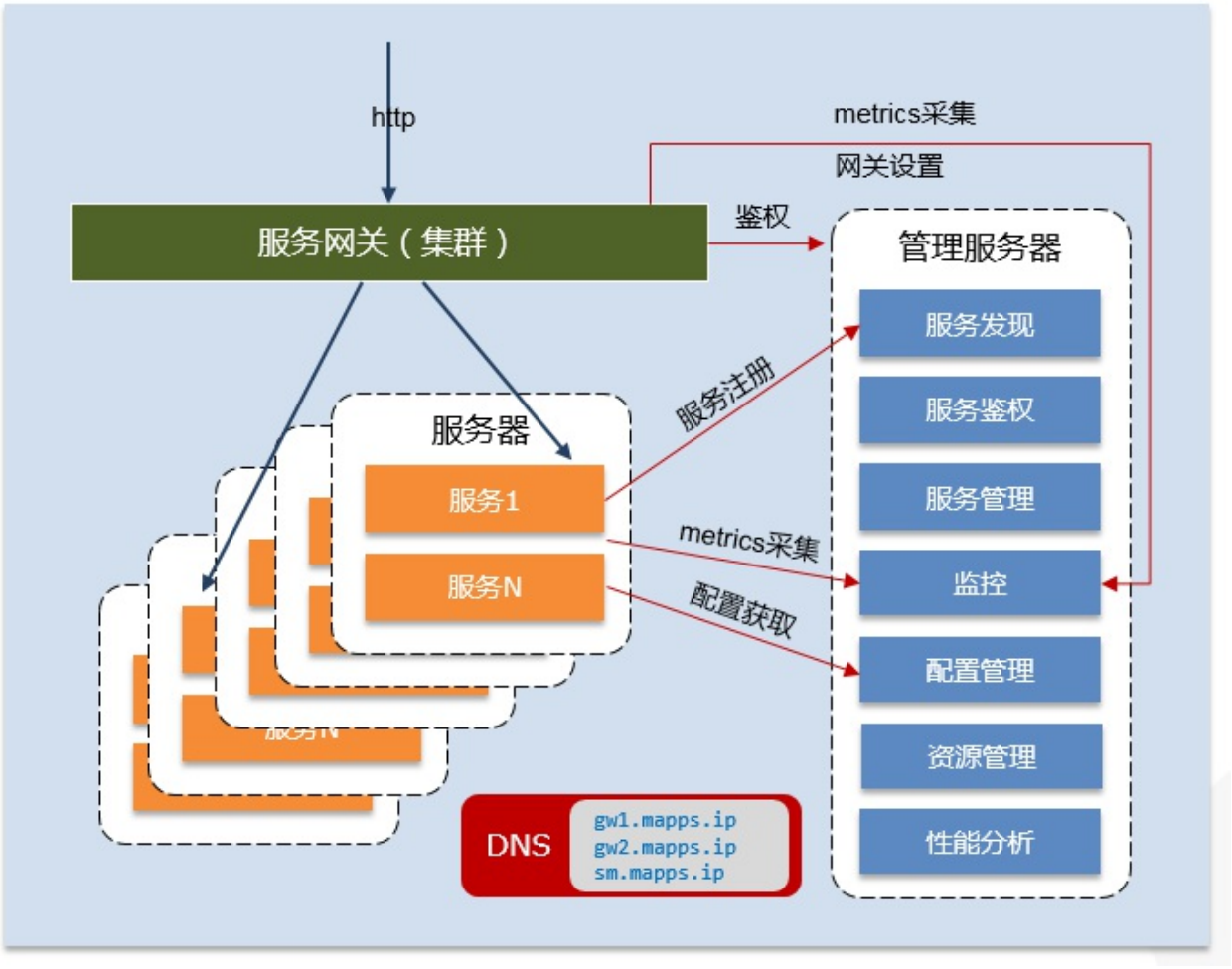
# Table of Contents

# **mplus**微服务应用集成开发指南

本文用来说明如何基于Spring Boot或者Spring MVC开发和集成在mplus轻应用的环境中。



# 名称解释

- 应用：可供最终用户使用的一系列功能的集合，和mplus 集成 ，调用自身及其他服务，依赖系统资源；
- 服务：特殊类应用，提供能力供其他应用使用；
- 资源：系统级别的服务，诸如数据库、redis等；
- 微服务：无状态的应用模块，实现单一功能的微小化的应&用服务；
- 服务注册：微服务管理的一个机制，每个应用&服务注册到管理中心，实现服务发现以及管理；
- 服务依赖：应用&服务对所使用的服务的声明，对于需要授权的服务，需经管理中心绑定授权才可以访问对应的服务；
- 资源分配：将管理中心管理的资源分配给相应的应用&服务；
- 服务绑定：依赖的服务分配及授权的过程；
- 配置管理：对于应用自身的配置项，可以通过服务注册声明，由管理中心统一配置、管理；

