aas-config插件开发文档

介绍

aas-config.jar是一个专为Apusic敏捷版服务器（简称AAMS）设计的插件，旨在简化与Nacos配置中心的交互过程。借助此插件，AAMS能够轻松地从Nacos配置中心获取配置信息，并实现配置的自动更新和同步。该插件适用于所有希望通过Nacos管理配置的Java应用，无论是微服务架构还是传统的单体应用。

功能特点

- **简化配置管理**：提供简洁的API，使得从Nacos配置中心获取和更新配置变得简单快捷。
- **动态配置更新**：支持配置更新的动态监听，当Nacos中的配置发生变化时，应用能够实时响应并更新本地配置。

目标读者

本文档面向有意使用aas-config.jar插件进行开发的Java程序员，读者应该对Java有基本的了解。

快速入门

以下是使用aas-config.jar进行基本配置获取操作的快速指南。

环境要求

- Java 8 或更高版本
- AAMS-V10大于等于SP9
- Nacos Server (确保Nacos服务已启动并可访问)

安装

- 确保${apusic.base}/conf下有configs.xml文件（ant构建时**已自动导入**）。
- 将aas-config.jar导入到AAMS的classpath（${apusic.base}/lib目录，这一步在使用ant构建时**已经自动导入**到到lib下）。
- 确保${apusic.base}/plugins/config目录下存在对应的jar包（ant构建时**已自动导入**）。
- 配置环境变量CONFIG_CENTER_ENABLE=true或者JVM参数-Dconfig.center.enable=true启动配置中心插件。
- 在${apusic.base}/conf目录中apusic.properties文件的common.loader参数添加导入类："${apusic.base}/plugins/config/*.jar"。

进入${apusic.base}/bin目录启动AAMS：

Windows

```
apusic.bat run
```

使用示例

以下是如何使用`aas-config.jar`对接配置中心的一个简单示例：

设置环境变量或者JVM参数

- Window
  - 环境变量（大小写不敏感）

  进入终端设置环境变量或者去设置中配置

  ```
  set CONFIG_CENTER_ENABLE=true或者set config_center_enable=true
  ```

  - JVM参数（大小写敏感）

  在`${apusic.base}/conf/apusic.bat`中添加参数`-Dconfig.center.enable=true`即可。

aas-config.jar插件配置

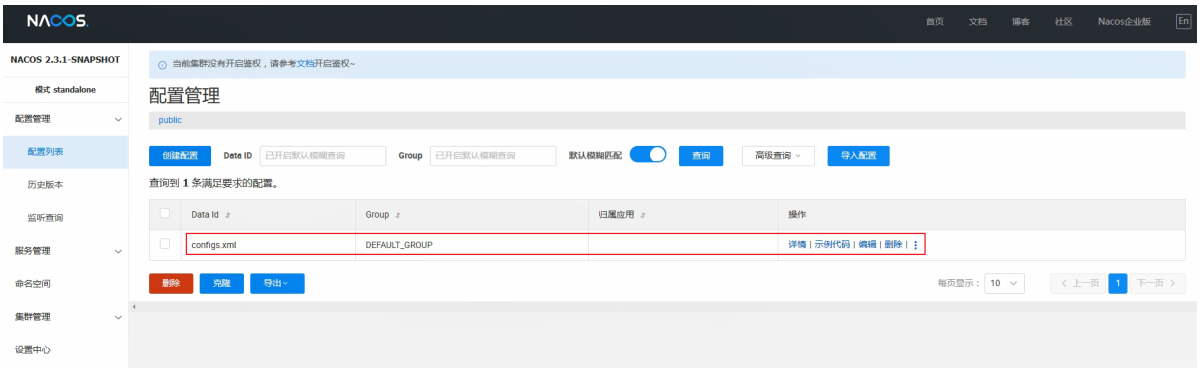
主要配置项包括：nacos服务端地址及端口、配置ID、配置分组、读取配置超时时间以及总配置文件名

配置项	配置参数	默认值	备注
nacos服务端地址及端口	环境变量：CONFIG_CENTER_ADDR=127.0.0.1:8848 JVM参数：-Dconfig.center.addr=127.0.0.1:8848	127.0.0.1:8848	
配置ID	环境变量：com.apusic.config.ConfigCenter.dataId=configs.xml JVM参数：-Dcom.apusic.config.ConfigCenter.dataId=configs.xml	configs.xml	
配置分组	环境变量： com.apusic.config.ConfigCenter.group=DEFAULT_GROUP JVM参数：- Dcom.apusic.config.ConfigCenter.group=DEFAULT_GROUP	DEFAULT_GROUP	
读取配置超时时间	环境变量：com.apusic.config.ConfigCenter.timeoutMs=3000 JVM参数：-com.apusic.config.ConfigCenter.timeoutMs=3000	3000	nacos官方推荐值
总配置文件名	环境变量： com.apusic.config.ConfigCenter.configsFileName=configs.xml JVM参数：- Dcom.apusic.config.ConfigCenter.configsFileName=configs.xml	configs.xml	

