ContextCreator

August 18, 2026

ContextCreator

Kind: Class

Source: `packages/core/helpers/context-creator.ts`

Part of: Core

`ContextCreator` is a shared base class for building execution-context components from metadata attached to classes and methods. It collects global, class-level, and method-level metadata, then delegates to a concrete implementation to create the final context objects. It also provides consistent request-to-`ContextId` handling for request-scoped dependency resolution.

Methods

Method	Signature	Returns
`createConcreteContext`	`createConcreteContext(metadata: T, contextId: ContextId, inquirerId: string)`	`R`
`getGlobalMetadata`	`getGlobalMetadata(contextId: ContextId, inquirerId: string)`	`T`
`createContext`	`createContext(instance: Controller, callback: (...args: any[]) => void, metadataKey: string, contextId: undefined, inquirerId: string)`	`R`
`reflectClassMetadata`	`reflectClassMetadata(instance: Controller, metadataKey: string)`	`T`
`reflectMethodMetadata`	`reflectMethodMetadata(callback: (...args: unknown[]) => unknown, metadataKey: string)`	`T`
`getContextId`	`getContextId(contextId: ContextId, instanceWrapper: InstanceWrapper)`	`ContextId`

Diagram

```
graph LR
  Request[Incoming request] --> ContextId[getContextId()]
  ContextId --> ContextFactory[ContextIdFactory]

  Controller[Controller / provider instance] --> CreateContext[createContext()]
  Handler[Handler callback] --> CreateContext
  MetadataKey[Metadata key] --> CreateContext

  Global[getGlobalMetadata()] --> CreateContext
  ClassMetadata[reflectClassMetadata()] --> CreateContext
  MethodMetadata[reflectMethodMetadata()] --> CreateContext

  CreateContext --> Concrete[createConcreteContext()]
  Concrete --> Result[Concrete context objects]
```

Usage

```
import { ContextCreator } from '@nestjs/core/helpers/context-creator';
import { ContextIdFactory } from '@nestjs/core/helpers/context-id-factory';

const FEATURE_METADATA = 'feature:contexts';

class ExampleController {
  handleRequest() {
    return 'ok';
  }
}

// Obtain a concrete ContextCreator implementation from the framework or adapter.
declare const contextCreator: ContextCreator;

const controller = new ExampleController();
const request = { headers: { 'x-request-id': 'req-123' } };

// Reuse the same ContextId when resolving request-scoped dependencies.
const contextId = contextCreator.getContextId(request, false);

const contexts = contextCreator.createContext(
  controller,
  controller.handleRequest,
  FEATURE_METADATA,
  contextId,
);

console.log(contexts);
```

AI Coding Instructions

- Treat `ContextCreator` as infrastructure: use a concrete context creator implementation rather than duplicating metadata collection logic.
- Preserve metadata precedence when extending behavior: global metadata is combined with class metadata and then method metadata.
- Pass the same `ContextId` through the execution flow so request-scoped providers resolve consistently.
- Use stable metadata keys for decorators and ensure metadata is applied to the intended class or handler function.
- Keep `createConcreteContext()` focused on converting merged metadata into runtime context objects; avoid performing reflection there.

Relationships

- IMPORTS → `Controller`