

## 一体化平台算法开发文档

一体化算法

开发说明使用文档

(版本号) v1.0

## 目录

1.概述.....	4
1.1 产品简介.....	4
(1) 数字服务平台.....	4
(2) 气象智脑系统.....	4
(3) 场景业务应用支撑分系统.....	4
1.2 平台架构.....	5
2.引用文档.....	5
2.1 应用文档.....	5
2.2 参考文档.....	5
3.产品介绍.....	5
3.1 开发 CLI 工具.....	6
3.2 运行环境要求.....	7
3.2.1 软件环境要求.....	7
3.2.2 硬件环境要求.....	8
3.2.3 其他接入要求.....	8
3.3 docker 环境部署方案.....	8
3.3.1 打包.....	9
3.3.2. 镜像构建.....	9
3.3.3. 部署.....	11
3.4. 服务管理部署方案.....	12
3.4.1. 介绍.....	12
3.4.2. 使用.....	13
3.5. 用户体系介绍.....	20
4.开发规范.....	23
4.1. 命名规范.....	23
4.1.1 包命名.....	23
4.1.2 类命名.....	23
4.1.3 方法命名.....	23

4.1.4 变量命名 .....	24
4.1.5 常量命名 .....	24
4.1.6 构造方法命名 .....	24
4.2. 代码规范 .....	24
4.2.1 缩进和格式 .....	24
4.2.2 编码风格 .....	24
4.2.3 依赖管理 .....	25
4.3. 其他 .....	25
4.3.1 单元测试 .....	25
4.3.2 异常处理 .....	25
5.开发示例 .....	25
5.1. 基础使用 .....	25
5.1.1 项目简介 .....	26
5.1.2 接入配置 .....	27
5.2. 数据库 .....	27
5.3. 接口文档 .....	28
5.4. 基础服务 .....	28
5.5. 扩展融入 .....	28

# 1.概述

## 1.1 产品简介

天气业务一体化包含数字服务平台、气象智脑系统、场景业务应用支撑分系统三大分系统。

天气业务一体化平台框架由开放式应用框架与组件库组成，为场景开发提供基础编码环境，其中开放式应用框架实现天气监测、预报、预警和服务业务流程中的整体框架和核心功能抽象和泛化，组件库提供包含认证、气象数据获取、诊断分析、二三维显示、人机交互、预报预警协同、检验评估等各类具体处理逻辑和交互可视功能组件。开放式应用框架包括标准服务接口定义、基础公用组件库、前端业务流程编排、消息中间件、界面风格定义等功能。

### (1) 数字服务平台

数字服务平台提供立体观测、三维实况分析、地球系统数值模式、导航 AIS、台风路径等基础数据访问，管理智能数字预报、灾害性天气预报预警、台风路径、航路航线等各类数字化预报信息，实现各类数据的集约高效的汇集、处理、管理和服 务，解决数据孤岛和数据不一致问题，为数字预报业务提供数据支撑，形成统一的数据存储，提供标准化的数据服务。

### (2) 气象智脑系统

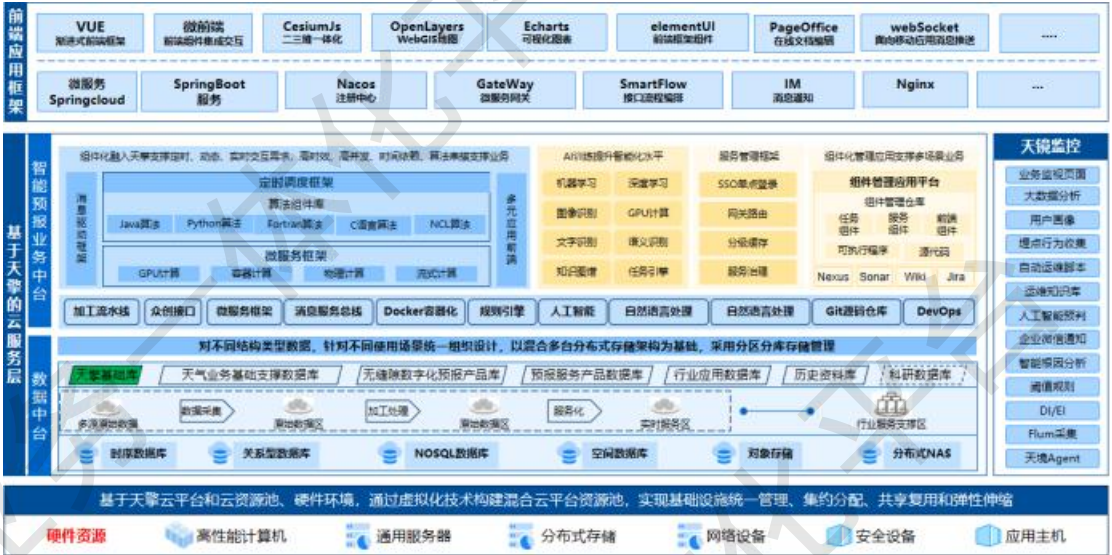
气象智脑系统使用智能数字服务作为基本的数据和服务来源，以实现数字预报员为目标，结合气象客观分析算法与大预报模型产品，利用人工智能与大数据应用技术实现面向智能感知、智能诊断、智能研判、智能生成等服务，为各类前端应用场景提供统一的信息提取与产品加工能力。

### (3) 场景业务应用支撑分系统

通过多场景业务应用支撑框架构建于数字服务平台和气象智脑核心的多场景业务应用位于天气业务一体化平台的顶端，是平台赋能、场景应用的实现层，是一套开放的应用生态体系，能够支持多场景的业务应用和业务扩展。支持海洋数字预报一张网、导航风险预警一张图、综合信息展示一块屏三类业务应用核心业务场景。

1.2 平台架构

天气业务一体化技术架构由组件开发应用框架、天气业务组件库和基础应用模板组成，总体技术架构图如下图所示：



2.引用文档

2.1 应用文档

《天气业务一体化平台设计分册 V4.0.0》

《天气业务一体化平台需求分册 V4.0.0》

2.2 参考文档

- 《总装备部软件工程规范》
- 《ISO9001 质量管理体系标准规范》
- 《气象软件工程规范》，中国气象局.
- 《计算机软件工程规范国家标准汇编 2007》

