MorphCluster Fastify

Данный пакет предоставляет готовые сервисы для обработки HTTP-запросов, организации единой точки входа (API Gateway), реверс-проксирования и раздачи статических файлов. Все компоненты наследуются от `ServiceRequire` и могут быть интегрированы в `ServiceHost` как обычные сервисы MorphCluster.

Fastify

`Fastify` — базовый класс для создания HTTP-сервисов на основе Fastify. Интегрируется с MorphCluster, автоматически регистрирует маршруты, обрабатывает ошибки и поддерживает загрузку файлов, CORS и Swagger-документацию.

Подключение

```
import { ServiceHost } from '@morphcluster/core';
import { Fastify } from '@morphcluster/web';

const host = new ServiceHost('main');
const web = new Fastify(host, {
  host: '0.0.0.0',
  port: 8080,
  title: 'My API',
  prefixUrl: '/api',
  maxFileSize: 2 * 1024 * 1024,
  disableProxy: false
});
host.addService(web);
```

Конфигурация

Параметр	Тип	По умолчанию	Описание
`host`	string	`127.0.0.1`	IP-адрес для прослушивания
`port`	number	`0` (случайный)	Порт (если `0`, назначается ОС)
`title`	string	`Fastify Service`	Заголовок для Swagger
`prefixUrl`	string	`''`	Общий префикс всех маршрутов

Параметр	Тип	По умолчанию	Описание
`maxFileSize`	number	`1000000` (1 МБ)	Максимальный размер тела запроса и загружаемых файлов
`disableProxy`	boolean	`false`	Отключить автоматическую регистрацию в `HttpProxy`
`useProxy`	boolean	`true`	Устаревший синоним `disableProxy` (инвертирован)

Основные методы

`addRoute(route)`

Добавляет маршрут Fastify. Принимает объект со свойствами `method`, `url` и `handler`. Если маршрут с таким методом и URL уже существует, он заменяется.

```
web.addRoute({
  method: 'GET',
  url: '/hello',
  handler: (req, reply) => reply.send({ hello: 'world' })
});
```

- URL автоматически дополняется префиксом `prefixUrl`.

`restart()`

Пересоздаёт экземпляр Fastify и заново регистрирует все накопленные маршруты. Полезно после массового добавления маршрутов.

Внутренние особенности

- Обработчик ошибок `setErrorHandler` оборачивает `ComplexError` в понятный JSON-ответ с HTTP-статусом из `options.httpStatus` и сообщением, если `showUser: true`.
- При запуске автоматически регистрируется в `HttpProxy` (если `disableProxy` не `true`), добавляя прокси-маршрут с адресом `http://127.0.0.1:<порт>`.
- Поддерживает загрузку файлов через `@fastify/multipart`.
- Включает Swagger UI по адресу `<prefixUrl>/documentation`.

FastifyGateway

`FastifyGateway` — наследник `Fastify`, реализующий **API Gateway**. Он автоматически создаёт HTTP-маршруты для всех запросов, зарегистрированных в `GlobalServices`. При появлении нового сервиса или изменении его схемы шлюз динамически перестраивает маршруты.

Подключение

```
import { FastifyGateway } from '@morphcluster/web';


const gateway = new FastifyGateway(host, {
  baseUrl: '/api',          // префикс всех маршрутов
  prefixUrl: '/gateway'     // собственный префикс Fastify (Swagger будет на /gateway/documentation)
});
```

Обычно `GlobalServices` передаётся через `host` или внедряется как опциональная зависимость `GlobalServices`. Также требуется `Sessions`, если не все запросы анонимны.

Как это работает

1. При старте `Gateway` подписывается на событие `onServiceChanged` реестра `GlobalServices`.
2. Для каждого сервиса и каждого его запроса, у которого задано свойство `http` (метод HTTP, например `'GET'`, `'POST'`), создаётся маршрут вида:

   ```
   <baseUrl>/<serviceName>/<requestName>
   ```

   Имена сервисов и запросов конвертируются из CamelCase в kebab-case (`MyService` → `my-service`, `doSomething` → `do-something`).
3. При получении HTTP-запроса шлюз:
   - извлекает токен сессии из заголовка `Authorization: Bearer <token>`;
   - если запрос не помечен как `anonymous`, проверяет сессию через сервис `Sessions`;
   - если требуется `needAdmin`, проверяет флаг `isAdmin` в сессии;
   - подготавливает параметры запроса (`request.body` + IP + данные сессии);
   - логирует вызов с указанием сервиса и метода (если не `noLogs`);
   - вызывает `service.sendRequest()` соответствующего `GlobalService`;
   - результат возвращает клиенту как JSON, либо статус 204, если ответа нет;
   - в случае ошибки оборачивает её в `ComplexError` и возвращает клиенту (если пользователь — админ, ошибка всегда показывается).

