Фреймворк Morphcluster 2

MorphCluster - это система организации структуры и взаимодействия внутри системы из множества процессов.

Задачи системы:

- Автоматическая организация связи внутри системы. MorphCluster находит и организует связь между различными частями системы в микросервисной архитектуре.
- Организация внутреннего контура. Разделение системы на внешний и внутренний трафик. Внешний трафик должен быть с защищен от злоумышленников, тогда как взаимодействие между доверенными частями системы должно происходить максимально быстро.
- Отслеживание состояния системы - позволить следить за тем, как работает система, выявлять и локализовать ошибки и зависания в реальном времени
- Тестирование. Система позволяет запускать отдельные части системы, чтобы проверить их работу, в процессе работы всей системы
- *** Масштабирование
- *** Модульность и взаимозаменяемость
- *** Возможность дробления и слияния модулей
- *** Возможность защиты исходного кода

Структура ядра

Сервис

Сервисы определяют неделимую единицу функционала приложения.

Имя сервиса идентифицирует его в системе и должно быть уникально.

Сервис, определенный, как локальный будет доступен только в пределах своего хоста. На разных хостах могут быть одноименные локальные сервисы. Локальные сервисы недоступны другим хостам

Сервис имеет состояние, показывающее, когда он будет готов к работе. В случае сбоя — сервис может сообщить о невозможности продолжить работу, и о восстановлении такой возможности (например, подключение к базе). Для выполнения инициализации своего сервиса нужно перегрузить метод `start`.

```
import {Service} from "@morphcluster/core"
export default class Test extends Service {
  async start() {
    console.log("Starting")
  }
}
```

`stop` работает по аналогии

Запросы позволяют выполнять специализированное действие, для управления сервисом или получения данных из него или с его помощью.

События вызываются изнутри сервиса и другие сервисы могут быть на него подписаны, для создания действия, во время выполнения события.

Управления хостом. Сервис имеет ограниченный интерфейс, для того чтобы определять, какие сервисы, которые находится на одним с ним хостом запущены, и для получения прямого доступа к этим сервисам.

Поучить запущенный сервис вручную из `ServiceHost` можно через метод `requireService`. Если сервис не закончил запуск, он не вернется.

Также это позволяет сервисы запускать дочерние сервисы, которые создаются внутри сервиса а также отслеживать состояние тех сервисов которые нужны для работы текущего сервиса.

Зависимости сервиса - это имена сервисов расположеных локально (на одном хосте с описываемым), которые необходимы для корректной работы описываемого сервиса. Они будут запущены до запуска текущего сервиса, и автоматически подключены.

Чтобы объявить зависимости сторонние сервисы в своем сервисе, можно наследовать свой класс от `ServiceRequire`. Перед запуском, `ServiceRequire` дождется запуска всех своих зависимостей.

```
import {ServiceRequire} from "@morphcluster/core"

export default class HelloWorlds extends ServiceRequire {
  constructor(host) {
    super(host)
    //Объявление поля для ссылки на зависимый сервис
    //Для работы подсказок внутри IDE и лучшей читаемости кода
    /** @type {Test} */
    this.HelloWorld = null
    //Список зависимостей
    this.requrements = [ "HelloWorld" ]
  }
  async start() {
    await super.start()
    //Вызов метода из другого сервиса, после super.start
    await this.Test.test({}, "common", null)
  }
}
```

Клиент – это способ взаимодействия между сервисами, который описывает схему сервиса, которая нужна текущему сервису, в котором он объявляен. Клиент позволяет организовывать зависимости с удаленными сервисами расположенными на другом хосте, также если клиент ссылается на сервис того же хоста, то вызов будет налажен без использования транспорта

Таймер позволяет удобно запускать периодические действия внутри сервиса, инициированные с определенным промежутком времени.

Триггер позволяет инициировать внутри сервиса действие, например, в случае получения запроса из внешней системы. Основные компоненты системы.