3.产品介绍

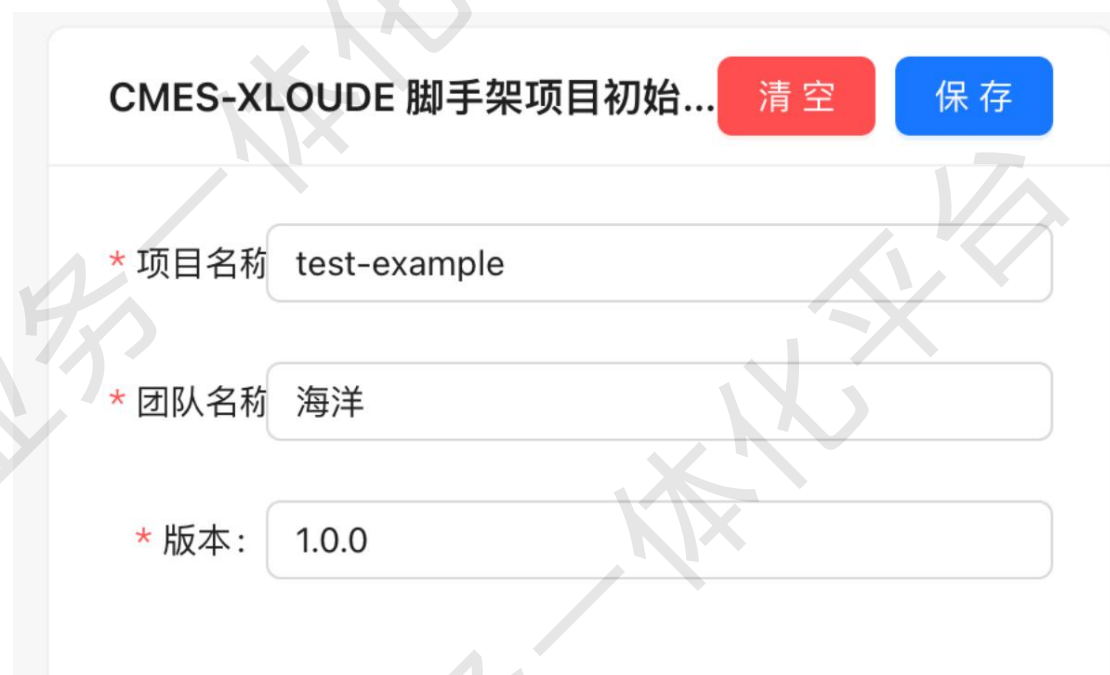
### 3.1 开发 CLI 工具

使用脚手架初始化项目

项目名称：应用名称，建议使用小写字母填写，多个单词之间用横杠连接

团队名称：自己填写就可以，例如远洋导航

版本：版本号，例如 1.0.0



CMES-XLOUDE 脚手架项目初始... 清空 保存

\* 项目名称 test-example

\* 团队名称 海洋

\* 版本: 1.0.0

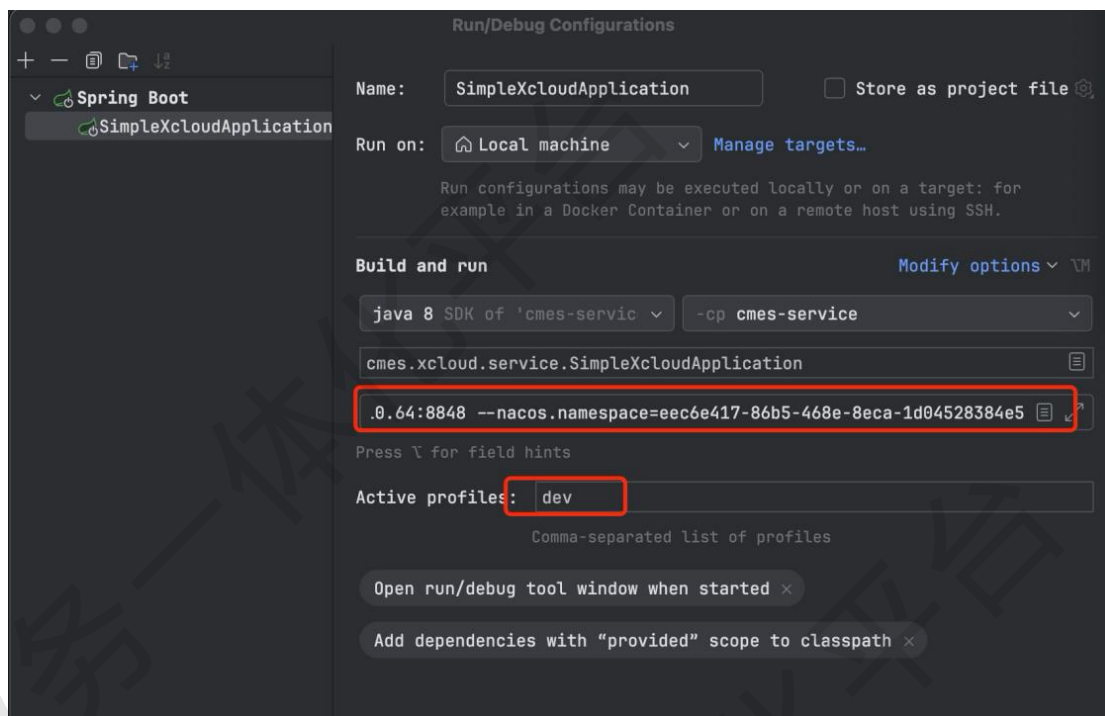
解压项目倒入 IDE

启动项目的时候需要指定运行参数：

--nacos.group=服务分组

--nacos.addr=nacos 地址

--nacos.namespace=命名空间



## 3.2 运行环境要求

### 3.2.1 软件环境要求

为了保证系统个的正常运行，需满足以下软件环境要求。

软件名称	版本	备注
jdk	1.8	
虚谷数据库	11.0.0	
redis	6.0.16	
maven	3.9.8	
docker	26.1.3	
nacos	2.1.2	

xxl-job	2.4.1	
---------	-------	--

### 3.2.2 硬件环境要求

一体化硬件环境参考下图

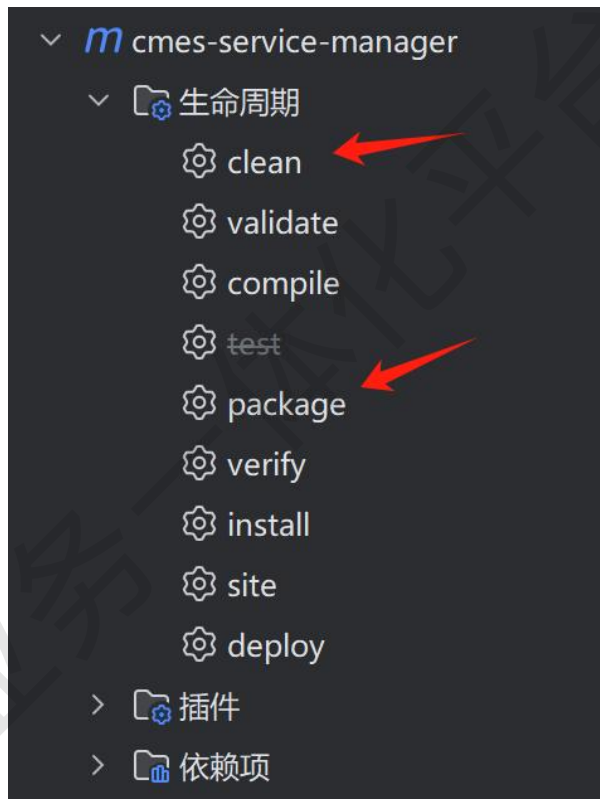
服务器	服务类别			资源				汇总
	服务名称	内容	节点数量	内存(G)	内存合计	CPU	CPU合计	磁盘
平台基础服务	网关服务		3	32	96	4	12	512G 固态*9 + 5T 共享 NAS
	注册中心		3	32	96	4	12	
	认证服务	登录、权限管理、接口认证	3	16	48	4	12	
	系统管理服务	用户角色菜单	2	16	32	4	8	
	系统支撑服务	气象图例、阈值、规则模板	2	16	32	4	8	
	服务管理服务	CICD	2	16	32	4	8	
	基础数据服务	站点、格点等基础数据支撑	3	16	48	4	12	
	热点数据处理服务	数据智能加载	2	32	64	4	8	
	监控管理服务		2	16	32	4	8	
中间件服务器	Redis集群		3	512	1536	64	192	1T 固态*7 + 10T 共享 NAS
	RocketMQ集群		3	64	192	16	48	
	Prometheus		3	32	96	8	24	
	Grafana		3	32	96	8	24	
	Elasticsearch		3	128	384	32	96	
	GitLab		1	64	64	16	16	
	Nexus		1	64	64	16	16	
	Skywalking		3	32	96	8	24	
支撑服务	组件库		2	64	128	16	32	1T存储
	IM		3	128	384	16	48	
	XX-JOB	简单流程处理	3	64	192	16	48	
业务应用服务器	业务界面、逻辑接口等		2	512	1024	64	128	512G固态*6 + 2T 共享 NAS
仿真环境	业务界面、逻辑接口等		1	512	512	64	64	512G固态*6 + 2T 共享 NAS