Тонкая настройка запросов

В схеме сервиса для каждого запроса можно указать:

- `http` — метод HTTP (`GET`, `POST`, `PUT`, `DELETE`). Если не задан, маршрут не создаётся.
- `anonymous` — разрешить доступ без токена.
- `needAdmin` — требовать права администратора.

- `noLogs` — отключить логирование вызова.
- `logLevel` — уровень логирования (по умолчанию 10).

Генерация Swagger

Для каждого созданного маршрута автоматически формируется схема:

- `summary` берётся из `description` запроса.
- Если запрос не `anonymous`, добавляется `security: [{ token: [] }]`.
- Для методов, отличных от `GET`, тело запроса описывается JSON-схемой `request`, из которой исключается поле `session`.

ServiceHealth

`ServiceHealth` — простой HTTP-сервис, предоставляющий информацию о состоянии всех сервисов в `ServiceHost`. Расширяет `Fastify` и добавляет один маршрут `GET /services`.

Подключение

```
import { ServiceHealth } from '@morphcluster/web';


const health = new ServiceHealth(host, { prefixUrl: '/health' });
```

Ответ `/services`

```
{
  "allStarted": false,
  "notStarted": ["SomeService"],
  "noRequirements": ["DependencyService"],
  "fullServices": [
    {
      "name": "MyService",
      "starting": false,
      "started": true,
      "requirements": []
    }
  ]
}
```

- `allStarted` — все ли сервисы запущены.
- `notStarted` — имена незапущенных сервисов.
- `noRequirements` — сервисы, чьи обязательные зависимости не загружены (но сами они не в `notStarted`).

- `fullServices` — детальная информация по каждому сервису.

HttpProxy

`HttpProxy` — обратный прокси-сервер, позволяющий объединить несколько внутренних HTTP-серверов (и WebSocket) под одним портом. Сам является сервисом MorphCluster и может динамически добавлять/удалять маршруты.

Подключение

```
import { HttpProxy } from '@morphcluster/web';


const proxy = new HttpProxy(host, { port: 80, host: '0.0.0.0', proxyTimeout: 60000 });
```

Конфигурация

Параметр	Тип	По умолчанию	Описание
`port`	number	обязательно	Порт, на котором будет слушать прокси
`host`	string	`'0.0.0.0'`	IP для прослушивания
`proxyTimeout`	number	`120000`	Таймаут проксирования запросов (мс)

Методы (доступны как запросы сервиса)

`addProxy({ url, address, type? })`

Добавляет новый прокси-маршрут.

- `url` — префикс пути, по которому прокси будет принимать запросы.
- `address` — целевой URL (например, `http://localhost:8080`).
- `type` — `'proxy'` (по умолчанию) или `'websocket'`.

`delProxy({ url })`

Удаляет маршрут по URL-префиксу.

`list()`

Возвращает массив всех текущих маршрутов.

Механизм работы

- Прокси слушает входящие HTTP-запросы и ищет маршрут с наиболее длинным совпадающим префиксом.
- Если маршрут найден, запрос проксируется на целевой сервер с добавлением заголовка `x-real-ip`.
- При ошибке соединения (ECONNREFUSED, ETIMEDOUT и др.) клиенту возвращается соответствующий HTTP-статус (502, 504) в формате JSON. Если ошибка `ECONNREFUSED` повторяется, прокси может автоматически удалить проблемный маршрут (опция включается принудительно в коде при 502).
- WebSocket-соединения проксируются с использованием `http-proxy` в режиме `ws`.
- Сервис также предоставляет эндпоинт `/ip`, возвращающий информацию об IP клиента (полезно для отладки заголовков `X-Forwarded-For` и `X-Real-IP`).

Автоматическая регистрация сервисов Fastify

Любой сервис, наследующий `Fastify`, при старте (если `disableProxy !== true`) автоматически регистрирует себя в `HttpProxy`, добавляя маршрут со своим `prefixUrl` и адресом `http://127.0.0.1:<порт>`. Это позволяет собирать все HTTP-интерфейсы под единым входным портом.

HttpStatic

`HttpStatic` — сервис для раздачи статических файлов (HTML, CSS, JS, изображений и т.д.). Поддерживает кэширование по ETag и автоматическую регистрацию в `HttpProxy` (опционально).

