# 地球系统数值模拟装置项目
# 区域高精度长期气候变化风险模拟分系统
# 平台使用和操作流程
## 培训教程

清华大学
航天宏图信息技术股份有限公司
2022年5月

# 目录

CONTENTS


01 安装部署

02 目录结构

03 集成规范

04 平台使用

# 01

# 安装部署

# 安装部署

## 软硬件环境

| | | 硬件环境配置信息 | | | | |
|---|---|---|---|---|---|---|
| 硬件配置 | 序号 | 硬件类型 | 服务器用途 | CPU | 内存 | 节点数量 |
| | 1 | 计算节点 | 应用服务器 | 2*Hygon C86 7185 32-core Processor | 256GB | 1960 |
| | 2 | 计算节点 | 应用服务器 | 2*Hygon C86 7185 32-core Processor | 512GB | 120 |
| | 3 | 计算节点 | 应用服务器 | 2*Hygon C86 7185 32-core Processor | 1024GB | 120 |
| | 4 | 计算节点 | 应用服务器 | 2*Hygon C86 7185 32-core Processor 2*DCU加速卡 | 128GB | 120 |
| | 5 | 存储节点 | 存储服务器 | 2*Hygon C86 7185 32-core Processor | 2PB | - |
| | 备注 | 本分系测试需要200核并行运算环境以及200TB存储 | | | | |

| 软件环境配置信息 | | | |
|---|---|---|---|
| 软件配置 | 服务器用途 | 操作系统版本 | 应用软件版本 |
| | 应用服务器 | Linux version 3.10.0-862.el7.x86_64 (higon@Chengdu09) (gcc version 4.8.5 20150623 (Red Hat 4.8.5-28) (GCC) ) | 基础库:<br>Jasper v1.701.0、ZLIB v1.2.8、netcdf-4.4.0、netcdf-cxx4-4.3.0、netcdf-fortran-4.4.4、HDF5 v1.10.0、MPICH v3.3.1、Pnetcdf v1.3.1<br>编译器:<br>Python v3.8.3、ifort version 17.0.5、GUN v4.8.5<br>应用软件:<br>NCL v6.6.2、CMake v3.18、Chrome V81.0、Xftp V6、Xshell V6、NotePad++ V7.9、NCO 4.9.1、CDO 1.9.7、NCL 6.6.2、Matlab、R version 3.6.1、IDL84envi52、MySQL5.7以上、redis5以上、nginx |
| | 数据库服务器 | Red Hat Enterprise Linux Server release 6.8 | Postgresql 12.1 |

# 安装部署

## 环境变量设置

1.打开~/.bashrc文件；

2.配置如下环境变量；

```
module purge
module remove mpi/hpcx/2.7.4/gcc-7.3.1
module load compiler/intel/2017.5.239
module load mpi/intelmpi/2017.4.239
module load mathlib/hdf5/intel/1.8.20
module load mathlib/libpng/intel/1.2.12
module load mathlib/zlib/intel/1.2.11
module load mathlib/jpeg/intel/9a
module load mathlib/jasper/intel/1.900.1
module load mathlib/szip/intel/2.1.1
module load mathlib/hdf4/intel/4.2.13
module load mathlib/netcdf/intel/4.4.1
module load mathlib/pnetcdf/intel/1.12.1
module load apps/nco/intel/4.8.1
module load apps/ncview/intel/2.1.7
module load mathlib/cdo/intel/1.10.19
module load mathlib/udunits/intel/2.2.28
module load apps/ncl_ncarg/6.6.2
module load apps/anaconda3/5.3.0

export SYSTEMPATH=/data/luoyong/H03
export QT_QPA_PLATFORM='offscreen'
```

```
(base) [luoyong@login03 ~]$ vi ~/.bashrc
```

```
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=
#module use ~/.module
module purge
module remove mpi/hpcx/2.7.4/gcc-7.3.1
module load compiler/intel/2017.5.239
module load mpi/intelmpi/2017.4.239
module load mathlib/hdf5/intel/1.8.20
module load mathlib/libpng/intel/1.2.12
module load mathlib/zlib/intel/1.2.11
module load mathlib/jpeg/intel/9a
module load mathlib/jasper/intel/1.900.1
module load mathlib/szip/intel/2.1.1
module load mathlib/hdf4/intel/4.2.13
module load mathlib/netcdf/intel/4.4.1
module load mathlib/pnetcdf/intel/1.12.1
module load apps/nco/intel/4.8.1
module load apps/ncview/intel/2.1.7
module load mathlib/cdo/intel/1.10.19
module load mathlib/udunits/intel/2.2.28
module load apps/ncl_ncarg/6.6.2
```