### 3.2.3 其他接入要求

## 3.3 docker 环境部署方案



### 3.3.1 打包



使用 idea 的 maven 插件本地打包

### 3.3.2. 镜像构建

构建镜像的 Dockerfile 参考

```
1 FROM openjdk:8
2 LABEL maintainer=yuelong \
3     version=1.0.0 \
4     description=cmes-service-manager
5
6 # 设置时区
7 RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && \
8     echo 'Asia/Shanghai' >/etc/timezone
9
10 WORKDIR /app
11
12 # 使用环境变量接收 JVM 参数
13 ARG JAVA_OPTS
14 ENV JAVA_OPTS=${JAVA_OPTS:-"-Xms1024m -Xmx4096m"}
15
16 # 拷贝 jar 包
17 COPY *.jar app.jar
```

```

18
19 EXPOSE $SERVER_PORT
20
21 ENTRYPOINT ["sh", "-c", "java $JAVA_OPS -Djava.security.egd=file:/dev/./urandom -jar
app.jar \
22     --spring.profiles.active=$SPRING_PROFILES_ACTIVE \
23     --nacos.addr=$NACOS_ADDR \
24     --nacos.group=$NACOS_GROUP \
25     --server.port=$SERVER_PORT \
26     --nacos.namespace=$NACOS_NAMESPACE"]

```

### 执行部署的脚本参考

```

1 # docker 容器名称, 例如: cmes-service-manager
2 docker_name=XXXX
3 # 服务版本,例如: v0.0.1
4 version=XXX
5 # nacos 地址,例如: 192.168.110.64:8848
6 nacos_addr=XXXX:xxxx
7 # nacos 命名空间,例如: eec6e417-86b5-468e-8eca-1d04528384e5
8 nacos_namespace=xxxxx
9 # 配置文件
10 profiles_active=dev
11 # 服务端口
12 server_port=8082
13 # JVM 最大内存
14 jvm_max_memory=-Xmx10240m
15 # JVM 最小内存
16 jvm_min_memory=-Xms1024m
17
18
19 RED='\033[31m'
20 GREEN='\033[32m'
21 YELLOW='\033[33m'
22 BLUE='\033[34m'
23 RESET='\033[0m'
24
25 echo -e "${GREEN} ==> 开发构建服务容器${RESET}"
26 echo -e "${RED} ==> 将正在运行的容器停止运行并移除${RESET}"
27
28 # 使用 docker ps -a 来检查容器是否存在, 包括已停止的容器
29 if docker ps -a | grep -q $docker_name; then
30     docker rm -f $docker_name
31 fi
32
33 echo -e "${RED} ==> 将移除" $docker_name "容器的镜像${RESET}"

```

```

34 if [[ "$(docker images -q $docker_name 2> /dev/null)" != "" ]];
35 then
36     docker rmi $(docker images -q $docker_name 2> /dev/null)
37 fi
38
39 echo -e "${GREEN} ==> 开始重新构建 docker 镜像${RESET}"
40 docker build -t $docker_name:$version .
41
42 echo -e "${GREEN} ==> 构建完成启动容器${RESET}"
43 if [[ $# -gt 0 ]] && [[ "$1" == "jk" ]]; then
44     echo -e "${BLUE} ==> 启动带监控的容器${RESET}"
45     docker run -d \
46         -e SPRING_PROFILES_ACTIVE=$profiles_active \
47         -e NACOS_ADDR=$nacos_addr \
48         -e SERVER_PORT=$server_port \
49         -e NACOS_NAMESPACE=$nacos_namespace \
50         -e JAVA_OPS="$jvm_min_memory $jvm_max_memory" \
51         -Djava.rmi.server.hostname=$jk_hostname -Dcom.sun.management.jmxremote
52         -Dcom.sun.management.jmxremote.port=$jk_port
53         -Dcom.sun.management.jmxremote.authenticate=false
54         -Dcom.sun.management.jmxremote.ssl=false" \
55         --name $docker_name \
56         --network host \
57         $docker_name:$version
58 else
59     echo -e "${BLUE} ==> 启动不带监控的容器${RESET}"
60     docker run -d \
61         -e SPRING_PROFILES_ACTIVE=$profiles_active \
62         -e NACOS_ADDR=$nacos_addr \
63         -e SERVER_PORT=$server_port \
64         -e NACOS_NAMESPACE=$nacos_namespace \
65         -e JAVA_OPS="$jvm_min_memory $jvm_max_memory" \
66         --name $docker_name \
67         --network host \
68         $docker_name:$version
69 fi
70 echo -e "${GREEN} ==> 请执行 docker logs -f $docker_name 查看启动情况${RESET}"