# 1 maven依赖管理

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.fiberhome.mapps</groupId>
            <artifactId>mapps-parent</artifactId>
            <version>1.1.0-SNAPSHOT</version>
            <scope>import</scope>
            <type>pom</type>
        </dependency>
    </dependencies>
</dependencyManagement>
```

# 2 依赖jar包

```
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-eureka</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>com.fiberhome.mapps</groupId>
            <artifactId>mapps-msclientsdk</artifactId>
            <version>1.0.0-SNAPSHOT</version>

        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
        </dependency>
```

# 3 Application集成

```
在Application类声明中添加以下Annotation.
@EnableEurekaClient
```

```
@Configuration
@ComponentScan
@EnableAutoConfiguration()
@EnableEurekaClient
@SpringBootApplication
@ImportResource("classpath:applicationContext-openapi.xml")
@Import({FdfsClientConfig.class})
@RestController
public class FileServiceApplication {

    public static void main(String[] args) {
        ConfigClientConfiguration ccc = new ConfigClientConfiguration();
        new SpringApplicationBuilder(FileServiceApplication.class).initializers(ccc).
web(true).run(args);
    }

}
```

# 4 需要注意的点

## 4.1 flywaydb集成方式

```
    @Bean(name = "flyway", initMethod = "migrate")
    public Flyway flywayNotADestroyer() {
        Flyway flyway = new Flyway();
        flyway.setDataSource(dataSource);
        flyway.setBaselineOnMigrate(true);
        flyway.setBaselineVersionAsString("0.0.1");
        flyway.setLocations(sqlLocation);

        // 需要修改Schema管理的表名，以区别不同应用/服务
        flyway.setTable("_mr_schema_version");

        return flyway;
    }
```

需要改成

```
    @Bean(name = "flyway")
    public Flyway flywayNotADestroyer() {
        Flyway flyway = new Flyway();
        flyway.setDataSource(dataSource);
        flyway.setBaselineOnMigrate(true);
        flyway.setBaselineVersionAsString("0.0.1");
        flyway.setLocations(sqlLocation);

        // 需要修改Schema管理的表名，以区别不同应用/服务
        flyway.setTable("_mr_schema_version");

        try {
            flyway.migrate();
        } catch (Exception ex) {
            ex.printStackTrace();
        }

        return flyway;
    }
```

## 4.2 参数注入

为了避免服务刚注册没有配置参数导致的参数找不到的配置错误，可采用${x.y.z:defaultValue}的形式，如

```
flywaydb:
  locations: ${sqllocaltion.${resources.database.default.type:postgresql}}
```

## 4.3 扫描包范围剔除msclientsdk

剔除msclientsdk对应的包com.fiberhome.mapps.mssdk,防止启动时被扫描到导致自动装在从而使项目启动失败

```
<context:component-scan base-package="com.fiberhome" >
        <context:exclude-filter type="regex" expression="com.fiberhome.mapps.mssdk.*"
/>
</context:component-scan>
```

## 4.4 其他

要确保配置错误时应用不自动退出。

目前集成仅支持**maven**管理的项目，建议**Spring**的版本为**4.2.6+**。

## **1**添加集成相关依赖包

```
        <dependency>
            <groupId>com.fiberhome.mapps</groupId>
            <artifactId>mapps-msclientsdk</artifactId>
            <version>1.1.0-SNAPSHOT</version>
            <exclusions>
                <exlusion>
                    <artifactId>validation-api</artifactId>
                    <groupId>javax.validation</groupId>
                </exlusion>
            </exclusions>
        </dependency>
```

## **2** 向**web.xml**中添加

```
<context-param>
    <param-name>contextInitializerClasses</param-name>
    <param-value>com.fiberhome.mapps.mssdk.ConfigClientConfiguration</param-value>
</context-param>
```

完整web.xml示例:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_3_0.xsd"
    id="WebApp_ID">
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>


<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

<!-- 微服务集成 begin-->
<context-param>
    <param-name>contextInitializerClasses</param-name>
    <param-value>com.fiberhome.mapps.mssdk.ConfigClientConfiguration</param-value>
</context-param>
<!-- 微服务集成 end-->

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-c
lass>
</listener>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-servlet.xml,classpath*:**/applicationContext*.xml</p
aram-value>
</context-param>
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
</welcome-file-list>

</web-app>
```