Хост

Корневой модуль запускаемый в процессе, который хранит остальные сущности и и управляет запуском системы. ***Разделение системы на хосты Готовые модули***

```
import {ServiceHost} from '@morphcluster/core'
host = new ServiceHost()
host.addService(new HelloWorld(host))
```

Транспорт

Транспорт - абстракция для определения способа взаимодействия между хостами

Контекст

Контекст определяет и идентифицирует действия в системе, и позволяет отслеживать ход их выполнения, в процессе которого может вызыватся действия в разных частях системы, в т.ч. в другом хосте.

- Монитор выполнения - позволяет понять в каком сейчас состоянии находится данный контекст, также можно понять где он сейчас находится
- Журнал выполнения - позволяет отслеживать действия которые были завершены, и просматривать отладочные данные
- Прерывание. По данным идентификации контекста можно послать сигнал прерывания, который сообщит внутрь действия том, что действие надо прервать, вызвав исключение.
- Информация о вызове. Внутри контекста хранится информация о том, кто создал этот контекст
- Данные о пользователе, если это внешний вызов. Это позволяет ограничивать доступ к системе и данным.

Схема сервиса

Схема сервиса - это структурированные данные, описывающая сервис для взаимодействия между ними и визуализации сервисов. Она описывает все способы взаимодействия с сервисом для внутреннего контура.

Схема используется для передачи между хостами, возможностей сервисов.

Схема описывается через json такого вида:

```
{
   name:"MyService",
   requests: [
      {
         name:"run",
         http: 'POST',
         request:{
            "type": "object",
            "properties": {
```

```
                "name": { "type": "string" },
                "params": {},
            },
            "required": [ "name" ]
        },
        response:{},
        description:"Выполнить метод",
    }
  ]
}
```

Схему целиком можно установить через метод `setSvcSchema(serviceSchema)`. Через метод `getSvcSchema()` можно получить текущую схему, вместе с добавленными изменениями отдельно.

Исключения

В отличии от стандартных исключений, исключения в фреймворке лучше вызывать классом `ComplexError`:

```
import { ComplexError } from '@morphcluster/core'
...
const name = "Test" //Имя исключения
//Дополнительные данные исключения в виде объекта (опционально)
const payload = { "mydata":123 }
//Опции
const options = {
  //Показывать ли внешнему серверу (FastifyRest) содержимое ошибки
  //Если пользователь - администратор, содержимое все равно будет показано
  "showUser": true,
//Можно перегрузить HTTP код возврата в FastifyRest, по умолчанию 500

  "httpStatus": 403,
  //Если цепочку логов создавали в этом методе, можно передать ее id в исключении
  //FastifyRest вернет этот ID, чтобы цепочку было проще найти
  "logId": "..."
}
throw new ComplexError( "Сообщение исключения", name, payload, options )
```

Данные внутри такого исключения будут переданы в т.ч. и удаленным сервисам

Конфигурация

Журналирование

Основные сервисы ядра

Registry

Реестр позволяет хранить данные в структурированном виде доступны любому сервису. Используются для настройки системы. В реестре для доступа данных необходимо заполнить схему реестра, отправить ее, которая заявит о необходимости заполнения этих полей и создаст для них описание и тип. С помощью локального сервиса RegistryHelper можно легко получить актуальный реестр без обработки событий.

Nats

NATS реализует транспорт через брокер.

Logger

Логгер хранит весь процесс выполнение контекста для отслеживания уже выполненных контекстов, выполненных запросов или иных вызовов системы.

AdminPanel

Сервисы шлюза

Proxy

Единая точка входа для http и websocket

Fastify

Один из вариантов коммуникации с клиентами и сторонними сервисами - веб сервер Fastify. Сервис-обвертка инициализирет библиотеку fastify с расширениями: `@fastify/cors, @fastify/swagger, @fastify/swagger-ui, @fastify/multipart`. В отличии от оригинального Fastify, сервис позволяет динамически добавлять и удалять новые маршруты без прерывания работы запущенных запросов после запуска сервера. Также реализована плавная(greceful) остановка сервиса.