```

### 3.3.3. 部署

将打包后的 jar 包、Dockerfile 和部署脚本放在待部署服务器同级目录

```

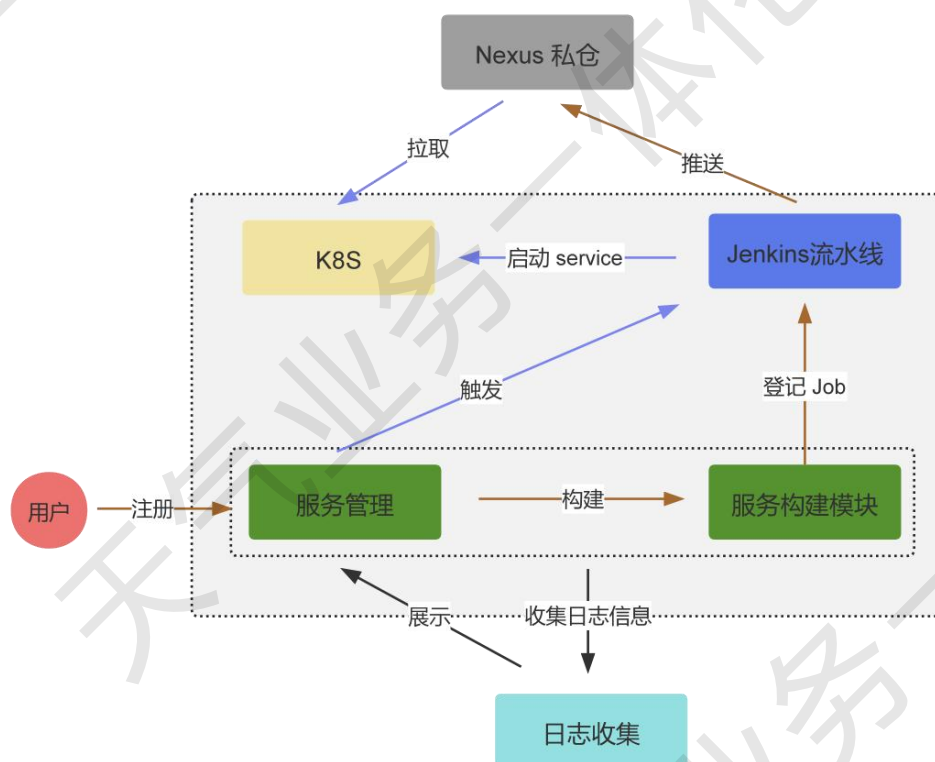
drwxr-xr-x  2 root root      4096 Aug 28 09:57 ./
drwxr-xr-x 40 root root      4096 Sep  4 16:27 ../
-rw-r--r--  1 root root 119269563 Aug 15 15:51 15a4e88423eab063cb170cbdca5d56e8
-rw-r--r--  1 root root      740 May 14 19:03 Dockerfile
-rw-r--r--  1 root root 703178752 Aug 28 09:57 cmes-manager21.tar
-rw-r--r--  1 root root 165341598 Aug 28 16:19 cmes-service-manager.jar
-rw-r--r--  1 root root 106722928 May 16 10:40 cmes-service-manager.jar.bak
-rw-r--r--  1 root root 119269563 Aug 15 15:51 cmes-service-manager.zip
-rwxr-xr-x  1 root root      2486 May 16 16:14 deploy.sh*
-rw-r--r--  1 root root      2000 May 10 18:17 out.log
(base) root@k8s-node1:/home/wxy/cmes-dev/cmes-manager#

```

执行 `bash deploy.sh` 即可将服务部署成 docker 服务

## 3.4. 服务管理部署方案

### 3.4.1. 介绍



方案提供了两种灵活的管理方式，以适应不同项目的需求：一种是允许平台直接管理源代码，另一种则是仅提供构建完成的文件。这两种方式都基于使用脚手架开发的项目：

- **服务元信息登记**: 业务方需在平台上注册服务的基本信息, 涵盖服务名称、版本号、资源配置、发布方式 (如 kubernetes 或 Docker) 、服务脚本等;
- **Jenkins 集成**: 针对可以提供源码的业务服务, 平台将使用 Jenkins 的流水线作为构建方式; 针对无法提供源码, 只能提供构建后的 Jar 等, 需要业务提供编译后的文件和 Dockerfile 文件以及构建和发布脚本信息构建 Jenkins 的 Job, 然后在由平台统一的调度;
- **服务操作**: 此时业务方在平台就可以对服务进行操作, 支持如下操作:
  - 构建: 平台将调用 Jenkins API 或执行构建脚本来打包项目, 并输出构建详情。随后, 利用 Dockerfile 将服务打包成 Docker 镜像
  - 推送: 此时平台会将打包好的镜像, 推送为 docker 的私仓中;
  - 启动: 平台从 docker 私仓中拉取对应的服务镜像信息, 根据配置的服务元信息启动为 docker 容器或者 k8s 中的 service;

### 3.4.2. 使用

具体使用步骤如下:

#### 3.4.2.1 通过 git 源码部署

- 首先要创建凭据 (即可拉取源码的 git 账号密码)



1. 其中凭据 id 要保证唯一, 并且不可以是中文, 可以是如: gitlabtest 或 gitlab-prod 等这样的命名, 凭据 id 不支持修改;
2. 用户名和密码是真实的具有拉取代码权限的 git 账号和密码;
3. 描述根据实际情况填写。

## ● 创建服务元数据

创建服务

×

\* 服务名称

\* 服务ID

\* 应用名称

请选择

▼

\* 服务类型

请选择

▼

\* 镜像名称

\* 版本号

请输入版本号

\* 发布类型

请选择

▼

\* 资源类型

git源码

▼

\* git地址

\* 凭据

请选择

▼

\* 分支

取消

提交

- 源码中要包含可正常打包服务的 Dockerfile 文件（前端项目要放在项目根目录，后端如果是多模块项目放在子模块根目录，否则也放在根目录）

### ○ 脚手架后端 Dockerfile 参考

```

1 FROM openjdk:8
2
3 # 设置时区
4 RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && \
5     echo 'Asia/Shanghai' > /etc/timezone
6
7 WORKDIR /
8
9 # 使用环境变量接收 JVM 参数
10 ARG JAVA_OPS
11 ENV JAVA_OPS=${JAVA_OPS:-"-Xms1024m -Xmx5120m"}
12
13 # copy jar
14 ADD target/*.jar app.jar
15
16
17 EXPOSE ${SERVER_PORT}
18

```

```

19 ENTRYPOINT ["sh", "-c", "java ${JAVA_OPS} \
20     -Djava.security.egd=file:/dev/./urandom -jar app.jar \
21     --spring.profiles.active=${SPRING_PROFILES_ACTIVE} \
22     --nacos.addr=${NACOS_ADDR} \
23     --server.port=${SERVER_PORT} \
24     --nacos.group=${NACOS_GROUP} \
25     --nacos.namespace=${NACOS_NAMESPACE}"]

```

### 前端 Dockerfile 参考

```

1 FROM nginx:alpine3.19
2
3 # 设置时区
4 RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime && \
5     echo 'Asia/Shanghai' >/etc/timezone
6
7 COPY dist /usr/share/nginx/html/
8 COPY nginx/nginx.conf /etc/nginx/conf.d/cmcs.conf
9
10 EXPOSE 80

```

前端项目要在项目根目录中包含 nginx/nginx.conf

nginx 配置参考：（其中

http://gateway-headless.application-fyqx-cmes.svc.cluster.local:888/是网关地址)

```

1 server {
2     listen 80;
3     server_name localhost;
4     location /api/ {
5         proxy_pass http://gateway-headless.application-fyqx-cmes.svc.cluster.local:888/;
6         add_header Access-Control-Allow-Origin *;
7         proxy_connect_timeout 2000;
8         proxy_set_header Host $host;
9         proxy_set_header X-Real-IP $remote_addr;
10        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
11        proxy_http_version 1.1;
12        proxy_set_header Connection "";
13    }
14    location / {
15        root /usr/share/nginx/html;
16        index index.html index.htm;
17        try_files $uri $uri/ /index.html;
18    }

```

```

19     error_page    500 502 503 504    /50x.html;
20     location = /50x.html {
21         root    html;
22     }
23 }

```

#### 3.4.2.1.1 脚手架后端项目部署 k8s 服务

1. 服务名称: 服务名称要和 nacos 注册名一致 (在脚手架启动类配置的名字), 例如: cmes-auth, 作为服务的唯一标识, 创建时会同步为 jenkins 的构建任务名。
2. 服务类型: 后端服务。
3. 镜像名称: 程序打包后发送到私仓的镜像名, 一般和服务名称一致即可。
4. 版本号: 格式为: v0.0.1
5. 发布类型: k8s
6. 资源类型: git 源码
7. git 地址: 项目的 git 仓库地址
8. 凭据 id: 选择上面创建好的 git 凭据 (即拉取 git 代码的账号密码)
9. 分支: 如分支是 dev, 这里填写\*/dev

