

Н·МЕДИАТОР

медиатор бизнеса

информационные системы и
электронные компоненты

Функциональные характеристики платформы: разработка
информационных систем

Версия: 1.0

Содержание

1. Назначение документа	3
2. Базис для разработки информационных систем.....	3
3. Создание модуля.....	5
4. Создание фонового процесса.....	7

1. Назначение документа

С точки зрения функциональных характеристик платформу можно рассматривать с трех аспектов - как платформу разработки информационных систем, пользовательского интерфейса в качестве базового функционала, поставляемого вместе с платформой, и в качестве API базового функционала. Данный документ описывает платформу с точки зрения разработки информационных систем и возможностей, которые предоставляет платформа для реализации новых подсистем и модулей на основе шаблонов программирования.

2. Базис для разработки информационных систем

С платформой поставляются шаблоны для разработки модулей платформы на языке java, позволяющие оперативно разрабатывать и модули и вводить их в эксплуатацию. Поставляются шаблоны для быстрой разработки модулей и фоновых процессов, реализующих специализированное универсальное api.

Для классической среды сборки maven можно указать репозитории nmediator для работы с требуемыми артефактами или вручную установить артефакты в локальный репозиторий .m2. В качестве примера работы с репо nmediator привожу settings.xml, который должен размещаться в локальном репозитории .m2. Важно: репозиторий nmediator автоматически кэширует все запросы на получение артефактов так, что сборка возможна в отсутствии связей с внешними репозиториями apache и maven:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0
http://maven.apache.org/xsd/settings-1.1.0.xsd">

  <servers>
    <server>
      <id>nmediator-nexus-snapshots</id>
      <username>xxx</username>
      <password>yyy</password>
    </server>
    <server>
      <id>nmediator-nexus-releases</id>
      <username>xxx</username>
      <password>yyy</password>
    </server>
    <server>
      <id>nmediator-private</id>
      <username>xxx</username>
      <password>yyy</password>
    </server>
  </servers>

  <profiles>
    <profile>
      <id>use-multiple-repos</id>
      <repositories>
        <repository>
          <id>nmediator-public</id>
          <name>NMediator public</name>
          <url>https://nexus.nmediator.ru/repository/maven-public/</url>
        </repository>
        <repository>
          <id>nmediator-private</id>
          <name>NMediator private</name>
          <url>https://nexus.nmediator.ru/repository/maven-private/</url>
        </repository>
        <repository>
          <id>clojars.org</id>
          <name>clojars.org</name>
          <url>https://repo.clojars.org/</url>
        </repository>
      </repositories>
    </profile>
  </profiles>

  <mirrors>
    <mirror>
      <id>nmediator-public</id>
```

```

        <name>central</name>
        <url>https://nexus.nmediator.ru/repository/maven-public/</url>
        <mirrorOf>central,repo.jenkins-ci.org</mirrorOf>
    </mirror>
</mirrors>

<activeProfiles>
    <activeProfile>use-multiple-repos</activeProfile>
</activeProfiles>

</settings>

```

Для реализации модуля платформы необходимо включить артефакт в проект:

```

<dependency>
    <groupId>ru.nmediator</groupId>
    <artifactId>nms.template</artifactId>
    <version>1.1</version>
</dependency>

```

Структура каталога шаблона представляет следующий вид (полный код шаблона поставляется вместе с платформой через git):

```

config
container_start.sh
container_stop.sh
docker
docker_env.sh
image_build.sh
image_push.sh
image_remote_start.sh
image_remote_stop.sh
image_save_local.sh
image_save.sh
image_start.sh
image_stop.sh
pom.xml
readme.md
src

```

Разработчику необходимо скорректировать файлы конфигурации для 3х сред (dev, test, prom) в каталоге config, включить необходимость поддержки базы данных, указать функции, которые будет реализовывать модуль, название модуля и прочее. В файле docker_env.sh необходимо откорректировать название модуля, название артефакта (jar), который будет использоваться как код модуля, и тип среды (DEV, TEST, PROM). В каталоге docker в файле Dockerfile скорректировать компоненты, которые, возможно, потребуются в работе модуля (в основном это не требуется). Сборка проекта осуществляется при помощи maven командой mvn install. Для локальной работы без docker репозитория нужно вызвать image_build.sh и image_save_local.sh, в результате которого на диск будет сохранен образ контейнера docker для запуска, который нужно перенести в соответствующий модулю каталог ~/sigma/components/XXXXX и вызвать ./deploy.sh XXXXX для загрузки модуля. Каталог ~/sigma/components/XXXXX содержит файлы start.sh, stop.sh, которые вызываются при запуске и остановке контейнера. Разработчик может самостоятельно указать специфические параметры запуска и остановки (обычно это не требуется), Файл docker_env.sh нужно скопировать из проекта в каталог ~/sigma/components/XXXXX. Для test и prom сред можно вызвать image_push.sh(требуется настроить параметры), при этом контейнер будет отправлен в репозиторий и может быть загружен средой, использующей репозиторий: в файле deploy.sh можно установить переменную:

```
export USEREPO=true
```

, которая говорит о необходимости загрузки образа из репо docker.

3. Создание модуля

3.1. Для создания модуля разработчик создает новый проект, включает в него шаблон модуля платформы

3.2. Создается класс конфигулятора, который может переопределять стандартные параметры и/или добавлять новые. С документацией на CConfigManager можно ознакомиться в публичном git nmediator.