# 安装部署

## Python环境配置

1. 安装miniconda，确认安装目录并确认conda init 初始化

```
Miniconda3 will now be installed into this location:
/public/home/luoyong/miniconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/public/home/luoyong/miniconda3] >>>
```

或者 module load apps/anaconda3/5.3.0

# 安装部署

## Python环境配置

2. 配置conda环境，环境名包括h03，hazard，py37，lowtemp，hightemp，dryrisk，bdfx，swfx。
3. 进入miniconda/envs所在目录，建立上述同名文件夹，拷贝安装包中的同名压缩文件到同名文件夹，解压上述环境名压缩包

```
(base) [luoyong@login01 miniconda3]$ cd envs/
(base) [luoyong@login01 envs]$ tar -xvf lowtemp.tar.gz
```

# 安装部署

## Python环境配置

4. conda env list 查看所有python环境
5. conda activate [env_name] 逐一载入第3步中的环境，检测conda环境是否安装成功，

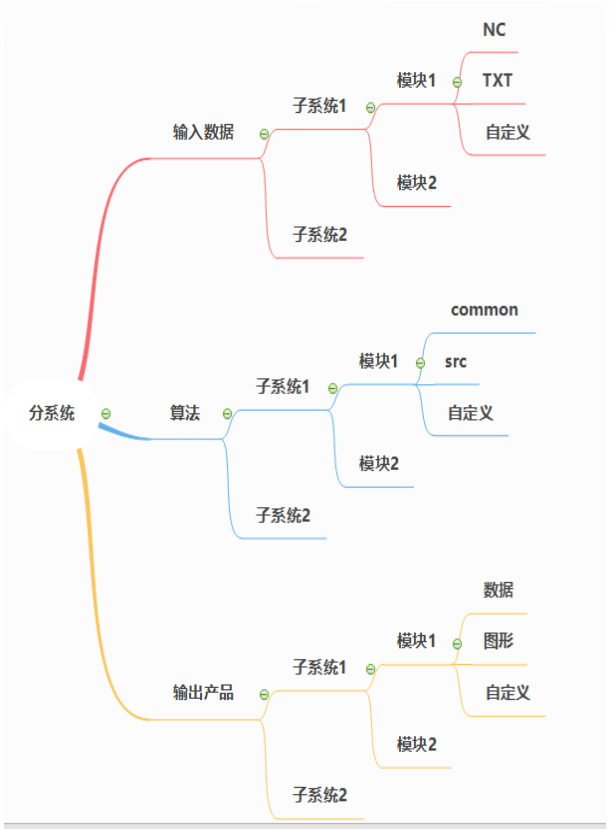
```
(base) [luoyong@server02 SWFX]$ conda env list
# conda environments:
#
base                  *  /public/home/luoyong/package/miniconda3
bdfx                     /public/home/luoyong/package/miniconda3/envs/bdfx
dryrisk                  /public/home/luoyong/package/miniconda3/envs/dryrisk
h03                      /public/home/luoyong/package/miniconda3/envs/h03
h03_2                    /public/home/luoyong/package/miniconda3/envs/h03_2
hazard                   /public/home/luoyong/package/miniconda3/envs/hazard
hightemp                 /public/home/luoyong/package/miniconda3/envs/hightemp
lowtemp                  /public/home/luoyong/package/miniconda3/envs/lowtemp
mirs                     /public/home/luoyong/package/miniconda3/envs/mirs
py37                     /public/home/luoyong/package/miniconda3/envs/py37
qgis                     /public/home/luoyong/package/miniconda3/envs/qgis
raindan                  /public/home/luoyong/package/miniconda3/envs/raindan
swfx                     /public/home/luoyong/package/miniconda3/envs/swfx
```

```
(base) [luoyong@login01 envs]$ conda activate lowtemp
(lowtemp) [luoyong@login01 envs]$
```

# 02

# 目录结构

# 目录结构




分系统目录：${SYSTEMPATH}（固定）

分系统输入数据目录：${SYSTEMPATH}/inputdata（固定）

分系统算法目录：${SYSTEMPATH}/algorithm（固定）

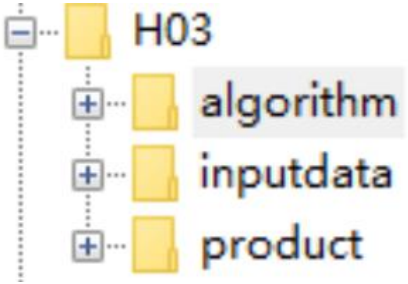
分系统输出产品目录：${SYSTEMPATH}/product（固定）。

# 目录结构

复制 H03.tar.gz到安装目录下，解压H03.tar.gz，然后进入H03目录。

```
(base) [luoyong@login01 H03]$ ls
algorithm  inputdata  product
```