#### 3.4.2.1.2 通过 git 源码部署前端项目部署 k8s 服务

10. 服务名称: 前端项目名, 作为唯一标识, 创建时会同步为 jenkins 的构建任务名。
11. 服务类型: 前端服务。
12. 镜像名称: 程序打包后发送到私仓的镜像名, 一般和服务名称一致即可。
13. 版本号: 格式为: v0.0.1
14. 发布类型: k8s
15. 资源类型: git 源码
16. git 地址: 项目的 git 仓库地址
17. 凭据 id: 选择上面创建好的 git 凭据 (即拉取 git 代码的账号密码)

#### ● 发布和部署

点击发布, 查询状态, 发布成功后可点击部署





### 3.4.2.2 通过程序包部署 k8s 服务

该功能主要用于集成未发布到 git 源码的项目到 k8s 服务

1. 资源类型：程序包
2. 上传程序包举例：

后端项目：包含 jar 包和 Dockerfile（注意这个 Dockerfile 要在当前目录可打包镜像）

名称	修改日期	类型	大小
cmes-service-manager.jar	2024/8/26 14:47	Executable Jar File	187,304 KB
Dockerfile	2024/8/26 14:26	文件	1 KB

全选右键生成zip文件

前端项目：包含 build 后的 dist 文件夹、nginx/nginx.conf 配置（同 git 源码前端项目）、Dockerfile

dist	2024/9/3 16:49	文件夹	
nginx	2024/9/3 16:49	文件夹	
Dockerfile	2024/8/21 14:14	文件	1 KB

全选右键生成zip包

3. 其他字段配置和 git 源码发布方式一致即可

### 3.4.2.3 上传镜像部署 k8s 服务

主要用于一些特定镜像部署为 k8s 服务

1. 资源类型：Docker 镜像
2. yaml 举例：（以 geoserver 为例）
3. 需要修改的地方为 xxxx 的位置,如 geosever，直接替换所有 “xxxx” 为 “geoserver”

```

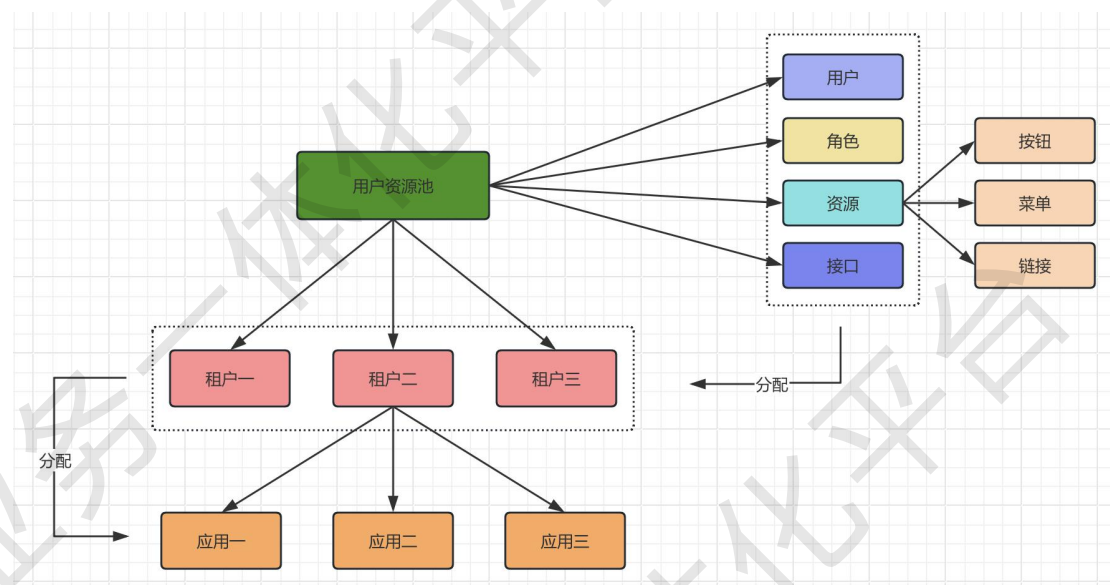
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: xxxx-headless
5    namespace: application-fyqx-cmes
6    labels:
7      app: xxxx-headless
8  spec:
9    type: ClusterIP
10   ports:
11     - port: 8080
12       name: server
13       targetPort: 8080
14   selector:
15     app: xxxx
16 ---
17 apiVersion: apps/v1
18 kind: StatefulSet
19 metadata:
20   name: geoserver
21   namespace: application-fyqx-cmes
22 spec:
23   serviceName: xxxx-headless
24   replicas: 1
25   template:
26     metadata:
27       labels:
28         app: xxxx
29       annotations:
30         pod.alpha.kubernetes.io/initialized: "true"
31     spec:
32       affinity:
33         podAntiAffinity:
34           requiredDuringSchedulingIgnoredDuringExecution:
35             - labelSelector:
36                 matchExpressions:
37                   - key: "fyqx"
38                     operator: In
39                     values:
40                       - cmes
41             topologyKey: "kubernetes.io/hostname"
42     containers:
43       - name: xxxx
44         imagePullPolicy: Always
45         image: 10.194.90.150:9091/xxxx:v1.0

```

```
46     resources:
47       requests:
48         memory: "2Gi"
49         cpu: "500m"
50     ports:
51     - containerPort: 8080
52       name: client
53     env:
54     - name: GEOSERVER_ADMIN_PASSWORD
55       value: "w@ng87631914"
56     - name: GEOSERVER_ADMIN_USER
57       value: "admin"
58     volumeMounts:
59     - name: geoserver-data
60       mountPath: "/opt/geoserver/data_dir"
61     imagePullSecrets:
62     - name: fyqx-cmes
63     nodeSelector:
64       fyqx: cmes
65     volumes:
66     - name: xxxx-data
67       hostPath:
68         path: /usr/local/geoserver_data/
69         type: DirectoryOrCreate
70     selector:
71       matchLabels:
72         app: xxxx
73
74 ---
75 apiVersion: v1
76 kind: Service
77 metadata:
78   name: xxxx-ui
79   namespace: application-fyqx-cmes
80 spec:
81   type: NodePort
82   selector:
83     app: xxxx
84   ports:
85   - name: http
86     port: 8080
87     targetPort: 8080
88     nodePort: 28849
```