```
import ru.nmediator.nms.template.impl.CConfigImpl;
import ru.nmediator.nms.users.functions.interfaces.IFunctionsConfig;
import ru.nmediator.nms.users.workers.cleaner.interfaces.ISessionCleanerConfig;
import ru.pcc.config.manager.annotations.AConfigInit;
import ru.pcc.config.manager.annotations.AConfigParameter;
import ru.pcc.config.manager.enums.EMapTypes;

@AConfigInit(configfile = "nms-users.conf", varprefix = "nms.", pathprefix = "./")
public class CConfigImplOverride extends CConfigImpl
implements IFunctionsConfig,
ISessionCleanerConfig {
    public CConfigImplOverride() {
        super();
    }
    /**
     * // IFunctionsConfig
     * @AConfigParameter(mapvalue = "users.session.ttl.max.sec", maptype = EMapTypes.INT,
     * idefaultvalue = 3600)
     * public int m_siUsersSessionTtlMaxSec;
     *
     * @Override
     * public int GetUsersSessionTtlMaxSec() {
     *     return m_siUsersSessionTtlMaxSec;
     * }
     *
     * // ISessionCleanerConfig
     * @AConfigParameter(mapvalue = "session.cleaner.start.delay.ms", maptype = EMapTypes.INT,
     * idefaultvalue = 35000)
     * public int m_siSessionCleanerStartDelayMs;
     *
     * @AConfigParameter(mapvalue = "session.cleaner.timeout.ms", maptype = EMapTypes.INT,
     * idefaultvalue = 120000)
     * public int m_siSessionCleanerTimeoutMs;
     *
     * @AConfigParameter(mapvalue = "session.cleaner.threads.max", maptype = EMapTypes.INT,
     * idefaultvalue = 1)
     * public int m_siSessionCleanerMaxThreads;
     *
     * @AConfigParameter(mapvalue = "session.cleaner.enabled", maptype = EMapTypes.LOGIC,
     * idefaultvalue = true)
     * public boolean m_lSessionCleanerEnabled;
     *
     * @Override
     * public int GetSessionCleanerStartDelayMs() {
     *     return m_siSessionCleanerStartDelayMs;
     * }
     *
     * @Override
     * public int GetSessionCleanerTimeoutMs() {
     *     return m_siSessionCleanerTimeoutMs;
     * }
     *
     * @Override
     * public int GetSessionCleanerMaxThreads() {
     *     return m_siSessionCleanerMaxThreads;
     * }
     *
     * @Override
     * public boolean GetSessionCleanerEnabled() {
     *     return m_lSessionCleanerEnabled;
     * }
```

```

}
/*****/
}

```

3.3. Создается класс логгера или используется стандартный логгер платформы (CLoggerImpl.class). Не рекомендуется переопределять логгер, поскольку информация о логах консолидируется в системе логгирования loki и доступна для анализа и используется в том числе для динамического масштабирования системы в SWARM.

3.4. Реализуется класс, прослушивающий входящие запросы и отвечающий на него при необходимости

```

public class CmsJsonMessageClientImpl extends CmsJsonMessagesClient {
    @Override
    public void ProcessMessage(String sTopic, CIncomeMsRequest oIncomeMsRequest,
    COutgoMsRequest oOutgoMsRequest) {
        if ("ping".equals(sTopic)) {
            oOutgoMsRequest.SetResult(true);
            JSONObject joResponse = new JSONObject();
            joResponse.put("ping", "pong");
            oOutgoMsRequest.SetPayload(joResponse);
        }
    }
}

```

3.5. Реализуем поток для останова модуля

```

import ru.nmediator.nms.template.core.CmsCore;

public class CShutdownThread extends Thread
{
    private CmsCore m_oMsCore;
    public CShutdownThread(CmsCore oMsCore)
    {
        m_oMsCore = oMsCore;
    }
    @Override
    public void run()
    {
        m_oMsCore.Shutdown();
    }
}

```

3.6. Реализуется класс с функцией main, в котором создается объект CmsCore. В конструктор передается класс, который используется в качестве конфигуратора(CConfigImplOverride) и класс логгера(CLoggerImpl.class)

```

public static void main(String[] args)
{
    ....
    msCore = new CmsCore(
        CConfigImplOverride.class, CLoggerImpl.class
    );
    if (!msCore.IsInitialized()) {
        Logger.getGlobal().log(Level.SEVERE, "CmsCore is not initialized");
        break ErrorPoint;
    }
    ....
}

```

3.7. Регистрируется прослушивающий класс

```

.....
CMSJsonMessageClientImpl oMessageClientImpl = new CMSJsonMessageClientImpl();
for (String sTopic: GetMsConfig().GetMsFunctions()) {
    oMsCore.Subscribe(oMessageClientImpl, sTopic);
}
.....

```

3.8. Создаем поток останова

```

try {
    Runtime.getRuntime().addShutdownHook(new CShutdownThread(oMsCore));
    while (true) {
        Thread.sleep(10000);
    }
} catch (Exception e) {
    GetLogger().WriteToLogWithTrace(Level.SEVERE, StackTraceToString(e));
}

```

Новый модуль готов к работе, на запрос ping модуль ответит pong на запрашивающую сторону. Для теста модуль можно запустить без контейнеризации java -jar module.jar положив файл конфигурации с соответствующей платформой в каталог запуска. При этом необходимо в файле конфигурации указать внешний адрес шины kafka, поскольку модуль будет запущен вне подсети docker.