## 3 修改Spring Component扫描配置

在Spring的配置文件中，向扫描**context:component-scan**中添加**com.fiberhome.mapps.mssdk**,以逗号隔开，例子如下:

```
    <!--① 扫描Spring Bean-->
    <context:component-scan base-package="com.fiberhome.mapps.mssdk, com.springdemo"
/>
```

若不存在,则在src/main/resources(classpath)中添加applicationContext.xml文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:rop="http://www.bookegou.com/schema/rop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
       http://www.bookegou.com/schema/rop http://www.bookegou.com/schema/rop/rop-1.0.
xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.1.xsd">


    <!--① 扫描Spring Bean-->
    <context:component-scan base-package="com.fiberhome.mapps.mssdk" />



</beans>
```

# 4 其他需要注意的点

要确保配置错误时应用不自动退出。

# 1 服务注册元数据

```
servicemanager:
  endpoint: http://sm.mapps.ip:8761

eureka:
  client:
    enabled: true
    serviceUrl:
      defaultZone: ${servicemanager.endpoint}/eureka
  instance:
    instanceId: ${spring.application.name}:${server.ipAddress}:${server.port}
    metadataMap:
      appName: testspringmvc                                    # 应用名称
      logo: /logo.png                                           # 应用logo
      remarks: 描述                                             # 应用描述
      portal: /websso                                           # 自服务门户集成入口
      mgr-port: ${management.port}                              # 管理端口
      mgr-context-path: ${management.context-path}              # 管理endpoint的context
 path
      configProperties:
      - key: mplus.sso.serviceUrl
        name: Mplus第三方接入地址
        type: text
        remark: Mplus可供访问的第三方接入地址,格式:http(s)://ip:port/thirdpartaccess
        size: 255
        options: null
        regex: http(s)?://([\w-]+\.)+[\w-]+(:[0-9]+)?(/[\w- ./?%&=]*)?
        group: 通用设置
        default: http://192.168.160.98:6001/thirdpartaccess
      - key: mplus.sso.appKey
        name: Mplus第三方接入appKey
        type: text
        remark: Mplus第三方接入的appKey
        size: 50
        options: null
        regex: \w+
        group: 通用设置
        default: mr
      - key: mplus.sso.secret
        name: Mplus第三方接入密钥
        type: text
        remark: Mplus第三方接入的密钥
        size: 50
        options: null
        regex: \w+
        group: 通用设置
        default: FHuma025
```

```
      - key: mplus.login.serviceUrl
        name: Mplus客户端登录地址
        type: text
        remark: Mplus可供访问的客户端登录地址,格式:http(s)://ip:port/clientaccess
        size: 255
        options: null
        regex: http(s)?://([\w-]+\.)+[\w-]+(:[0-9]+)?(/[\w- ./?%&=]*)?
        group: 通用设置
        default: http://192.168.160.98:6001/clientaccess
      - key: mplus.login.loginPage
        name: Mplus自服务门户登录页面
        type: text
        remark: Mplus自服务门户的登录页面地址,格式:http(s)://ip:port/m
        size: 255
        options: null
        regex: http(s)?://([\w-]+\.)+[\w-]+(:[0-9]+)?(/[\w- ./?%&=]*)?
        group: 通用设置
        default: https://192.168.160.98:8443/m
      - key: mplus.mos.serviceUrl
        name: mos接口地址
        type: text
        remark: mos接口地址,格式:http://ip:port/mos/api
        size: 255
        options: null
        regex: http(s)?://([\w-]+\.)+[\w-]+(:[0-9]+)?(/[\w- ./?%&=]*)?
        group: 通用设置
        default: http://192.168.160.72:7777/mos/api
      dependencies:                                          # 依赖声明
        services: mapps-fileservice                          # 依赖的服务，id，逗号分隔
        resources:                                           # 依赖的资源
          database: default                                  # resId: resCode(资源id
:  资源代码)
          redis: default

spring:
  application:
    name: mapps-testspringmvc

management:
  context-path: /__mng__
  port: 38010

server:
  port: 8080

endpoints:
  restart:
    enabled: true
```

注意：空格的保留，**yml**格式对空格有严格的要求

## 1.1 服务注册客户端：client

该部分配置默认不需要修改。

## 1.2 实例信息：instance

instanceId: 实例id，默认不修改

## 1.3 元数据：metatdataMap

### 1.3.1 基本信息

