Хост Supervisor

Supervisor — это центральный управляющий хост в экосистеме MorphCluster, построенный на базе фреймворка `@morphcluster/core`. Он выполняет функции развёртывания, мониторинга и обновления микросервисных пакетов, а также предоставляет служебные сервисы (лицензирование, WebSocket-уведомления, проксирование и др.). Supervisor запускается как самостоятельный процесс Node.js и управляет жизненным циклом дочерних процессов и воркеров, в которых работают другие микросервисы.

Документация описывает внутреннее устройство Supervisor на основе предоставленного исходного кода.

Общая архитектура

Supervisor оформлен как главный модуль, создающий экземпляр `ServiceHost` и регистрирующий в нём все необходимые сервисы.

Каждый функциональный блок реализован в виде отдельного сервиса, наследующего `ServiceRequire` (или `Service`). Сервисы общаются между собой через механизм зависимостей (`requirements`) и вызовы запросов (`sendRequest`), а также через глобальные объекты (например, `GlobalServices`, `Config`, `Logger`).

Supervisor управляет **пакетами** — директориями, содержащими микросервисы (каждый пакет — это самостоятельное приложение, которое может быть запущено как отдельный процесс или воркер). Пакеты могут быть двух типов:

- **Дистрибутивные (dist)** — поставляются вместе с Supervisor и автоматически запускаются при первом старте.
- **Установленные вручную или через обновление** — находятся в `packagesDir`.

Сам Supervisor тоже является пакетом и может обновлять сам себя.

Точка входа и инициализация

Файл `src/start.mjs` выполняет следующие шаги:

1. Создаёт экземпляр `ServiceHost` с именем `"Supervisor"`.
2. Загружает конфигурацию (`Config`) из `config.mjs` и сохраняет в `host.Config`.
3. Настраивает корневой логгер `Logger`:
   - Читает параметры из `config.values.Logger`.
   - Добавляет бэкенды: `LoggerBackendConsole` и `LoggerBackendStore` (отправка логов в централизованное хранилище).
4. Регистрирует глобальный обработчик `uncaughtException`.

5. Последовательно добавляет все системные и пользовательские сервисы при помощи функции `addService(svcName, svcClass)`. Каждый сервис получает конфигурацию, объединяющую `config.values[svcName]` и `config.values.common`.
6. Запускает хост: `host.start(log)`.
7. После успешного старта выводит `"***All Services Started***"`.
8. Регистрирует обработчики сигналов `SIGINT` и `SIGTERM`, а также сообщения `shutdown` для воркеров, которые вызывают функцию `stop()`.

Функция `stop()` вызывает `host.stop()`, после чего завершает процесс с заданным кодом (по умолчанию 0). Таймаут принудительного завершения — 60 секунд.

Список системных сервисов, добавляемых в хост:

Имя в хосте	Класс / Назначение
`PostgresMigrator`	Миграция схем БД PostgreSQL
`LogStoreBackend`, `LogStoreAdmin`, `LogOraList`	Сервисы логирования (из `@morphcluster/logger`)
`NatsTimersPublisher`, `NatsTimersAggregator`	Мониторинг таймеров через NATS
`License`	Проверка и управление лицензией
`Sessions`, `Auth`	Хранение сессий и аутентификация
`Registry` (Dummy), `CommonRegistry`, `RegistryHelper`	Централизованное хранилище настроек
`GlobalServices`	Реестр глобальных сервисов
`HttpProxy`, `FastifyGateway`, `ServiceBridge`	HTTP-инфраструктура
`NatsConnection`, `NatsRequestListener`, `NatsEventPublisher`, `NatsSchemaPublisher`, `NatsSchemaListener`, `NatsHealthPublisher`, `NatsHealthSubscriber`, `NatsConnectionTest`	Подключение и маршрутизация через NATS
`Eventer`	WebSocket-сервер для real-time уведомлений
`PackageInfo`, `NodeHosts`, `Packages`, `PkgExtractor`, `PkgUpdater`, `PkgCleanup`	Управление пакетами (см. ниже)

Основные сервисы

PackageInfo

Файл: `src/PackageInfo/index.js`

Хранит информацию об установленных пакетах в JSON-файле `package-infos.json` в директории данных (`dataDir`). Каждая запись содержит имя пакета, источник установки, признак ручного управления и дату установки.