```

#####
# Конфигурация Kafka
#####
#
nms.kafka.server.address=192.168.110.174:9092
#параметр nms.kafka.server.address указывает непосредственно на
#ip контейнера, а не на название контейнера внутри подсети docker
#kafka-container:9092
nms.kafka.group=nms
nms.kafka.acks=all
nms.kafka.block.max.ms=1000
nms.kafka.idle.max.ms=5000

```

4. Создание фонового процесса

Для разработки фонового процесса в прокат необходимо включить артефакты

```

<dependency>
  <groupId>ru.nmediator</groupId>
  <artifactId>nms.worker.template</artifactId>
  <version>1.0</version>
</dependency>
<dependency>
  <groupId>ru.nmediator</groupId>
  <artifactId>sigma.worker.interface</artifactId>
  <version>1.0</version>
</dependency>
<dependency>
  <groupId>ru.nmediator</groupId>
  <artifactId>nms.template</artifactId>
  <version>1.1</version>
  <scope>compile</scope>
</dependency>

```

4.1. Пользователь реализует стандартный интерфейс API фоновых процессов (С описанием API фоновых процессов можно ознакомиться на сайте)

```

public class CWorkerApiImpl extends CWorkerLogger implements IWorkerApi {
    private HashMap < String, CSettingsItem > settings = new HashMap < String,
CSettingsItem > () {
        {
            put("число", new CSettingsItem("число", "заполнить число", EDataType.INTEGER,
Integer.valueOf(1)));

```

```

        put("строка", new CSettingsItem("строка", "заполнить строку", EDataType.STRING,
"Test string"));
        put("Логическое что-то такое", new CSettingsItem("Логическое что-то такое",
"выбери меня из чекбокса", EDataType.BOOLEAN, false));
        put("Даточка", new CSettingsItem("Даточка", "пожалуйста, заполните дату и
время", EDataType.DATETIME, new java.util.Date()));
        put("мстрок", new CSettingsItem("мстрок", "массив строк для заполнения. как это
реализовать? списком? Сортировать низя!", EDataType.STRINGARRAY, new ArrayList <
String > (Arrays.asList(new String[] {
                "aa",
                "bb",
                "ccc",
                "куку"
            })))));
        put("мчисел", new CSettingsItem("мчисел", "массив чисел для заполнения. как это
реализовать? списком? Сортировать низя!", EDataType.INTEGERARRAY, new ArrayList <
Integer > (Arrays.asList(new Integer[] {
                1,
                3,
                2,
                5
            })))));
    }
};

private EStatus status = EStatus.E_WAITING;
private ArrayList < CLogRecord > logRecords = new ArrayList < CLogRecord > ();

public CWorkerApiImpl() {
    super(CMSScore.GetConfig().GetMsName());
}

@Override
public String getName() {
    return "Фоновый процесс test1";
}

@Override
public String getDescription() {
    return "Для тестирования";
}

@Override
public EStatus getStatus() {
    return status;
}

@Override
public HashMap < String, CSettingsItem > getSettings() {
    addLog(Level.INFO, "getSettings called");
    return settings;
}

@Override
public void setSettings(HashMap < String, CSettingsItem > hashMap) {
    addLog(Level.INFO, "setSettings called");
    settings = hashMap;
}

@Override
public void updateSettings() {
    addLog(Level.INFO, "updateSettings called");
}

@Override
public void reloadSettings() {
    addLog(Level.INFO, "reloadSettings called");
}

@Override
public void startNow() {
    status = EStatus.E_EXECUTING;
    addLog(Level.INFO, "startNow called");
}

@Override
public void shutdown() {
    status = EStatus.E_SUSPENDED;
    addLog(Level.INFO, "shutdown called");
}

```

```
}  
  
@Override  
public void resume() {  
    status = EStatus.E_WAITING;  
    addLog(Level.INFO, "resume called");  
}  
}
```

4.2. Аналогично сценарию, описанному для модуля, реализуется класс конфигурации и логгера

4.3. Реализуется функция main. Всю остальную работу выполняет шаблон, разработчку нужно сконцентрироваться лишь на реализации функций api IWorkerApi

```
public static void main(String[] args) {  
    self = new CTest1Worker();  
    self.initCore();  
    self.api = new CWorkerApiImpl();  
    self.getShutdownList().add(self.api);  
    self.publicClient = new CMessageClientPublicImpl(self.api);  
    self.privateClient = new CMessageClientPrivateImpl(self.api);  
    self.start();  
}
```