H03目录结构如下：




其中，algorithm为所有子系统的算法脚本目录，inputdata为所有子系统的输入数据目录，product为所有子系统的输出产品目录。以上三个目录内结构一致，均包含10个子系统目录

# 目录结构

以上三个目录内结构一致，均包含子系统及其各模块目录（详见部署方案）


子系统及其各模块目录缩写列表如下：

| 子系统名称 | 子系统缩写 | 模块名称 | 模块缩写 |
|---|---|---|---|
| 全球气候变化趋势分析子系统 | QQQH | 全球气候变化事实分析模块 | SSFX |
| | | 全球气候模拟试验分析模块 | MNSY |
| | | 全球气候系统能量-水循环演变模拟分析模块 | NLSH |
| | | 全球气候变化归因分析模块 | BHGY |
| | | 全球气候敏感性估算模块 | MGGS |
| | | 全球未来气候变化趋势分析模块 | WLBH |
| | | 全球气候变化集成预估数据集开发模块 | YGSJ |

# 03

# 集成规范

# 集成规范

使用**bash**脚本集成多源代码：shell、python、NCL、R、Fortran

每个模块主要由3个组件组成：

- run_{模块缩写}.sh：主程序，诊断分析运行入口；
- useconfig.json：用户配置文件；
- scripts：NCL、Python等诊断分析程序；

# 集成规范

## useconfig.json

```
{
    "analyseStrtYear": 1979,
    "analyseEndYear": 2014,
    "refPeriodStart": 19790101,
    "refPeriodEnd": 20141231,
    "hist_syr": 1850,
    "hist_eyr": 2014,
    "latS": 0,
    "latN": 80,
    "lonW": 60,
    "lonE": 180,
    "modelNum": 1,
    "modelName": "CAS-ESM2-0",
    "modelProject": "CMIP6",
    "modelMip": "day",
    "modelExp": "historical",
    "modelEnsemble": "r1i1p1f1",
    "modelGrid": "gn",
    "ynOutPlot": 1,
    "outPlotType": "png",
    "ynOutData": 1,
    "obsDataDir": "/data/luoyong/H03/inputdata/QQQH/WLBH/OBS",
    "modelDataDir": "/data/luoyong/H03/inputdata/common_data/",
    "productDir": "/data/luoyong/H03/product/QQQH/WLBH/",
    "resultJsonFile": "/data/luoyong/H03/product/QQQH/WLBH/output.json",
    "algorithmLogFile": "/data/luoyong/H03/product/QQQH/WLBH/output.log",
    "algorithmFlowFile": "/data/luoyong/H03/product/QQQH/WLBH/outputFlow.json"
}
```

| 参数名称 | 参数描述 |
| --- | --- |
| analyseStrtYear | 分析起始年 |
| analyseEndYear | 分析终止年 |
| refPeriodStart | 气候态起始年份 |
| refPeriodEnd | 气候态终止年份 |
| hist_syr | 历史时段起始年份 |
| hist_eyr | 历史时段终止年份 |
| modelName | 输入模式数据集名 |
| modelProject | 耦合模式计划类型 |
| modelMip | 模式时间频率 |
| modelExp | 耦合模式比较计划试验 |
| modelEnsemble | 耦合模式比较计划集合 |
| modelGrid | 耦合模式比较格点标签 |
| latS | 区域南边界纬度 |
| latN | 区域北边界纬度 |
| lonW | 区域西边界经度 |
| lonE | 区域东边界经度 |
| inputDataDir | 输入数据目录 |
| productDir | 输出目录 |
| resultJsonFile | 输出结果json |
| algorithmLogFile | 输出日志json |
| algorithmFlowFile | 输出流程json |

# 集成规范

## run_{模块缩写}.sh

```
#!/bin/bash
#source ~/.bashrc
```

### 判断是否传递用户配置参数文件,不存在则终止程序

```
if [ -z $1 ];then
    echo "未输入参数配置文件userconfig.json"
    exit 1
fi
```

### 程序记录开始时间

```
starttime=$(date '+%Y-%m-%d %H:%M:%S') # 记录程序开始事件
MODULEM="QQQH"  # 分系统名称 {全球气候}
MODULE="SSFX"   # 定义模块英文缩写 {事实分析}
MODULENAME="全球气候变化事实分析" # 定义模块中文名称
```

### 接收日志文件输出路径

```
log_file=$(cat $1 | jq .algorithmLogFile -r)  # 使用jq解析输入json文件 algorithmLogFile 为算法输出日志 json 关键词
json_file=$(cat $1 | jq .resultJsonFile -r) # 输出json文件
flow_file=$(cat $1 | jq .algorithmFlowFile -r) # 输出流程图json
```

# 集成规范

run_{模块缩写}.sh

## 检查日志和json文件是否已经存在，存在则删除

```
if [ -f $log_file ];then
    rm $log_file
fi
if [ -f $json_file ];then
    rm $json_file
fi
if [ -f $flow_file ];then
        rm $flow_file
fi
```