```
appName: 文件服务                                  # 应用名称
logo: /images/logo.png                             # 应用logo
remarks: 描述                                      # 应用描述
portal: /websso                                    # 自服务门户集成入口
mgr-port: ${management.port}                           # 管理端口
mgr-context-path: ${management.context-path}  # 管理endpoint的context path
```

### 1.3.2 配置参数：configProperties

形式为：参数key: {属性定义 }，以fileservice.store参数为例

```
- key: fileservce.store                          # 属性key
  name: 存储类型                                  # 属性名称
  type: radio                                    # 类型，text,radio,checkbox
  remark: 选择文件系统或是fdfs存储                 # 说明+单位(如果该key需要填值)+格式
  size: 100                                      # 值列长度
  options: file:文件系统,fdfs:FastDFS             # radio或者checkbox时的选项，可为空
  regex: "(file)|(store)"                        # 校验正则表达式
  group: 通用设置                                 # 参数分组名
  default: file                                  # 缺省值
```

通过以上参数的声明，在服务管理中心，进行统一配置；在应用重启后，通过配置客户端获取参数，就上文例子，fileservice.store作为参数名被引用。

注意:说明字段请填写：说明+单位\(如果该key需要填值\)+格式

例子1(说明中添加单位):

```yaml
- key: book.orderNum                                  # 属性key
  name: 书本预定数量                                  # 属性名称
  type: input                                         # 类型，text,radio,checkbox
  remark: 书本预定数量,单位(本)                       # 说明+单位(如果该key需要填值)+格式
  size: 100                                           # 值列长度
  options: null                                       # radio或者checkbox时的选项，可为空
  regex: \d+                                          # 校验正则表达式
  group: 通用设置                                     # 参数分组名
  default: 0                                          # 缺省值
```

例子2(说明中添加格式):

```yaml
- key: book.submitUrl                                                    # 属性key
  name: 书本服务器地址                                                   # 属性名称
  type: input                                                            # 类型，text,radio,ch
eckbox
  remark: 提交书本预定信息的服务器地址,格式:http(s)://ip:port/path  # 说明+单位(如果该key需要
填值)+格式
  size: 100                                                              # 值列长度
  options: null                                                          # radio或者checkbox时
的选项，可为空
  regex: http(s)?://([\w-]+\.)+[\w-]+(:[0-9]+)?(/[\w- ./?%&=]*)? # 校验正则表达式
  group: 通用设置                                                        # 参数分组名
  default: 0                                                             # 缺省值
```

例子3(说明中添加范围):

```yaml
- key: book.reqInvalidateTime                                            # 属性key
  name: 请求超时时间                                                     # 属性名称
  type: input                                                            # 类型，text,radio,c
heckbox
  remark: 提交书本预定信息的请求超时时间,单位：分钟,范围：1~999,默认值：30  # 说明+单位(如果该k
ey需要填值)+范围+格式
  size: 100                                                              # 值列长度
  options: null                                                          # radio或者checkbox
时的选项，可为空
  regex: "^([1-9][0-9]{0,2})$"                                           # 校验正则表达式
  group: 通用设置                                                        # 参数分组名
  default: 30
```

例子4(说明中添加选项):

```
- key: book.isSelfOwn                                    # 属性key
  name: 私有书本                                          # 属性名称
  type: radio                                            # 类型，text,radio,ch
eckbox
  remark: 是否为私有书本,1：是 (默认) 0：不是              # 说明+单位(如果该key需
要填值)+格式
  size: 100                                              # 值列长度
  options: null                                          # radio或者checkbox时
的选项，可为空
  regex: "(1)|(0)"                                       # 校验正则表达式
  group: 通用设置                                         # 参数分组名
  default: 0
```

### 1.3.3 依赖声明：dependencies

services，服务依赖，逗号分隔的服务所在的应用id，在应用中通过services.${serviceId}.serviceUrl获取服务的访问地址；services.${serviceId}.appkey和services.${serviceId}.secret获取访问密钥。

resources，声明所有依赖的系统资源，资源id：资源代码，在应用中通过resources.${resId}.${resCode}.key获取资源的参数；

目前提供数据库资源的配置，其配置参数如下：

| 参数**key** | 说明 | 示例 |
|---|---|---|
| type | 数据库类型，oracle或者postgresql | postgresql |
| host | 主机 | 192.168.160.103 |
| port | 端口 | 5432 |
| db | 数据库，oracle为sid | mr_test |
| username | 用户名 | mr |
| password | 密码 | 123456 |

redis资源的配置，其配置参数如下：

| 参数**key** | 说明 | 示例 |
|---|---|---|
| dbIndex | redis数据库索引 | 0 |
| host | 主机 | 192.168.160.45 |
| port | 端口 | 6379 |
| password | 密码 | (空) |

# 2管理功能集成