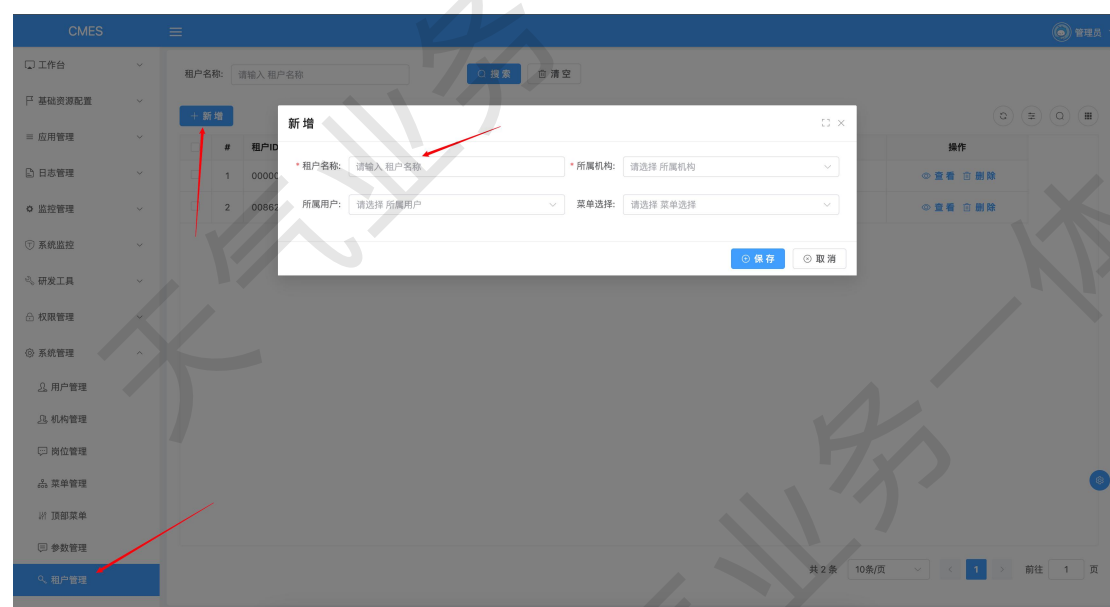
### 3.5. 用户体系介绍

一体化平台是一个多租户多应用的用户体系，平台负责整体的用户数据管理，各个业务单位对应各个租户，业务单位的应用分别有各自的租户管理。



#### 3.5.1. 创建租户

使用平台管理员账号登录，选择系统管理 > 租户管理，点击新增创建租户。



- 租户名称：租户的名称；
- 所属机构：选择机构归属哪些机构，选择机构之后，机构下属的用户都将授权给这个租户；

- 所属用户：这里选择的用户，将被授权给租户管理员，后续需要使用这个用户进行登录，并进行租户的初始化和角色分配；
- 菜单选择：选择租户可以授予的菜单（这里建议选择好对应租户的菜单按钮，敏感菜单建议不授权给租户）；

### 3.5.2. 租户设置

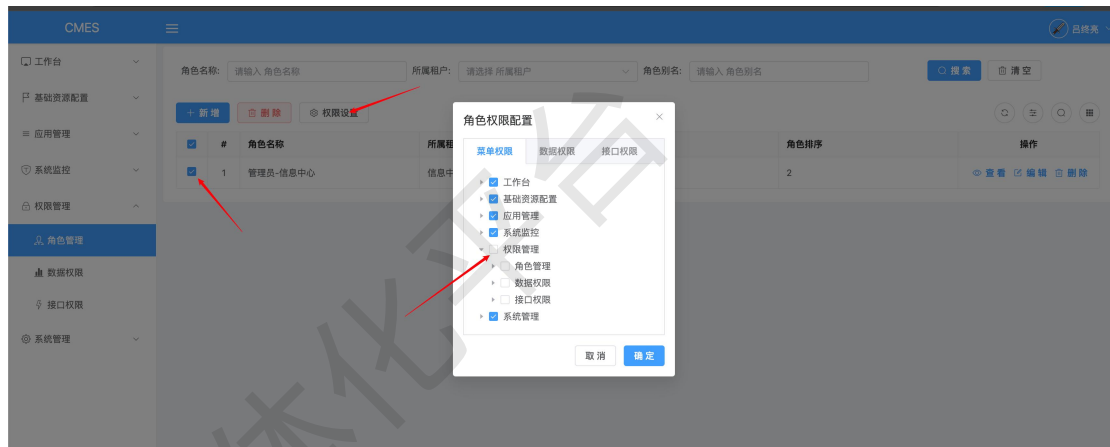
创建之后，使用租户登录进行租户的界面，建议先去配置角色管理，之前已经初始化了一个系统管理员，这里我们还需要创建一个业务角色和普通角色（建议）。



这里输入角色名称、角色别名、上级角色等信息。



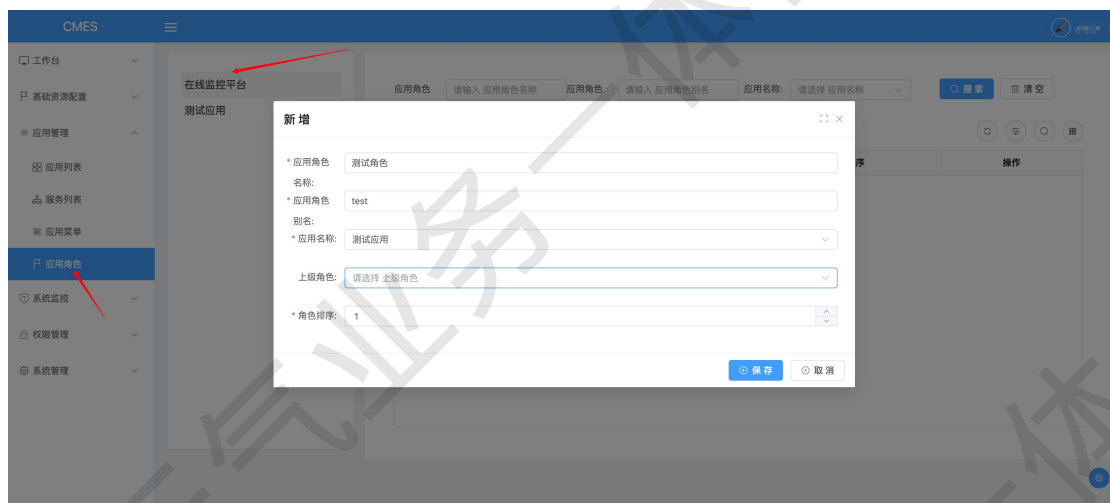
创建好角色之后，选择角色，点击权限设置，配置菜单权限，可以给各个角色设置不同的权限。