Запросы:

- `list` — возвращает список всех записей.

- `set({ name, ... })` — добавляет или обновляет информацию о пакете.
- `remove({ name })` — удаляет запись.
- `reload` — перечитывает файл с диска.

Зависимости: нет (кроме базового `ServiceRequire`). Используется другими сервисами для отслеживания статуса установки.

NodeHosts

Файл: `src/NodeHosts/index.js`

Управляет запущенными экземплярами пакетов — процессами и воркерами. Хранит массив объектов `ProcessMaster` или `WorkerMaster`.

Запросы:

- `list` — статусы всех запущенных хостов (id, тип, статус, ошибка).
- `startHost({ id, dir, dataDir, nodeExec, isWorker })` — запускает новый процесс или воркер.
- `stopHost({ id })` — останавливает процесс/воркер.
- `getStdOut({ id })` — возвращает накопленный вывод stdout/stderr.

При запуске `startHost` проверяет, запущен ли уже хост с таким ID. Если нет — создаёт экземпляр `ProcessMaster` или `WorkerMaster` в зависимости от флага `isWorker`, устанавливает переменные окружения (`MCL_CONFIG`, `MCL_DATA_DIR`, `TZ`) и запускает.

Зависимости: нет (базовый сервис).

Packages

Файл: `src/Packages/index.js`

Центральный сервис управления жизненным циклом пакетов. Отвечает за:

- Сканирование директорий с пакетами (`packagesDir` и `packagesDistDir`).
- Ведение списка автозапуска (`packages-autostart.json`).
- Запуск и остановку пакетов.
- Удаление пакетов.
- Перезапуск самого Supervisor при обновлении.

Запросы:

- `list` — возвращает список всех пакетов с их статусами, информацией из PackageInfo, автостартом.
- `startPackage({ id })` — запускает пакет. Если пакет с таким именем уже запущен (но с другой версией), останавливает старый и запускает новый, обновляя автостарт. Для supervisor выполняется обновление файла запуска и перезапуск всей системы.
- `stopPackage({ id })` — останавливает пакет и удаляет из автозапуска.

- `remove({ id })` — удаляет пакет (файлы) после остановки.
- `setManual({ pkgId, manual })` — устанавливает признак ручного управления (через PackageInfo).
- `restartSupervisor` — вызывает `HostStopper.stop(85)`.

Методы жизненного цикла:

- `start()`: создаёт директории пакетов, загружает список автозапуска и запускает все пакеты из него. Если файл автозапуска отсутствует, вызывает `createAutostart()`, который автоматически запускает все дистрибутивные пакеты.
- `stop()`: вызывает `super.stop()` (базовый).

Зависимости: `PackageInfo`, `NodeHosts`.

PkgExtractor

Файл: `src/PkgExtractor/index.js`

Распаковывает архивы `.7z` с пакетами, находящиеся в `dataDir/pkg-install`, в директорию пакетов.

Запросы:

- `extractAll` — извлекает все архивы. Сначала обрабатывает все пакеты, кроме supervisor, затем запускает каждый из них через `Packages.startPackage`. Supervisor извлекается и запускается последним.
- `extract({ pkgId, arcPath, source, manual })` — извлекает конкретный архив и регистрирует установку в `PackageInfo`.

Использует библиотеку `node-7z` для распаковки.

Зависимости: `Packages`, `PackageInfo`.

PkgUpdater

Файл: `src/PkgUpdater/index.js`

Сервис автоматического обновления пакетов с удалённого сервера обновлений. Работает по таймеру, настраиваемому через `CommonRegistry`.