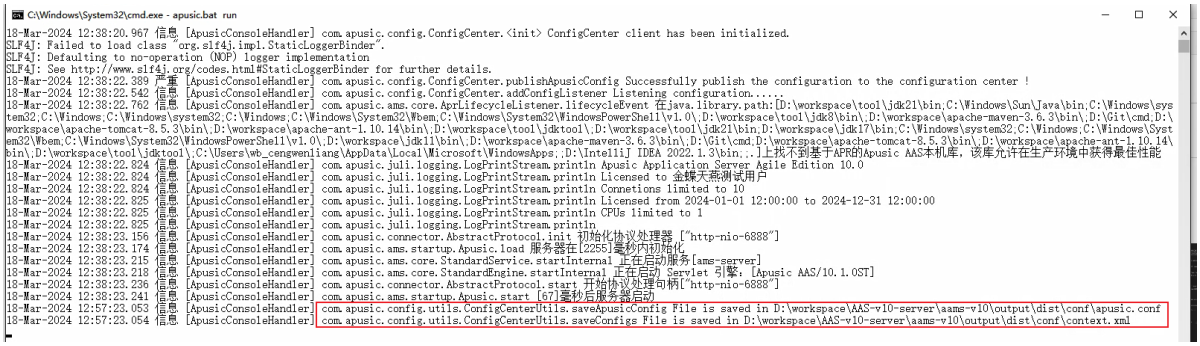
启动AAMS

输入`apusic.bat run`启动aams即可。

随后前往nacos配置中心地址即可看到已经发布的配置



点击编辑对配置进行修改后再次发布，对应的修改将会保存到`${apusic.base}/conf`目录下的文件中。



修改成功后即可在日志中看到修改成功的配置文件保存位置。

configs.xml文件格式

插件首先将需要放入配置中心进行配置发布、修改以及监听的配置放入configs.xml文件中，格式如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<configs>
<!-- config标签的name属性存放对应文件名，修改后的配置将会保存到name属性对应的文件名中     -->
<config name="apusic.conf">
    <!-- apusic.conf文件中的相关配置     -->
</config>
<config name="context.xml">
<Context>
    <!-- context.xml文件中的相关配置     -->
</Context>
</config>
</configs>
```

功能说明

`aas-config.jar`提供以下主要功能：

- **发布配置**：允许用户通过API向配置系统提交新的配置项。
- **监听配置**：允许用户注册监听器，当特定的配置项发生变化时接收通知。
- **更新配置**：支持在配置源中更新配置项，同时通知所有相关的监听器。

API文档

`ConfigCenterImp`接口

```
package com.apusic.config.impls;

public interface ConfigCenterImpl {
    public void updateApusicConfigs(String content);
    public boolean publishApusicConfig();
    public void addConfigListener() throws Exception;
}
```

方法

- `updateApusicConfigs(String content)`

  根据内容修改配置

  参数:

  - `content` (str): 新的配置

- `publishApusicConfig()`

  发布总配置文件中的配置

- `addConfigListener()`

  监听配置中心的配置

ConfigCenter实现类

```
public class ConfigCenter implements ConfigCenterImplements {
    private static final Logger logger =
Logger.getLogger(ConfigCenter.class.getName());
    private final String configCenterAddr;
    private final String dataId;
    private final String group;
    private final long timeoutMs;
    private final String configsFileName;
    private final Listener listener;

    // Static inner class for thread-safe singleton initialization
    private static class Holder {
        private static final ConfigCenter INSTANCE = new ConfigCenter();
    }

    public static ConfigCenter getInstance() {
        return Holder.INSTANCE;
    }
    private String getConfigValue(String envKey, String sysPropKey, String
defaultValue) {
        String value = System.getenv(envKey);
        return value != null ? value : System.getProperty(sysPropKey,
defaultValue);
    }
```