### 3.5.3. 应用管理

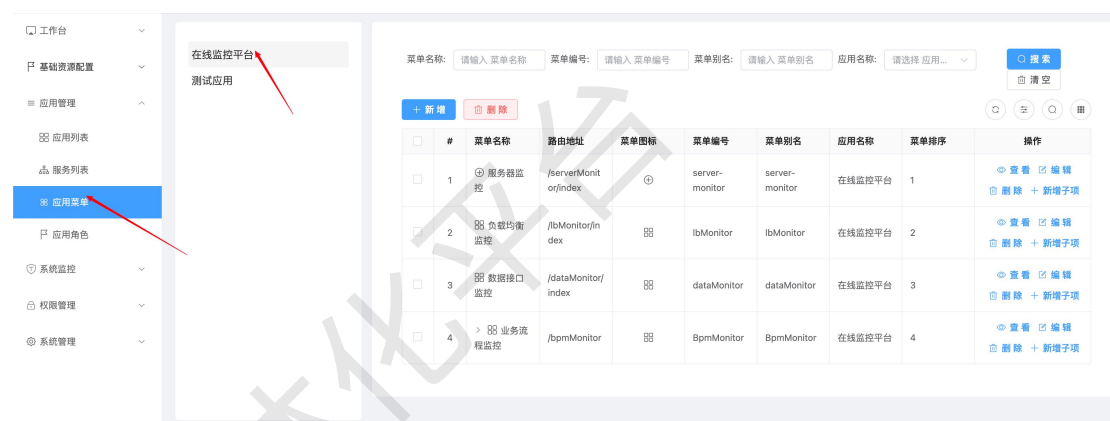
### 3.5.4. 应用角色管理

点击选择右侧应用角色，选择对应的应用，可以进行角色管理，创建角色等操作。



### 3.5.4. 应用菜单管理

点击选择右侧应用菜单，选择对应的应用，可以进行菜单管理，创建菜单等操作。



## 4.开发规范

### 4.1. 命名规范

#### 4.1.1 包命名

- (1) **全部小写**：包名应全部使用小写字母，并且使用点（.）分隔，如 `com.example.project`。
- (2) **有意义**：包名应使用有意义的单词或短语，以便于理解和维护。
- (3) **遵循反向域名规则**：包名应遵循反向域名规则，这有助于避免命名冲突。

#### 4.1.2 类命名

- (1) **首字母大写**：类名应以大写字母开头，使用驼峰命名法（大驼峰），即每个单词的首字母都大写，如 `HelloWorld`。
- (2) **名词或名词短语**：类名通常是名词或名词短语，应具有描述性。
- (3) **接口命名**：接口名除了可以用名词和名词短语外，还可以使用形容词或形容词短语，如 `Cloneable`，表示实现该接口的类具有某种功能或能力。

#### 4.1.3 方法命名

- (1) **首字母小写**：方法名应以小写字母开头，使用驼峰命名法（小驼峰），即除了第一个单词外，每个单词的首字母都大写，如 `showMessage`。

(2) **动词或动词短语**：方法名通常使用动词或动词短语，与参数或参数名共同形成动宾短语，即动词+名词。

#### 4.1.4 变量命名

- (1) **首字母小写**：变量名也应以小写字母开头，使用驼峰命名法（小驼峰）。
- (2) **描述性**：变量名应具有描述性，能够反映其用途或所代表的数据。
- (3) **避免单个字符**：尽量避免使用单个字符作为变量名，除非是在循环计数器等特殊情况下。

#### 4.1.5 常量命名

- (1) **全部大写**：常量名应全部使用大写字母，并使用下划线（\_）分隔单词，如 `MAX_VALUE`。
- (2) **避免缩写**：常量名应尽量避免使用缩写，除非该缩写是广泛认可的。

#### 4.1.6 构造方法命名

- (1) **与类名相同**：构造方法的命名应与类名相同，并且没有返回类型（即使用 `void` 以外的任何类型都是错误的）。
- (2) **参数描述性**：构造方法的参数应具有描述性，并且与类的属性名称保持一致。

### 4.2. 代码规范

#### 4.2.1 缩进和格式

- (1) **统一缩进**：使用空格或制表符（Tab）进行缩进，但整个项目应保持一致。通常推荐使用 4 个空格作为缩进单位。
- (2) **大括号风格**：对于大括号（`{}`）的放置，Java 社区中有多种风格（如 K&R 风格、Allman 风格等）。选择一种风格并在项目中保持一致。
- (3) **行宽限制**：设置合理的行宽限制（如 80 或 120 个字符），过长的行应适当换行以提高可读性。

#### 4.2.2 编码风格



- (1) **代码可读性**: 编写易于阅读的代码, 避免使用过于复杂的表达式或结构。
- (2) **一致性**: 在整个项目中保持一致的编码风格, 包括命名、缩进、空格使用等。
- (3) **避免硬编码**: 尽量使用常量或配置文件来管理可能变化的值, 避免在代码中直接硬编码。

### 4.2.3 依赖管理

- (1) **使用构建工具**: 使用 Maven、Gradle 等构建工具来管理项目依赖。
- (2) **避免过度依赖**: 尽量减少对外部库的依赖, 特别是那些重量级或不稳定的库。
- (3) **版本控制**: 在构建文件中明确指定依赖的版本号, 避免因为依赖更新而导致的问题。

## 4.3. 其他

### 4.3.1 单元测试

- (1) **编写单元测试**: 为关键代码编写单元测试, 确保代码的正确性和稳定性。
- (2) **测试覆盖率**: 尽量提高测试覆盖率, 特别是对公共方法和关键逻辑的测试。
- (3) **持续集成**: 将单元测试集成到持续集成流程中, 确保每次提交都通过测试。

