InterceptorsConsumer

August 18, 2026

InterceptorsConsumer

Kind: Class

Source: `packages/core/interceptors/interceptors-consumer.ts`

Part of: Core

`InterceptorsConsumer` executes a chain of NestJS interceptors around a route handler or other framework callback. It creates an `ExecutionContextHost`, invokes interceptors in order, and defers execution of the final handler so synchronous values, promises, and observables can be handled consistently.

Methods

Method	Signature	Returns
`intercept`	`intercept(interceptors: NestInterceptor[], args: unknown[], instance: Controller, callback: (...args: unknown[]) => unknown, next: () => Promise<unknown>, type: TContext)`	`Promise<unknown>`
`createContext`	`createContext(args: unknown[], instance: Controller, callback: (...args: unknown[]) => unknown)`	`ExecutionContextHost`
`transformDeferred`	`transformDeferred(next: () => Promise<any>)`	`Observable<any>`

Where it refuses work

- `InterceptorsConsumer` stops the work with an early return when `isEmpty(interceptors)`.
- `InterceptorsConsumer` stops the work with an early return when `i >= interceptors.length`.

Diagram

```
graph LR
  A[InterceptorsConsumer.intercept] --> B[createContext]
  B --> C[ExecutionContextHost]
  A --> D{Interceptors available?}
  D -->|No| E[Invoke next handler]
  D -->|Yes| F[Invoke current interceptor]
  F --> G[CallHandler.handle]
  G --> H[Next interceptor]
  H --> I[transformDeferred]
  I --> J[Route handler / callback result]
  J --> K[Observable, Promise, or value]
```

Usage

```
import { InterceptorsConsumer } from '@nestjs/core/interceptors/interceptors-consumer';
import { CallHandler, ExecutionContext, NestInterceptor } from '@nestjs/common';
import { map } from 'rxjs/operators';

class AddMetadataInterceptor implements NestInterceptor {
  intercept(_context: ExecutionContext, next: CallHandler) {
    return next.handle().pipe(
      map((result) => ({
        data: result,
        processedBy: 'AddMetadataInterceptor',
      })),
    );
  }
}

async function runInterceptors() {
  const consumer = new InterceptorsConsumer();

  const result = await consumer.intercept(
    [new AddMetadataInterceptor()],
    [{ id: '42' }],
    {} as object,
    () => ({ id: '42', name: 'Ada' }),
    async () => ({ id: '42', name: 'Ada' }),
    'http',
  );

  return result;
}
```

AI Coding Instructions

- Preserve interceptor ordering: each interceptor must receive a `CallHandler` that continues to the next interceptor in the chain.
- Use `createContext()` when constructing execution contexts so interceptors receive the expected arguments, instance, callback, and context type.
- Keep handler execution deferred through `transformDeferred()`; do not eagerly invoke `next()` before an interceptor calls `next.handle()`.
- Support route handler results that are plain values, promises, or observables when changing deferred-result handling.
- This class is core framework infrastructure; application code should normally register interceptors through NestJS decorators or providers rather than invoking `InterceptorsConsumer` directly.

Relationships

- IMPORTS → `NestInterceptor`
- IMPORTS → `Type`
- IMPORTS → `CallHandler`
- IMPORTS → `ContextType`
- IMPORTS → `Controller`
- IMPORTS → `isEmpty`

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