Подключение

```
import { HttpStatic } from '@morphcluster/web';


const staticServer = new HttpStatic(host, {
  prefixUrl: '/static',
  staticPath: './public',
  port: 8081,
  useProxy: true
});
```

Конфигурация

Параметр	Тип	По умолчанию	Описание
`prefixUrl`	string	`''`	URL-префикс, по которому доступна статика

Параметр	Тип	По умолчанию	Описание
`staticPath`	string	`'./static'`	Локальная директория с файлами (разрешается относительно `Config.packageRoot`)
`host`	string	`127.0.0.1`	IP для прослушивания
`port`	number	`0`	Порт
`useProxy`	boolean	`false`	Автоматически добавить прокси-маршрут в `HttpProxy`

Поведение

- Запросы к путям, начинающимся с `prefixUrl`, транслируются в файловую систему относительно `staticPath`. Например, запрос `/static/css/style.css` ищет файл `./public/css/style.css`.
- Если путь заканчивается на `/`, добавляется `index.html`.
- Поддерживается условный запрос через заголовок `If-None-Match`: сервер отправляет ETag, основанный на времени модификации файла, и при совпадении возвращает `304 Not Modified`.
- MIME-тип определяется по расширению файла с помощью библиотеки `mime-types`.
- Запрещены пути, содержащие `..`.

Swagger UI

Встроенный модуль Swagger UI реализован как Fastify-плагин (`fastifySwaggerUi`). Он автоматически добавляется ко всем сервисам, наследующим `Fastify`, и предоставляет интерфейс по адресу `<prefixUrl>/documentation`.

Файлы

- **`swagger-initializer.mjs`** – Генерирует JavaScript-код для инициализации Swagger UI с конфигурацией, полученной из OpenAPI-схемы Fastify.
- **`index-html.mjs`** – Генерирует HTML-страницу документации с подключением необходимых скриптов и стилей.
- **`serialize.mjs`** – Сериализует JavaScript-значения в строковый код для вставки в инициализирующий скрипт.

Особенности

- Swagger UI берёт спецификацию через эндпоинт `./json`, предоставляемый `@fastify/swagger`.
- Поддерживает кастомные темы через `opts.theme.css` и `opts.theme.js`.
- Все статические ресурсы Swagger UI (CSS, JS) встроены в бандл и раздаются виртуальными маршрутами Fastify (без необходимости отдельной папки).

Схемы сервисов

FastifyGateway (`src/fastify-gateway/service-schema.mjs`)

```
{
  "name": "FastifyGateway",
  "description": "HTTP шлюз для сервисов",
  "requests": [
    {
      "name": "recreateRoutes",
      "description": "Пересоздать маршруты",
      "anonymous": false,
      "needAdmin": true,
      "noLogs": true,
      "http": null,
      "request": {},
      "response": {}
    }
  ]
}
```

Запрос `recreateRoutes` позволяет администратору принудительно перестроить все маршруты шлюза.

HttpProxy (`src/http-proxy/service-schema.js`)

Содержит запросы:

- `list` – получить список маршрутов (только для админов).
- `addProxy` – добавить HTTP- или WebSocket-прокси.
- `addWebsocket` (устаревший алиас для `addProxy` с типом `websocket`).
- `addStatic` (зарезервирован, но не реализован в коде; в схеме описан).
- `delRoute` – удалить маршрут по URL.

Все административные запросы требуют сессии с правами администратора.

Экспорт

Пакет экспортирует все основные классы через `src/index.js`:

```
import {
  Fastify,
  FastifyGateway,
  ServiceHealth,
  HttpProxy,
  HttpStatic
} from '@morphcluster/web';
```

Вспомогательные функции и схема Swagger UI доступны для внутреннего использования, но не экспортируются наружу.

Интеграция с MorphCluster Core

Все описанные сервисы построены на базе `ServiceRequire`, поэтому могут использовать:

- `this.host.requireService(name)` / `this.host.waitService(name)` для доступа к другим сервисам.
- `this.Config` и `this.Logger` для конфигурации и логирования.
- Автоматическое внедрение зависимостей, указанных в `this.requirements` и `this.optionalServices`.

Для корректной работы в системе, использующей `GlobalServices`, необходимо, чтобы `HttpProxy`, `Sessions` (если нужна аутентификация) и все сервисы, к которым обращается `FastifyGateway`, были зарегистрированы в реестре.

Revision #2
Created 21 June 2026 15:35:19 by Георгий
Updated 26 June 2026 13:52:59 by Георгий