Запросы:

- `check` — обращается к API сервера обновлений (`/api/main/csp-update/check`), передаёт системное имя, получает список доступных обновлений. Сравнивает с установленными версиями, отмечает флаги `downloaded` и `installed`.
- `downloadAll` — для каждого неустановленного пакета скачивает архив `.7z` в `pkgUpdaterDir`, затем вызывает `PkgExtractor.extract` и `Packages.startPackage` (с учётом

ручного режима). Старые версии пакетов удаляются через `Packages.remove`.

Зависимости: `Packages`, `PackageInfo`, `PkgExtractor`, `RegistryHelper`.

Конфигурация в `CommonRegistry` (схема `registry-schema.js`):

- `main.name` — системное имя (передаётся в запросы).
- `PkgUpdater.token` — токен авторизации.
- `PkgUpdater.timer` — интервал проверки обновлений в мс (по умолчанию 300000).

PkgCleanup

Файл: `src/PkgCleanup/index.js`

Сервис автоматической очистки устаревших пакетов (не запущенных и не помеченных как ручные, установленных через обновление). Периодичность задаётся в `CommonRegistry`, но в текущей реализации таймер закомментирован, очистка выполняется только при ручном вызове `cleanup`.

Запросы:

- `cleanup` — удаляет все неиспользуемые автообновлённые пакеты.

Зависимости: `Packages`, `PackageInfo`, `RegistryHelper`.

Eventer

Файл: `src/Eventer/index.mjs`

Реализует WebSocket-сервер для доставки real-time сообщений подключённым пользователям. Работает поверх библиотеки `ws`.

Запросы:

- `send({ userId, login, broadcast, channel, message })` — отправляет JSON-сообщение пользователям по фильтрам.
- `connectedUsers` — возвращает список авторизованных пользователей (уникальных).
- `requestConnections` — технический список всех соединений.

События:

- `onUserOnline` — генерируется при подключении первого соединения пользователя или отключении последнего.

Аутентификация: клиент после подключения должен отправить сообщение `{"type":"auth","token":"..."}`. Токен проверяется через сервис `Sessions`. После успешной аутентификации клиент может подписаться на каналы сообщением `{"type":"subscribe","channel":"..."}`. Внутренний класс `Connection` управляет состоянием одного

WebSocket-соединения.

Сервер запускается на выделенном порту (или 0, тогда назначается случайный), регистрируется в `HttpProxy` как WebSocket-прокси по префиксу `prefixUrl` (по умолчанию `/eventer`).

Периодически (каждые `pingTimeout` мс) проверяет живучесть соединений с помощью ping/pong.

Зависимости: `Sessions`.

License

Файл: `src/License/index.mjs`

Управляет файлом лицензии (`license.json`). Содержит поля `name`, `expire`, `key` (зашифрован). Периодически (каждые 60 секунд) проверяет изменение лицензии и генерирует событие `onLicenseChanged`.

Запросы:

- `getLicense` — возвращает объект с полями `name`, `expire`, `expired` (bool).
- `setLicense({ license })` — записывает новую лицензию в файл.

Зависимости: нет.

Вспомогательные классы управления процессами

Находятся в `src/lib/`. Иерархия: `BaseMaster` → `ProcessMaster`, `WorkerMaster`.

BaseMaster

Абстрактный класс, реализующий общую логику управления запущенным процессом или воркером:

- Хранение статуса (`running`, `stopping`, `restarting`, `error`, `null`).
- Буферизация stdout/stderr (ограничение по `maxLines`).
- Таймеры перезапуска (`restartMsec`) и принудительного завершения (`terminateMsec`).
- Методы `start()`, `stop()`, `restart()`, `terminate()`.
- `_sendStop()` — абстрактный метод отправки сигнала остановки.
- `_processStopped(e)` — обработчик завершения: если остановка ожидаемая, разрешает промис остановки; иначе запускает таймер перезапуска.

ProcessMaster

Наследует `BaseMaster`. Запускает дочерний процесс через `child_process.spawn`. Передаёт переменные окружения, рабочую директорию. При `stop()` отправляет `SIGTERM`, при `terminate()`

— `SIGKILL`.

