PipesConsumer

August 17, 2026

PipesConsumer

Kind: Class

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

Part of: Core

`PipesConsumer` is NestJS core infrastructure that executes a sequence of `PipeTransform` instances against an argument value. It applies pipes in declaration order, passing each transformed result to the next pipe along with normalized argument metadata.

Methods

Method	Signature	Returns
`apply`	`apply(value: TInput, { metatype, type, data }: ArgumentMetadata, pipes: PipeTransform[])`	`void`
`applyPipes`	`applyPipes(value: TInput, { metatype, type, data }: { metatype: any; type?: any; data?: any }, transforms: PipeTransform[])`	`void`

Diagram

```
graph LR
  A[Route argument value] --> B[PipesConsumer.apply]
  B --> C[Normalize ArgumentMetadata]
  C --> D[applyPipes]
  D --> E[Pipe 1 transform]
  E --> F[Pipe 2 transform]
  F --> G[Additional pipes]
  G --> H[Transformed value]
```

Usage

```
import { PipeTransform, ArgumentMetadata } from '@nestjs/common';
import { PipesConsumer } from '@nestjs/core/pipes/pipes-consumer';

class TrimPipe implements PipeTransform<string, string> {
  transform(value: string, metadata: ArgumentMetadata): string {
    if (metadata.type === 'body' && typeof value === 'string') {
      return value.trim();
    }

    return value;
  }
}

async function transformInput() {
  const pipesConsumer = new PipesConsumer();

  const result = await pipesConsumer.apply(
    '  NestJS  ',
    {
      type: 'body',
      data: undefined,
      metatype: String,
    },
    [new TrimPipe()],
  );

  console.log(result); // "NestJS"
}
```

AI Coding Instructions

- Preserve sequential pipe execution: every pipe must receive the resolved output of the previous pipe.
- Use `apply()` when working with Nest route argument metadata; it normalizes route parameter types before calling `applyPipes()`.
- Ensure custom pipes implement `PipeTransform` and return either a transformed value or a `Promise` resolving to one.
- Do not bypass pipe errors; validation and transformation exceptions should propagate to Nest's exception handling flow.
- Treat `PipesConsumer` as core framework infrastructure; application code should usually configure pipes with decorators or global pipe registration instead of invoking it directly.

How it works

`PipesConsumer`

`PipesConsumer` is a core class that runs an input value through an ordered array of `PipeTransform` instances, passing argument metadata to every transform. Its `apply()` method accepts framework route-parameter metadata, while `applyPipes()` accepts metadata with a direct `type` value. [packages/core/pipes/pipes-consumer.ts:5-28]

Relationships

- IMPORTS → `RouteParamtypes`
- IMPORTS → `ArgumentMetadata`
- IMPORTS → `PipeTransform`

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