RpcHandlerMetadata

August 17, 2026

RpcHandlerMetadata

Kind: Interface

Source: `packages/microservices/context/rpc-context-creator.ts`

Part of: Microservices

`RpcHandlerMetadata` describes the runtime metadata required to invoke an RPC handler in the microservices context. It tracks the handler argument count, reflected parameter types, and provides a module-aware resolver for RPC parameter metadata.

Properties

Property	Type
`argsLength`	`number`
`paramtypes`	`any[]`
`getParamsMetadata`	`(moduleKey: string) => RpcParamProperties[]`

Diagram

```
graph LR
  A[RPC Request] --> B[RpcContextCreator]
  B --> C[RpcHandlerMetadata]
  C --> D[argsLength]
  C --> E[paramtypes]
  C --> F[getParamsMetadata(moduleKey)]
  F --> G[RpcParamProperties[]]
  D --> H[Resolved Handler Arguments]
  E --> H
  G --> H
  H --> I[RPC Handler]
```

Usage

```ts
import type { RpcHandlerMetadata } from './rpc-context-creator';
import type { RpcParamProperties } from './rpc-params-factory';

const handlerMetadata: RpcHandlerMetadata = {
  argsLength: 2,
  paramtypes: [String, Object],

  getParamsMetadata(moduleKey: string): RpcParamProperties[] {
    // Resolve metadata registered for the handler within this module.
    return getRegisteredRpcParams(moduleKey);
  },
};

function createHandlerArgs(
  metadata: RpcHandlerMetadata,
  moduleKey: string,
  payload: unknown,
  context: unknown,
) {
  const args = new Array(metadata.argsLength);
  const params = metadata.getParamsMetadata(moduleKey);

  params.forEach(({ index, type }) => {
    args[index] = type === 'payload' ? payload : context;
  });

  return args;
}
```

AI Coding Instructions

- Keep `argsLength` aligned with the target RPC handler's declared parameter count so argument arrays are created at the correct size.
- Preserve `paramtypes` from reflection metadata; consumers may use these types for transformation, validation, or dependency resolution.
- Implement `getParamsMetadata` as module-aware logic, since the same handler metadata may be resolved differently across module contexts.
- Return `RpcParamProperties[]` in parameter-index order or ensure each item includes the correct target index before constructing handler arguments.
- Avoid caching parameter metadata globally without including `moduleKey` in the cache key.

How it works

`RpcHandlerMetadata`

`RpcHandlerMetadata` is an exported TypeScript interface describing cached, per-RPC-handler parameter metadata used by `RpcContextCreator`. Its three fields are `argsLength`, `paramtypes`, and `getParamsMetadata(moduleKey)`. [packages/microservices/context/rpc-context-creator.ts:35-39]

- `argsLength: number` is calculated from the reflected parameter-metadata keys as the highest parameter index plus one, or `0` when there are no keys. [packages/microservices/context/rpc-context-creator.ts:184-185] [packages/core/helpers/context-utils.ts:52-56]
- `paramtypes: any[]` holds the reflected design-time parameter types for the controller method. [packages/microservices/context/rpc-context-creator.ts:186-189] [packages/core/helpers/context-utils.ts:29-34]
- `getParamsMetadata(moduleKey)` is a closure that converts the reflected RPC parameter entries into `RpcParamProperties[]` for a specified module key. [packages/microservices/context/rpc-context-creator.ts:195-202] `RpcParamProperties` extends `ParamProperties` with an optional `metatype`; each resulting parameter record includes its index, type, data, pipe list, and an `extractValue` function. [packages/microservices/context/rpc-context-creator.ts:34-39] [packages/core/helpers/context-utils.ts:15-21]

`RpcContextCreator.getMetadata()` first looks up this object in a `HandlerMetadataStorage<RpcHandlerMetadata>` cache by controller instance and method name. If cached metadata exists, it returns that object; otherwise, it reflects parameter metadata, constructs the interface object, stores it, and returns it. [packages/microservices/context/rpc-context-creator.ts:44-45] [packages/microservices/context/rpc-context-creator.ts:174-210] The storage key combines the controller constructor’s assigned ID or name with the method name. [packages/core/helpers/handler-metadata-storage.ts:50-64]

When creating an RPC callback wrapper, `RpcContextCreator.create()` reads all three fields: it creates an `undefined`-filled argument array with `argsLength`, converts parameter metadata using the supplied module key, and merges each parameter’s reflected type into its metadata when parameter types exist. [packages/microservices/context/rpc-context-creator.ts:68-73] [packages/microservices/context/rpc-context-creator.ts:104-108] [packages/microservices/context/rpc-context-creator.ts:125-136] [packages/core/helpers/context-utils.ts:58-61] [packages/core/helpers/context-utils.ts:64-75]

Calling `getParamsMetadata()` sets the pipes context's module context and creates concrete pipe instances for each parameter entry. For standard RPC parameter types, its extraction function reads payload from argument `0` —optionally selecting a payload property from `data` —context from argument `1`, or the gRPC call from argument `2`; unsupported types return `null`. [packages/microservices/context/rpc-context-creator.ts:213-242] [packages/microservices/factories/rpc-params-factory.ts:4-21] For custom route-argument metadata, it instead creates an extractor that invokes the metadata factory with the parameter data and an RPC-typed `ExecutionContextHost`; a non-function factory yields `null`. [packages/microservices/context/rpc-context-creator.ts:228-235] [packages/core/helpers/context-utils.ts:77-97]

If no reflected RPC parameter metadata is found, `getMetadata()` uses its `defaultCallMetadata` argument. The ordinary default defines parameter `0` as the payload; listener registration switches to a gRPC default that also defines context at index `1` and gRPC call at index `2` when the server is a `ServerGrpc`. [packages/microservices/context/rpc-context-creator.ts:178-184] [packages/microservices/context/rpc-metadata-constants.ts:3-10] [packages/microservices/listeners-controller.ts:71-74]

The interface itself declares no validation, errors, or side effects. In the visible construction path, there is no explicit error handling around metadata reflection or conversion. [packages/microservices/context/rpc-context-creator.ts:168-210]

Relationships

- IMPORTS → `CUSTOM_ROUTE_ARGS_METADATA`
- IMPORTS → `PARAMTYPES_METADATA`
- IMPORTS → `ContextType`
- IMPORTS → `Controller`
- IMPORTS → `PipeTransform`
- IMPORTS → `isEmpty`
- IMPORTS → `FORBIDDEN_MESSAGE`
- IMPORTS → `GuardsConsumer`
- IMPORTS → `GuardsContextCreator`
- IMPORTS → `ContextUtils`
- IMPORTS → `ParamProperties`
- IMPORTS → `ExecutionContextHost`
- IMPORTS → `HandlerMetadataStorage`

- IMPORTS → `ParamsMetadata`
- IMPORTS → `STATIC_CONTEXT`
- IMPORTS → `InterceptorsConsumer`
- IMPORTS → `InterceptorsContextCreator`
- IMPORTS → `PipesConsumer`
- IMPORTS → `PipesContextCreator`