WorkerMaster

Наследует `BaseMaster`. Запускает worker thread (`worker_threads.Worker`). Поддерживает передачу сообщений через `postMessage`. При `_sendStop()` отправляет сообщение `{ type: 'shutdown' }` в воркер. При `terminate()` вызывает `worker.terminate()`.

Взаимодействие с инфраструктурой MorphCluster

Supervisor активно использует следующие компоненты MorphCluster:

- **NATS** — для обнаружения сервисов на других узлах, публикации схем, событий, здоровья и каналов. Сервисы `NatsSchemaPublisher`, `NatsSchemaListener`, `NatsRequestListener`, `NatsEventPublisher`, `NatsHealthPublisher`, `NatsHealthSubscriber` делают локальные сервисы доступными удалённо.
- **GlobalServices** — реестр глобальных сервисов, скрывающий разницу между локальными и удалёнными реализациями.
- **HttpProxy / FastifyGateway** — единая точка HTTP-доступа ко всем сервисам. `FastifyGateway` создаёт маршруты вида `/<prefix>/<serviceName>/<requestName>`. Supervisor регистрирует `ServiceBridge` для ручного вызова сервисов.
- **Sessions / Auth** — хранение сессий и базовая аутентификация администратора.
- **CommonRegistry** — централизованные настройки (токены, интервалы).
- **Logger** — все логи Supervisor отправляются в централизованное хранилище через `LoggerBackendStore`.

Жизненный цикл пакетов

1. **Установка** — архив `.7z` помещается в `dataDir/pkg-install`. PkgExtractor распаковывает его в `packagesDir`, создавая директорию с именем `<name>-<version>`. В PackageInfo добавляется запись об установке.
2. **Первый запуск** — Packages сканирует директории, при отсутствии автозапуска запускает все дистрибутивные пакеты.
3. **Автозапуск** — список ID пакетов сохраняется в `packages-autostart.json`. При старте Supervisor запускает все пакеты из этого списка.
4. **Обновление** — PkgUpdater периодически опрашивает сервер обновлений, скачивает новые версии, устанавливает их рядом со старыми. При запуске новой версии старая останавливается, а после успешного старта — удаляется (если не помечена как manual).
5. **Ручное управление** — если пакет помечен `manual: true`, он не будет автоматически остановлен или удалён при обновлении. Пользователь сам управляет его запуском.
6. **Очистка** — PkgCleanup удаляет неиспользуемые автоустановленные пакеты.

Конфигурация

Файл `src/config.mjs` содержит значения по умолчанию, которые могут быть переопределены через внешний файл конфигурации (`Config.load()`). Основные параметры:

- `Logger.*` — настройки логирования.
- `Sessions.pg` — подключение к PostgreSQL для сессий.
- `NatsConnection.queue` — группа очереди NATS.
- `Packages.packagesDir` — путь к директории с установленными пакетами (по умолчанию `./packages`).
- `PkgUpdater.updateServerUrl` — URL сервера обновлений.
- `Eventer.port`, `prefixUrl`, `pingTimeout` — параметры WebSocket-сервера.

Процесс остановки

1. При получении сигнала или вызове `HostStopper.stop()` запускается `stop()` в `start.mjs`.
2. Устанавливается 60-секундный таймаут, после которого процесс будет принудительно завершён, а список незавершённых сервисов выведен в консоль.
3. Вызывается `host.stop()`, который последовательно останавливает все сервисы (каждый сервис вызывает остановку своих процессов/воркеров через `NodeHosts`).
4. После успешной остановки процесс завершается с заданным кодом.

`HostStopper` — глобальный синглтон, позволяющий другим сервисам (например, `Packages.restartSupervisor`) инициировать остановку всего хоста с определённым кодом (85 для перезапуска supervisor).

Revision #1
Created 21 June 2026 20:27:40 by Admin
Updated 26 June 2026 15:00:34 by Admin