Для начала, надо добавить пакет из corelibs:

```
npm install ./corelibs/fastify
```

Пример работы с Fastify:

```
import {Fastify} from  '@morphcluster/fastify'
...
//Добавление Fastify к хосту
let fastify = new Fastify( host, {"host" : "127.0.0.1", "port" : 80} )
host.addService(fastify)
...
//Добавление маршрута, в формате fastify
fastify.addRoute({...})
//Запуск пересоздания машрутов, чтобы отобразились изменения. Начатые вызовы сброшены не будут
await fastify.restart()
```

Формат для добавления новых маршрутов можно посмотреть в официальной документации

Gateway

Шлюз это HTTP сервер, который позволяет общаться из внешней среды с сервисами. Чтобы запрос сервиса был доступен через шлюз, надо описать его схеме, что он доступен во внешнем контуре. Также внешний контур проверяет наличие сессии. При отсутствии сессии будут работать только анонимные методы.

Чтобы публичные запросы сервисов хоста были доступны по http(s), можно воспользоваться сервисом Gateway. Он автоматически обнаружит запросы сервиса к публикации и создаст маршруты.

```
import {ServiceHost} from '@morphcluster/core'
import {Fastify,FastifyRest} from  '@morphcluster/fastify'
...
//Fastify - зависимость FastifyRest
host.addService(new Fastify( host, {"host" : "127.0.0.1", "port" : 80} ))
//Добавление FastifyRest
host.addService(new FastifyRest( host ))
//Для публикации, сервис должен обязательно быть добавлен в ServiceHost
host.addService(new Test())
```

Проверить работу можно будет по ссылке `http://127.0.0.1/documentation`. Там автоматически разворачиватся OpenApi/Swagger со всеми доступными маршрутами. Вызов, производимый извне имеет проверку доступа, при этом запросы между сервисами не замедляются этой проверки, сессия передается напрямую.

Sessions

Сессии соответственно хранят информацию о доступе, подтвержденную токеном. Токен создается при авторизации или иным способом. Пользователю или внешнему сервису нужен токен для взаимодействия с системой через шлюз. Система через него идентифицирует пользователя. Сессия может быть определена как привилегированная(isAdmin), что расширяет возможности такого пользователя.

Auth

Eventer

Bridge

Мост. Мост позволяет общаться привилегированным сессиям с внутренним контуром, недоступным через шлюз.

Компоненты CSP

Компоненты CSP запускаются поверх основных компонентов и запускаются для взимодействя с информацонной системой.

Основные компоненты

- Document Helper это локальный сервис который организовывает удобное взаимодействие с информационными объектами внутри баз данных.
- Eventer - сервис, позволяющий системе отправлять события (в т.ч. сообщения) в клиенту через websocket, так как HTTP шлюз не поддерживает обратного взаимодействия с клиентами.
- Users
- CarabiAuth это средство авторизации, использующее внутренних пользователей информационной системы.

Система хранения файлов

- Хранилище файлов позволяет прикреплять файлы к информационным объектам и другим сущностям и хранить эти файлы удаленно от самой системы
- StorageMaster это сервис который управляет запросами на загрузку и отправку сообщений загрузку и отправку на сервер файлами знает где хранится файл (???)
- Storage Host - это отдельная программа которая обеспечивает хранение файлов, и непосредственно и связывается со Storage Master для взаимодействия с системой. С самой системой помимо StorageMaster StorageHost не взаимодействует.
- СервисFiles позволяет пользователям взаимодействовать с хранилищем файлов.
- StorageHelper это локальный сервис, который помогает в получении файлов, организуя всю цепочку общения между различными компонентами хранилища файлов.
- StorageBlob -

Система генерации отчетов

Интеграции с внешними сервисами

- DaData
- Mailer
- SMS
- Uiscom
- TelegramBot
- Firebase