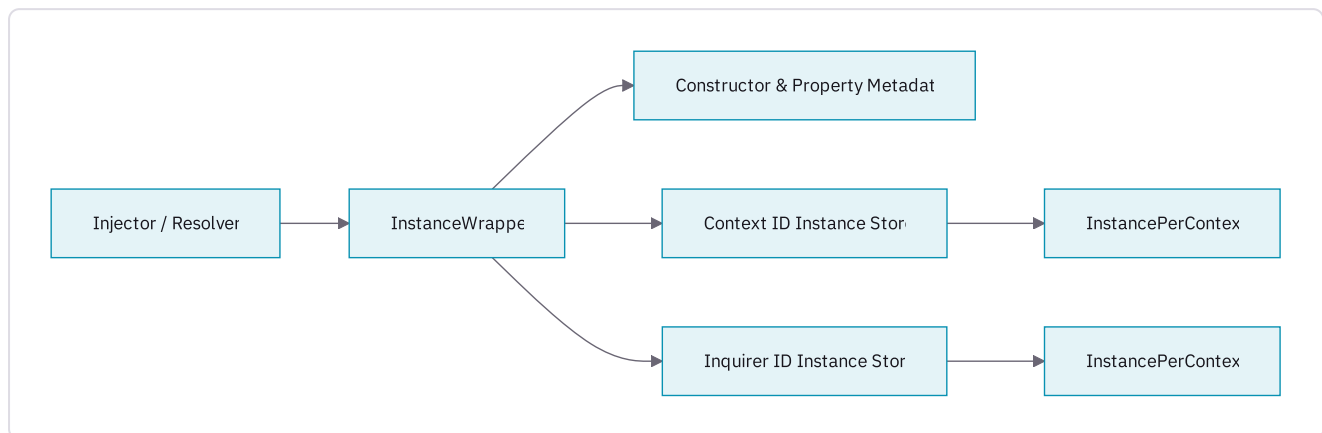
2 Properties

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

3 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])`.

4 Diagram



5 Usage

```

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);
  
```

6 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.

7 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