Services in Moleculer framework

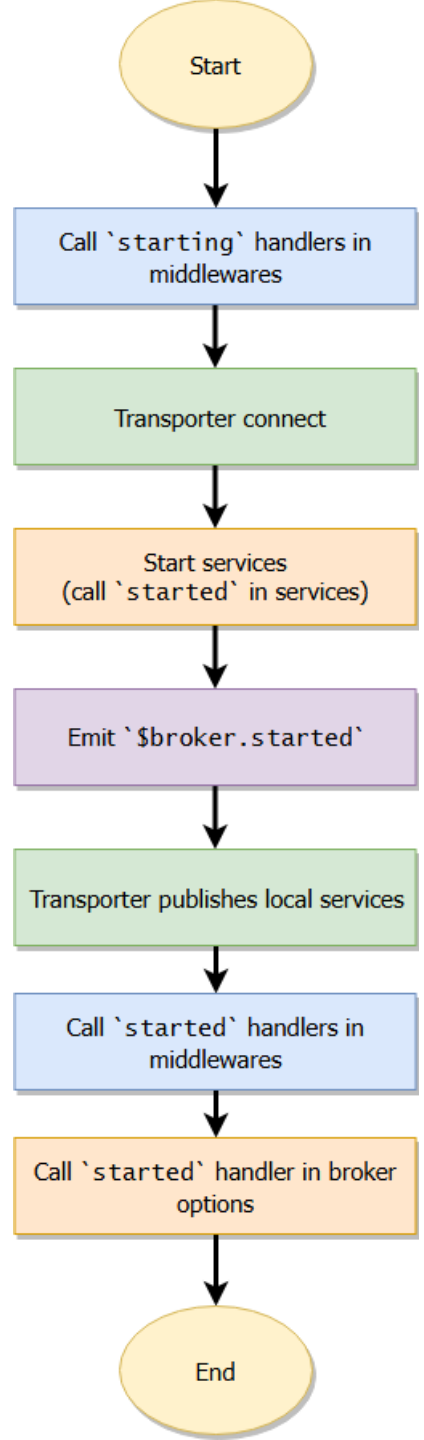
ANJU MUNOTH

Broker lifecycle

- created event handler
- started event handler
- stopped event handler
- merged event handler

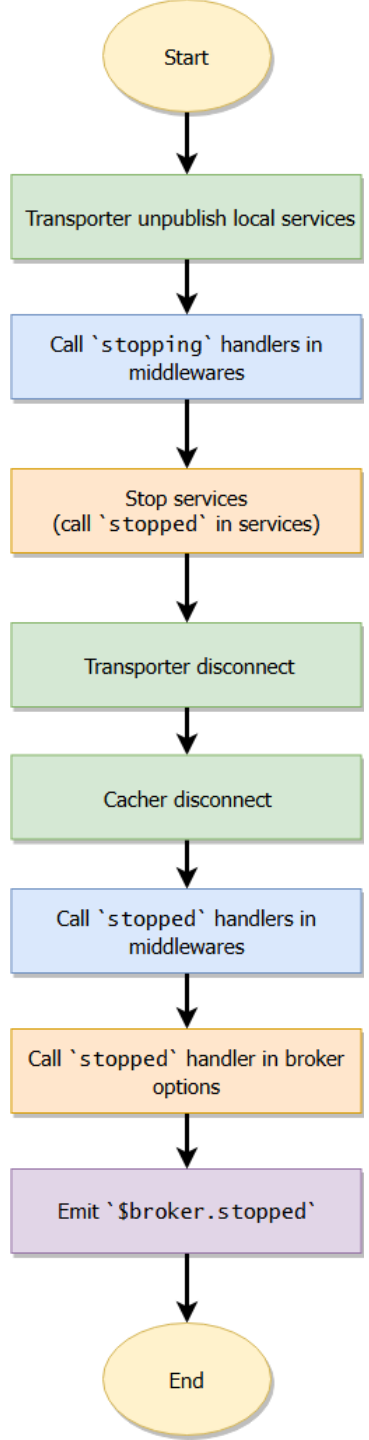
Starting logic

- When starting, the broker tries to establish a connection with the transporter.
- When it's done, it doesn't publish the local service list to remote nodes because it can't accept request yet.
- It starts all services (calls every service started handler).
- Once all services started successfully, broker publishes the local service list to remote nodes.
- Remote nodes only send requests after all local services are properly initialized and started.


Start
Call `starting` handlers in middlewares
Transporter connect
Start services (call `started` in services)
Emit `$broker.started`
Transporter publishes local services
Call `started` handlers in middlewares
Call `started` handler in broker options
End

Stopping logic

- When you call broker.stop or stop the process, at first broker publishes an empty service list to remote nodes, so they will route the requests to other instances instead of services that are stopping.
- Next, the broker starts stopping all local services.
- After that, the transporter disconnects and process exits.


Start
Transporter unpublish local services
Call `stopping` handlers in middlewares
Stop services (call `stopped` in services)
Transporter disconnect
Cacher disconnect
Call `stopped` handlers in middlewares
Call `stopped` handler in broker options
Emit `$broker.stopped`
End

created event handler

- Handler is triggered when the service instance is created (e.g.: at broker.createService or broker.loadService).
- Can use it to create other module instances (e.g. http server, database modules) and store them in this.
- Is a sync event handler.
- Cannot return a Promise and cannot use async/await.

created event handler

```
const http = require("http");

module.exports = {
    name: "www",
    created() {
        // Create HTTP server
        this.server = http.createServer(this.httpHandler);
    }
};
```

started event handler

- Handler is triggered when the broker.start is called and the broker starts all local services.
- Use it to connect to database, listen servers…etc.
- Is an async event handler.
- A Promise can be returned or use async/await.

started event handler

```
module.exports = {
    name: "users",
    async started() {
        try {
            await this.db.connect();
        } catch(e) {
            throw new MoleculerServerError("Unable to connect to database.", e.message);
        }
    }
};
```

stopped event handler

- Handler is triggered when the broker.stop is called and the broker starts stopping all local services.
- Use it to close database connections, close sockets…etc.
- Is an async event handler.
- A Promise can be returned or use async/await.

stopped event handler

```
module.exports = {
    name: "users",
    async stopped() {
        try {
            await this.db.disconnect();
        } catch(e) {
            this.logger.warn("Unable to stop database connection gracefully.", e);
        }
    }
};
```

merged event handler

- Handler is called after the service schemas (including mixins) has been merged but before service is registered.
- Can manipulate the merged service schema before it's processed.

```
// posts.service.js
module.exports = {
    name: "posts",

    settings: {},

    actions: {
        find: {
            params: {
                limit: "number"
            },
            handler(ctx) {
                // ...
            }
        }
    },

    merged(schema) {
        // Modify the service settings
        schema.settings.myProp = "myValue";
        // Modify the param validation schema in an action schema
        schema.actions.find.params.offset = "number";
    }
};
```