```
private ConfigCenter() {
        this.configCenterAddr = getConfigValue("CONFIG_CENTER_ADDR",
"serverAddr", "127.0.0.1:8848");
        this.dataId = getConfigValue("DATA_ID",
"com.apusic.config.ConfigCenter.dataId", "configs.xml");
        this.group = getConfigValue("GROUP",
"com.apusic.config.ConfigCenter.group", "DEFAULT_GROUP");
        this.timeoutMs = Long.parseLong(getConfigValue("TIMEOUT_MS",
"com.apusic.config.ConfigCenter.timeoutMs", "3000"));
        this.configsFileName = getConfigValue("CONFIGS_FILE_NAME",
"com.apusic.config.ConfigCenter.configsFileName", "configs.xml");
        this.listener = new Listener() {
            @Override
            public void receiveConfigInfo(String configInfo) {
                updateApusicConfigs(configInfo);
            }

            @Override
            public Executor getExecutor() {
                return null; // Optionally provide an executor for asynchronous
processing
            }
        };
        logger.info("ConfigCenter client has been initialized.");
    }
    /**
     * 更新配置
     *
     * @param content
     */
    public void updateApusicConfigs(String content) {
        ConfigCenterUtils.saveConfigs(content, configsFileName);
    }
    public boolean publishApusicConfig() {
        return publishApusicConfig(dataId, group);
    }

    /**
     * 发布AAMS配置
     *
     * @param dataId 配置ID
     * @param group  配置所在组
     * @return
     * @throws Exception
     */

    public boolean publishApusicConfig(String dataId, String group) {
        try {
            Properties properties = new Properties();
            properties.put("serverAddr", configCenterAddr);
            ConfigService configService =
NacosFactory.createConfigService(properties);
            String content = ConfigCenterUtils.getXMLConfig(configsFileName);
            if (content == null) {
                logger.warning("The content to be published is null.");
                return false;
```

```
            }
            return configService.publishConfig(dataId, group, content);
        } catch (Exception e) {
            logger.severe("Failed to publish configuration to ConfigCenter: " +
e.getMessage());
            return false;
        }
    }
    public void addConfigListener() throws Exception {
        addConfigListener(dataId, group, listener);
    }

    private void addConfigListener(String dataId, String group, Listener
listener) throws Exception {
        Properties properties = new Properties();
        properties.put("serverAddr", configCenterAddr);
        ConfigService configService =
NacosFactory.createConfigService(properties);
        String content = configService.getConfig(dataId, group, timeoutMs);
        logger.info("Listening configuration......");
        configService.addListener(dataId, group, listener);
    }
}
```

ConfigCenterUtils类

```
public class ConfigCenterUtils {
    private static final Logger logger =
Logger.getLogger(ConfigCenterUtils.class.getName());

    /**
     * xml文件内容转为字符串
     *
     * @param doc
     * @return
     */
    public static String convertXMLDocumentToString(Document doc) {
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
            Transformer transformer = tf.newTransformer();
            transformer.setOutputProperty(OutputKeys.METHOD, "xml");
            transformer.setOutputProperty(OutputKeys.INDENT, "no");
            transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
            transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION,
"yes");

            try (StringWriter writer = new StringWriter()) {
                transformer.transform(new DOMSource(doc), new
StreamResult(writer));
                String output = writer.toString();
                if (!output.startsWith("<?xml")) {
                    String xmlDeclaration = "<?xml version=\"1.0\"
encoding=\"UTF-8\"?>\n";
                    output = xmlDeclaration + output;
```

```
                }
                return output;
            }
        } catch (Exception e) {
            logger.warning("The xml file failed to convert the string !");
            return null;
        }
    }
    /**
     * 从apusicConfig标签中获取配置文件名
     *
     * @param config
     * @return
     */
    public static String getSaveConfigFile(String config) {
        // 正则表达式匹配config标签的name属性
        Pattern pattern = Pattern.compile("config\\s+name=\"([^\"]+)\"");
        // 创建一个Matcher对象
        Matcher matcher = pattern.matcher(config);
        String fileName = null;
        // 遍历所有匹配项
        while (matcher.find()) {
            // 获取匹配到的文件名
            fileName = matcher.group(1);
        }
        return fileName;
    }

    public static String getFilePath(String root, String name) {
        if (root == null) {
            root = System.getProperty("user.dir");
        }
        if (File.separatorChar != '/') {
            name = name.replace('/', File.separatorChar);
            root = root.replace('/', File.separatorChar);
        }
        String path = System.getProperty("apusic.home");

        if (path == null) {
            throw new RuntimeException("Did not set system property
'apusic.home'");
        }

        return path + File.separatorChar + root + File.separatorChar + name;
    }
    /**
     * 分离每个apusicConfig标签中的配置
     *
     * @param config
     */
    public static void saveConfigs(String config, String configsFileName) {
        // 更新configs.xml
        writeXMLFile(getFilePath("conf", configsFileName), config);
        // 正则表达式匹配<config >标签的内容
        Pattern pattern = Pattern.compile("<config [^>]*>(.*?)</config>",
Pattern.DOTALL);
```