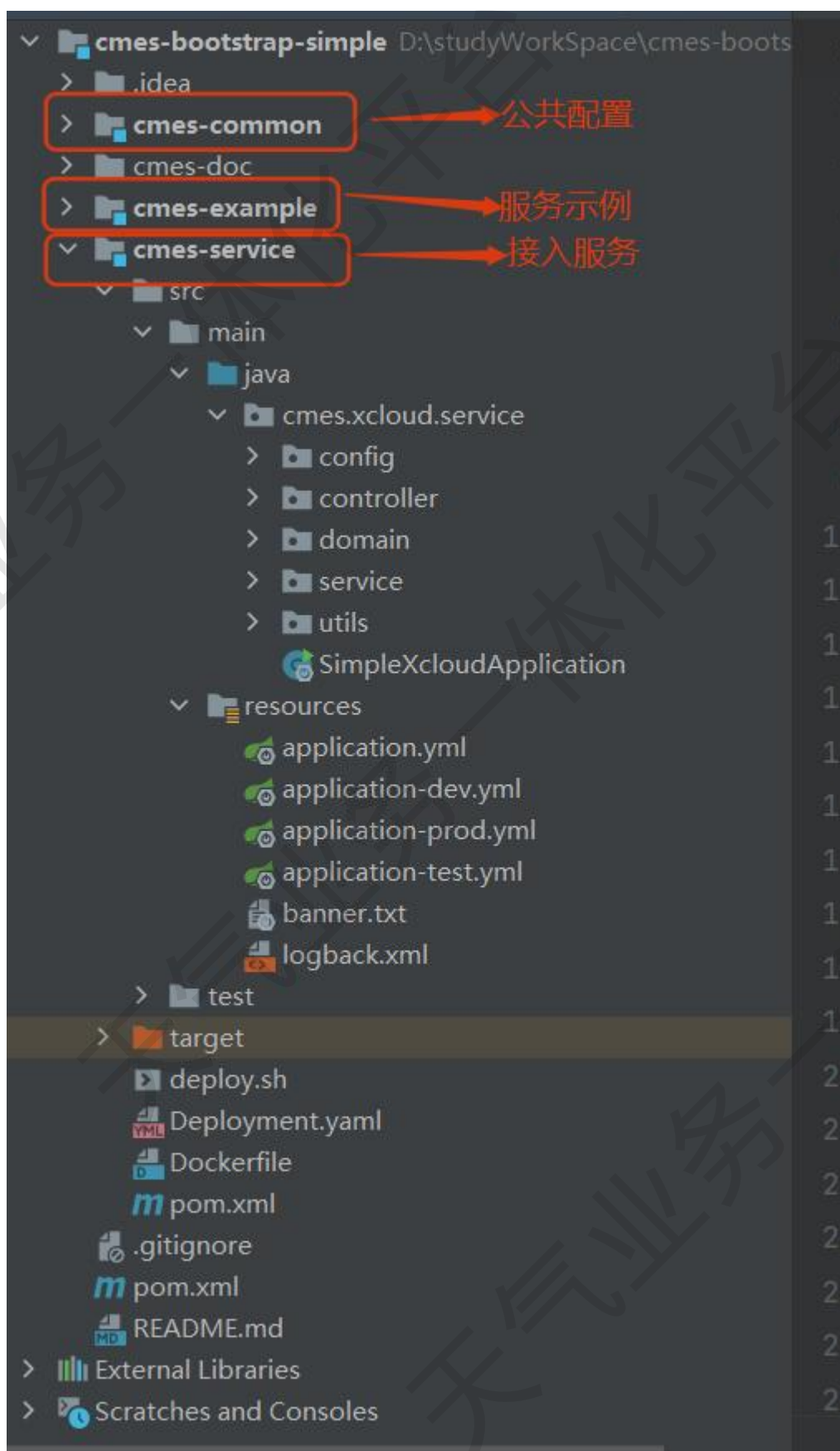
### 4.3.2 异常处理

- (1) **合理抛出异常**: 在方法签名中声明可能抛出的异常, 并在适当的位置抛出异常。
- (2) **捕获并处理异常**: 在调用可能抛出异常的方法时, 应捕获并处理这些异常, 避免程序崩溃。
- (3) **避免吞没异常**: 捕获异常后应进行处理或记录日志, 避免简单地吞没异常而不做任何处理。

## 5. 开发示例

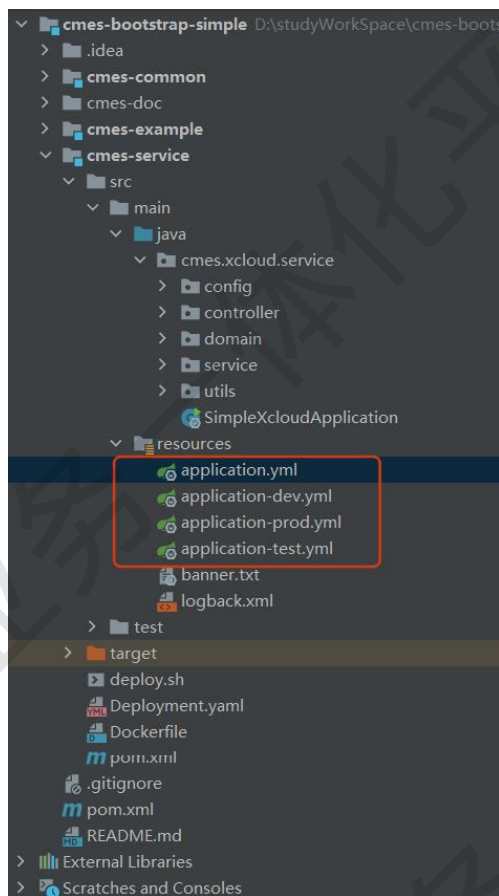
### 5.1. 基础使用

### 5.1.1 项目简介

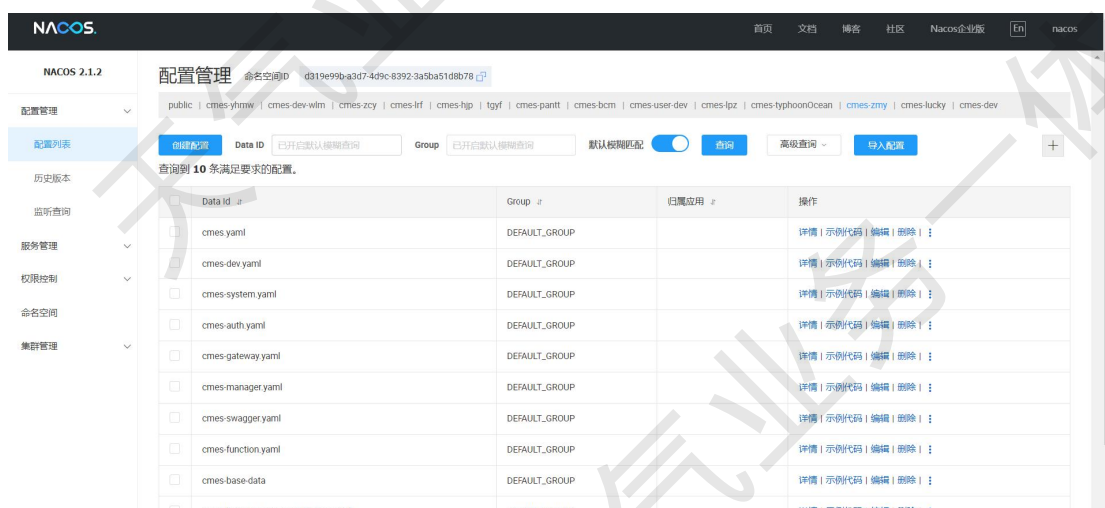


## 5.1.2 接入配置

接入系统配置需要将配置按环境写入 yml 文件中

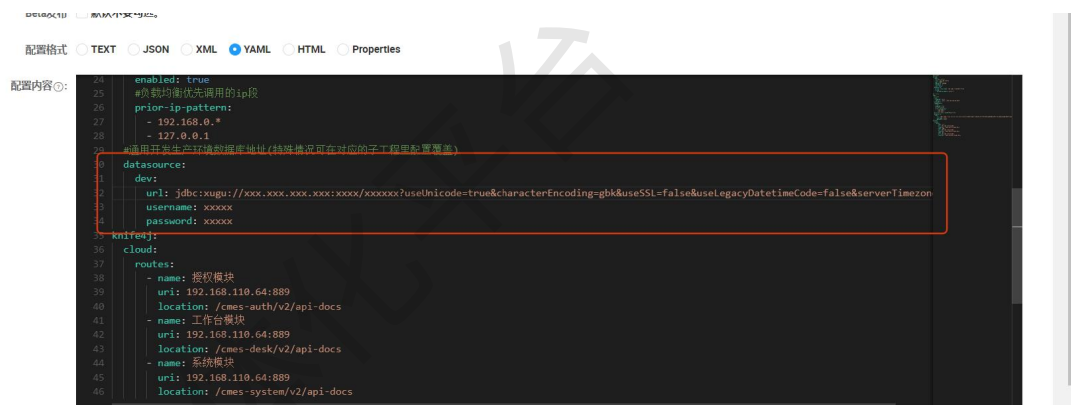


针对公共配置则可配置于线上 nacos



## 5.2. 数据库

公有配置可参考 nacos 中的 spring.datasource



私有配置则可以配置于个人项目

### 5.3. 接口文档

接口文档接入:

- 1, 注册 APIFox, 官网: <https://apifox.com/>
- 2, 寻找所在地一体化管理员, 加入一体化 APIFox 项目

### 5.4. 基础服务

项目提供包含权限, 角色, 用户, 登录, 机构, 菜单等相关项目, 详情请参考:

[一体化分系统](#)

### 5.5. 扩展融入

拓展组件相关, 可引入 cmes-tools 项目后新增项目, 并发布私库, 即可引用:

