WsContextCreator

August 17, 2026

WsContextCreator

Kind: Class

Source: `packages/websockets/context/ws-context-creator.ts`

Part of: Websockets

`WsContextCreator` builds the executable wrapper for WebSocket gateway handlers. It resolves handler metadata, parameter values, guards, pipes, interceptors, and exception filters before delegating execution through the WebSocket proxy layer.

Methods

Method	Signature	Returns
`create`	`create(instance: Controller, callback: (...args: unknown[]) => void, moduleKey: string, methodName: string)`	`(...args: any[]) => Promise<void>`
`reflectCallbackParamtypes`	`reflectCallbackParamtypes(instance: Controller, callback: (...args: any[]) => any)`	`any[]`
`reflectCallbackPattern`	`reflectCallbackPattern(callback: (...args: any[]) => any)`	`string`
`createGuardsFn`	`createGuardsFn(guards: any[], instance: Controller, callback: (...args: unknown[]) => any, contextType: TContext)`	`Function
`getMetadata`	`getMetadata(instance: Controller, methodName: string, contextType: TContext)`	`WsHandlerMetadata`
`exchangeKeysForValues`	`exchangeKeysForValues(keys: string[], metadata: TMetadata, moduleContext: string, paramsFactory:`	`ParamProperties[]`

Method	Signature	Returns
	`WsParamsFactory, contextFactory: (args: unknown[]) => ExecutionContextHost)`	
`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>`

Where it refuses work

- `WsContextCreator` stops the work with `WsException` when `!canActivate`.
- `WsContextCreator` stops the work with an early return when `cacheMetadata`.

Diagram

```
graph LR
  A[Incoming WebSocket event] --> B[WsContextCreator.create]
  B --> C[Reflect handler metadata and pattern]
  C --> D[Create guards]
  C --> E[Create pipes]
  C --> F[Create interceptors]
  C --> G[Create exception filter handler]
  D --> H[Validate execution]
  E --> I[Transform handler parameters]
  F --> J[Invoke gateway callback]
  G --> K[Handle WebSocket exceptions]
  H --> I
  I --> F
  J --> K
```

Usage

```
import { WsContextCreator } from '@nestjs/websockets';

class ChatGateway {
  async handleMessage(client: unknown, payload: { text: string }) {
    console.log(`Received: ${payload.text}`);
  }
}
```

```
// WsContextCreator is typically created and injected by Nest's WebSocket
// runtime rather than instantiated directly.
class GatewayHandlerBinder {
  constructor(private readonly wsContextCreator: WsContextCreator) {}

  bind(gateway: ChatGateway, server: { on(event: string, handler: Function): void }) {
    const handler = this.wsContextCreator.create(
      gateway,
      gateway.handleMessage,
      'ChatModule',
      'handleMessage',
    );

    server.on('message', handler);
  }
}
```

AI Coding Instructions

- Treat `WsContextCreator` as framework infrastructure; prefer gateway decorators and Nest module configuration over manually constructing it.
- Preserve the handler creation flow: metadata reflection, guards, pipes, interceptors, and exception filters must all be included in wrapped handlers.
- Ensure parameter metadata matches the gateway callback signature, since `getParamValue()` and pipe execution depend on reflected parameter configuration.
- Do not invoke gateway callbacks directly when framework behavior is required; use the function returned by `create()` so guards, pipes, and exception handling run.
- Keep WebSocket message patterns stable, as `reflectCallbackPattern()` is used to associate handlers with incoming events.

How it works

`WsContextCreator`

`WsContextCreator` builds the executable wrapper for a WebSocket gateway callback. Its constructor receives the WebSocket proxy plus contexts/consumers for exception filters, pipes, guards, and interceptors; it also creates its own context utility, WebSocket parameter factory, and per-handler metadata cache. [packages/websockets/context/ws-context-creator.ts:40-55]