GuardsContextCreator

August 18, 2026

GuardsContextCreator

Kind: Class

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

Part of: Core

`GuardsContextCreator` resolves the guards that apply to a controller handler, including method/class metadata and globally registered guards. It converts guard classes or instances into executable `CanActivate` instances while respecting module scope, request context, and dependency injection.

Extends: `ContextCreator`

Methods

Method	Signature	Returns
`create`	`create(instance: Controller, callback: (...args: unknown[]) => unknown, module: string, contextId: undefined, inquirerId: string)`	`CanActivate[]`
`createConcreteContext`	`createConcreteContext(metadata: T, contextId: undefined, inquirerId: string)`	`R`
`getGuardInstance`	`getGuardInstance(metatype: Function	CanActivate, contextId: undefined, inquirerId: string)`
`getInstanceByMetatype`	`getInstanceByMetatype(metatype: Type<unknown>)`	`InstanceWrapper
`getGlobalMetadata`	`getGlobalMetadata(contextId: undefined, inquirerId: string)`	`T`

Where it refuses work

- `GuardsContextCreator` stops the work with an early return when `isEmpty(metadata)`.
- `GuardsContextCreator` stops the work with an early return when `isObject`.
- `GuardsContextCreator` stops the work with an early return when `!instanceWrapper`.
- `GuardsContextCreator` stops the work with an early return when `!this.moduleContext`.
- `GuardsContextCreator` stops the work with an early return when `!moduleRef`.
- `GuardsContextCreator` stops the work with an early return when `!this.config`.

Diagram

```
graph LR
  A[Controller handler] --> B[GuardsContextCreator.create]
  B --> C[Read @UseGuards metadata]
  B --> D[Read global guard metadata]
  C --> E[createConcreteContext]
  D --> E
  E --> F[getGuardInstance]
  F --> G[Resolve injectable from module container]
  G --> H[CanActivate[]]
```

Usage

```
import { ApplicationConfig } from '@nestjs/core/application-config';
import { GuardsContextCreator } from '@nestjs/core/guards/guards-context-creator';
import { NestContainer } from '@nestjs/core/injector/container';

const container = new NestContainer();
const applicationConfig = new ApplicationConfig();

const guardsContextCreator = new GuardsContextCreator(
  container,
  applicationConfig,
);

// `moduleToken` should identify the module that owns the controller.
const guards = guardsContextCreator.create(
  usersController,
  usersController.findAll,
  moduleToken,
);

for (const guard of guards) {
  const allowed = await guard.canActivate(executionContext);

  if (!allowed) {
    throw new Error('Request denied by guard');
```

```
  }
}
```

AI Coding Instructions

- Use `create()` when resolving guards for a controller method; it combines handler, controller, and global guard metadata.
- Ensure the supplied module token matches the module containing guard providers, or class-based guards cannot be resolved from the container.
- Guard metadata may contain either guard instances or injectable guard classes; preserve support for both when extending resolution logic.
- Pass the correct request `contextId` and `inquirerId` for request-scoped guards instead of always relying on the static context.
- Treat this as core framework infrastructure; application code should normally configure guards with `@UseGuards()` or global guard registration rather than instantiate this class directly.

Relationships

- IMPORTS → `CanActivate`
- IMPORTS → `GUARDS_METADATA`
- IMPORTS → `Controller`
- IMPORTS → `Type`
- IMPORTS → `isEmpty`
- IMPORTS → `isFunction`

Used by

4 references from 4 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (4)

- `RpcHandlerMetadata` — `packages/microservices/context/rpc-context-creator.ts`:35
- `MicroservicesModule` — `packages/microservices/microservices-module.ts`:23
- `WsHandlerMetadata` — `packages/websockets/context/ws-context-creator.ts`:34
- `SocketModule` — `packages/websockets/socket-module.ts`:27