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

Revision #15
Created 12 October 2024 18:28:50 by Георгий
Updated 17 June 2026 18:11:53 by Admin