```
        Matcher matcher = pattern.matcher(config);

        while (matcher.find()) {
            String apusicConfigItem = matcher.group();
            String fileName = getSaveConfigFile(apusicConfigItem);
            String configItem = matcher.group(1);

            if ("apusic.conf".equals(fileName)) {
                saveApusicConfig(apusicConfigItem, getFilePath("conf",
fileName));
                continue;
            }
            // 如果不是properties结尾，则添加XML声明
            if (!fileName.endsWith("properties")) {
                configItem = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
configItem;
                writeXMLFile(getFilePath("conf", fileName), configItem);
            }
            writeXMLFile(getFilePath("conf", fileName), configItem);
            logger.info("File is saved in " + getFilePath("conf", fileName));
        }
    }
    /**
     * 从xml文件中获取配置信息（字符串形式）
     *
     * @return
     */
    public static String getXMLConfig(String path) {
        // 获取configs.xml的文件路径
        String filePath = ConfigCenterUtils.getFilePath("conf", path);
        // 判断文件是否存在
        if (!Files.exists(Paths.get(filePath))) {
            logger.warning("The file " + filePath + " does not exist.");
            return null;
        }

        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        // 禁用外部实体 - 防御XXE攻击
        factory.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
        factory.setAttribute(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");

        try (InputStream is = Files.newInputStream(Paths.get(filePath))) {
            DocumentBuilder builder = factory.newDocumentBuilder();
            // 解析 XML 文件获取 Document 对象
            Document document = builder.parse(is);
            // 标准化 XML 结构
            document.getDocumentElement().normalize();
            return ConfigCenterUtils.convertXMLDocumentToString(document);
        } catch (Exception e) {
            logger.warning("Failed to parse the XML file: " + e.getMessage());
        }

        return null;
    }
    /**
     * 存储配置到相应路径
```

```
     *
     * @param filePath
     * @param content
     */
    private static void writeXMLFile(String filePath, String content) {
        File file = new File(filePath);
        try (FileWriter writer = new FileWriter(file)) {
            writer.write(content);
            writer.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * 处理apysic.conf文件的保存
     *
     * @param apusicConfig
     * @param filePath
     */
    private static void saveApusicConfig(String apusicConfig, String filePath) {
        try {
            // 创建 DocumentBuilderFactory
            DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();

            // 解析 XML 字符串
            Document document = builder.parse(new
ByteArrayInputStream(apusicConfig.getBytes("UTF-8")));
            document.getDocumentElement().normalize(); // 标准化文档结构

            // 移除所有 'config' 元素中的 'name' 属性
            NodeList configList = document.getElementsByTagName("config");
            for (int i = 0; i < configList.getLength(); i++) {
                ((Element) configList.item(i)).removeAttribute("name");
            }

            // 创建新的 Document 来保存需要的 XML 内容
            Document newDocument = builder.newDocument();
            // 创建 Transformer 一次，重用
            Transformer transformer =
TransformerFactory.newInstance().newTransformer();
            transformer.setOutputProperty(OutputKeys.INDENT, "no");
            transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
            transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION,
"yes");

            // 手动创建不包含 standalone 的 XML 声明
            String xmlDeclaration = "<?xml version=\"1.0\" encoding=\"UTF-8\"?
>\n";
            StringBuilder xmlContentBuilder = new StringBuilder(xmlDeclaration);

            // 将处理过的节点写入新的 Document 并转换为字符串
            for (int i = 0; i < configList.getLength(); i++) {
                Node importedConfig = newDocument.importNode(configList.item(i),
true);
```

```
                newDocument.appendChild(importedConfig);

                StringWriter writer = new StringWriter();
                try {
                    transformer.transform(new DOMSource(newDocument), new
StreamResult(writer));
                    xmlContentBuilder.append(writer.toString());
                    // 清理当前的文档以供下一个循环使用
                    newDocument.removeChild(importedConfig);
                } finally {
                    writer.close(); // 确保资源被关闭
                }
            }
            // 将最终的 XML 内容写入到文件
            try (FileWriter fileWriter = new FileWriter(new File(filePath))) {
                fileWriter.write(xmlContentBuilder.toString());
            }
            logger.info("File is saved in " + filePath);

        } catch (Exception e) {
            logger.warning("An error occurred while saving the Apusic config.");
        }
    }
}
```

版本历史

- **1.0.0** - 初始发布。提供基本基本配置发布、配置监听以及配置修改功能