ExternalContextCreator

August 17, 2026

ExternalContextCreator

Kind: Class

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

Part of: Core

`ExternalContextCreator` builds executable handlers for framework-managed entry points outside the standard HTTP request pipeline. It resolves parameter metadata, guards, pipes, interceptors, and request-scoped providers, then wraps the target method in an external execution context.

Methods

Method	Signature	Returns
`fromContainer`	`fromContainer(container: NestContainer)`	`ExternalContextCreator`
`create`	`create(instance: Controller, callback: (...args: unknown[]) => unknown, methodName: string, metadataKey: string, paramsFactory: ParamsFactory, contextId: undefined, inquirerId: string, options: ExternalContextOptions, contextType: TContext)`	`void`
`getMetadata`	`getMetadata(instance: Controller, methodName: string, metadataKey: string, paramsFactory: ParamsFactory, contextType: TContext)`	`ExternalHandlerMetadata`
`getContextModuleKey`	`getContextModuleKey(moduleCtor: Function	undefined)`

Method	Signature	Returns
`exchangeKeysForValues`	`exchangeKeysForValues(keys: string[], metadata: TMetadata, moduleContext: string, paramsFactory: ParamsFactory, contextId: undefined, inquirerId: string, contextFactory: undefined)`	`ParamProperties[]`
`createPipesFn`	`createPipesFn(pipes: PipeTransform[], paramsOptions: (ParamProperties & { metatype?: unknown })[])`	`void`
`getParamValue`	`getParamValue(value: T, { metatype, type, data }: { metatype: any; type: any; data: any }, pipes: PipeTransform[])`	`Promise<any>`
`transformToResult`	`transformToResult(resultOrDeferred: any)`	`void`
`createGuardsFn`	`createGuardsFn(guards: any[], instance: Controller, callback: (...args: any[]) => any, contextType: TContext)`	`Function
`registerRequestProvider`	`registerRequestProvider(request: T, contextId: ContextId)`	`void`

Where it refuses work

- `ExternalContextCreator` stops the work with `ForbiddenException` when `!canActivate`.
- `ExternalContextCreator` stops the work with an early return when `cacheMetadata`.
- `ExternalContextCreator` stops the work with an early return when `!moduleCtor`.
- `ExternalContextCreator` stops the work with an early return when `moduleRef.hasProvider(moduleCtor)`.
- `ExternalContextCreator` stops the work with an early return when `isObservable(resultOrDeferred)`.

Diagram

```
graph LR
  A[DI Container] --> B[ExternalContextCreator.fromContainer]
  B --> C[ExternalContextCreator]
```

```
  D[Target Instance and Method] --> E[create]
  C --> E
  E --> F[Read Handler Metadata]
  F --> G[Resolve Parameters]
  G --> H[Run Guards]
  H --> I[Apply Pipes]
  I --> J[Execute Interceptors]
  J --> K[Invoke Target Method]
  K --> L[Transform Result]
```

Usage

```ts
import { ExternalContextCreator } from '@nestjs/core/helpers/external-context-creator';

// `container` is the application's NestContainer instance.
const contextCreator = ExternalContextCreator.fromContainer(container);

class ReportService {
  async rebuild(): Promise<{ status: string }> {
    return { status: 'completed' };
  }
}

const reportService = new ReportService();

const handler = contextCreator.create(
  reportService,
  reportService.rebuild,
  'rebuild',
);

// Executes the method through Nest's external context pipeline.
const result = await handler();

console.log(result.status); // "completed"
```

AI Coding Instructions

- Create instances through `ExternalContextCreator.fromContainer()` so guards, pipes, interceptors, and module references use the application container.
- Pass the target instance, callback, and method name consistently to `create()`; the method name is used to retrieve handler metadata.
- Preserve the execution order: resolve parameters first, then guards, pipes, interceptors, and finally the target callback.

- Use `registerRequestProvider()` when creating request-scoped execution contexts; otherwise scoped dependencies may not resolve correctly.
- Avoid invoking decorated handlers directly when framework behavior is required; use the created proxy to retain metadata-driven processing.