```yaml
management:
  context-path: /__mng__
  port: 38010

endpoints:
  restart:
    enabled: true
```

# 1 数据写入器（CustomMetricsWriter）

类名：com.fiberhome.mapps.mssdk.metrics.CustomMetricsWriter

# 2 引用方式：

1） 使用@Autowired，自动注入

2）通过id为“customMetricsWriter”的本注入

# 3 调用方法

方法：

```
/**
 * 提交指标数据
 * @param metric 指标信息，如内存大小
 * @param value 指标值
 */
public void submit(MetricName metric, Double value) {

/**
 * 计数器增加
 * @param metric 计数器指标，如调用次数
 */
public void increment(MetricName metric) {

/**
 * 重置计算器
 * @param metric 计数器指标
 */
public void reset(MetricName metric) {
```

MetricName定义：

```
package com.fiberhome.mapps.mssdk.metrics;

import java.util.HashMap;
import java.util.Map;

import org.springframework.util.StringUtils;

/**
 * 指标信息
 * @author Administrator
 *
```

```
 */
public class MetricName {
    /**
     * 指标代码
     */
    private String measurement;

    /**
     * 指标tag
     */
    private Map<String, String> tags = new HashMap<>();

    public MetricName(String measurement) {
        this(measurement, null);
    }

    public MetricName(String measurement, Map<String, String> tags) {
        this.measurement = measurement;
        if (tags != null) {
            tags.putAll(tags);
        }
    }

    /**
     * 设置tag
     * @param key tag key
     * @param value tag Value
     * @return 当前MetricName
     */
    public MetricName tag(String key, String value) {
        this.tags.put(key, value);
        return this;
    }

    /**
     * 获取指标代码
     * @return
     */
    public String getMeasurement() {
        return measurement;
    }

    /**
     * 设置指标代码
     * @param measurement
     */
    public void setMeasurement(String measurement) {
        this.measurement = measurement;
    }

    /**
```

```
     * 获取所有的tag信息
     * @return
     */
    public Map<String, String> getTags() {
        return tags;
    }

    /**
     * 设置tag信息
     * @param tags
     */
    public void setTags(Map<String, String> tags) {
        this.tags = tags;
    }

    /**
     * 获取通过指标代码和tag拼接的id
     * @return
     */
    public String getId() {
        return measurement + "," + StringUtils.collectionToCommaDelimitedString(tags.
values());
    }

}
```

# 4 例子

## 4.1 计数器

比如，写入rop方法的调用计数：

```
MetricName meter = new MetricName("rop.api.meter");
meter.tag("method", method);
meter.tag("v", version);
metricsWriter.increment(meter);
```

4.2 指标数值

比如，写入rop方法的调用耗时

```
MetricName timer = new MetricName("rop.api.timer");
timer.tag("method", method);
timer.tag("v", version);

double e = ropEvent.getServiceEndTime() - ropEvent.getServiceBeginTime();
metricsWriter.submit(timer, e);
```

# 1 实现方法

实现接口org.springframework.boot.actuate.health.HealthIndicator

# 2 集成方式

1） 实现类的@Bean方式的annotation声明

```
@Bean
public HealthIndicator gateHealthIndicator() {
    return new GatewayHealthIndicator();
}
```

2） 实现类的<bean 定义>

```
<bean id="gatewayHealthIndicator" class="com.fiberhome.mapps.gateway.health.GatewayHealthIndicator"/>
```

# 3 代码实现

在HealthIndciator.health()方法中返回Health对象，对象中设置状态为：
org.springframework.boot.actuate.health.Status.up.build()或者
org.springframework.boot.actuate.health.Status.down.build();

```
    @Override
    public Health health() {
        Health health = null;
        long current = System.currentTimeMillis();

        // 缓存一分钟，超过一分钟进行自检
        if (cache == null || current - lastHealthRenewTime > 60000 || cache.getStatus
() == Status.DOWN) {

            try {
                LOG.debug("Custom HealthIndicator working...");
                String result = restTemplate.postForObject(checkUrl, null, String.cla
ss);
                lastHealthRenewTime = current;

                if ("SUCCESS".equals(result)) {
                    health = Health.up().build();
                }
            } catch (RestClientException e) {
                LOG.error("Health check error", e);
            }
            if (health == null) {
                health = Health.down().build();
            }
            cache = health;
        }

        return cache;
    }
```