## 日志输出脚本函数

此函数由于打印算法执行日志，并保存提供给平台组和前端显示 <br> 为前段展示中的换行符号

调用方法为 LOG “要打印的日志信息”

```
LOG()
{
    time=$(date '+%Y-%m-%d %H:%M:%S')
    echo "$time"" : "$1 <br> >${log_file}
    echo "$time"" : "$1
}
```

# 集成规范

## run_{模块缩写}.sh

### 程序初始化部分

```
SYSTEMPATH="/home/liuhaolin/qixiangyaogan/H03" # 读取整个算法系统路径 清华服务器上为/data/qhbh01/H03
ALGPATH="${SYSTEMPATH}/algorithm/{MODULEM}/{MODULE}" # 算法所在路径
LOG "主程序初始化"  # 第一次打印日志 记录程序开始执行
strtyear=$(cat $1 | jq .analyseStrtYear -r) # 解析输入json中的参数 起始年份
lastyear=$(cat $1 | jq .analyseEndYear -r) # 结束年份
OBSINPUTPATH=$(cat $1 | jq .obsDataDir -r) # 输入观测数据
OBSOUTPUTPATH=$(cat $1 | jq .productDir -r) # 输出结果路径
# 导出相关的环境变量给ncl主程序算法
export OBSINPUTPATH=${OBSINPUTPATH}
export OBSOUTPUTPATH=${OBSOUTPUTPATH}
export STARTYEAR=${strtyear}
export LASTYEAR=${lastyear}
```

### 算法参数解析完毕后输出流程和日志

```
LOG "完成参数解析"
jq -n '{"product":"'$MODULE'"}' > $flow_file

TIMEF=$(date '+%Y-%m-%d %H:%M:%S')
jq '.step[.step| length] |= . + {"stepname": "解析输入参数", "stepnum": "1", "status": "1",
                                 "message": "解析输入参数--成功","opertime":$TIMEF}' \
        --arg TIMEF "$TIMEF" $flow_file > tmp.json && mv tmp.json $flow_file
```

# 集成规范

## run_{模块缩写}.sh

### 执行主程序部分

```
##############################执行obs_trend.ncl主程序算法###
LOG "执行主程序算法 obs_trend.ncl"
cd ${ALGPATH}/trend # 进入算法程序所在路径
run_ncl obs_trend 2 # 调用执行ncl脚本函数
```

执行脚本
（不带后缀）

执行顺序

### ncl算法执行函数

算法输入为要执行的算法名称和执行此算法的步骤n.

```
run_ncl()
{
        # 检查调用ncl程序是否存在 若存在删除
        if [ -f $1.log ];then

                rm $1.log
        fi
        # 调用ncl算法函数 并将执行结果写入算法名相同的Log日志 记录算法执行耗时
        TIME="$( TIMEFORMAT='%lR';time (ncl $1.ncl > $1.log) 2>&1 1>/dev/null )"
        # 在算法执行日志中抓取算法执行的错误信息，不同语言编写的算法错误信息不同，需要提前执行错误算法测试错误信息。
        # ncl程序错误信息包含 fatal、 error at line等， 错误检查信息用于程序执行成功判定
        ERRCHECK=$( grep -e "fatal\:error at line" -e "fatal" -e "Error occurred at or near line" $1.log )
        # 算法执行情况判定部分
        if [[ -z $ERRCHECK ]];then # 如果算法执行错误信息为空
                # 打印算法执行成功日志 并输出算法执行耗时
                LOG "主程序算法 $1.ncl 执行完毕,耗时$TIME"
                # 记录算法执行完毕事件
                TIMEF=$(date '+%Y-%m-%d %H:%M:%S')
                # 使用jq程序编写算法执行流程图json
                # 包含关键字信息如下stepname:步骤名称，stepnum:步骤编号，status:执行状态,输出信息:message，完成时间:opertime.
                jq '.step[.step| length] |= . + {"stepname": $srcname, "stepnum":$stepnum,
                                                 "status": "1", "message": $message,"opertime":$TIMEF}' \
                        --arg srcname "执行主程序算法$1.ncl" \ # 步骤名称:执行输入的主程序算法xxx.ncl
                        --arg stepnum "$2" \ # 步骤编号，通用模块第一步为解析参数，因此算法执行步骤从2开始
                        --arg message "执行主程序算法$1.ncl--成功" \ # 此处为执行成功输出信息
                --arg TIMEF "$TIMEF" $flow_file > tmp.json && mv tmp.json $flow_file # 输出流程图json更新
                export STATUS${2}=1 # 将第n步算法执行状态返回 作为最终执行情况判定结果
                rm $1.log # 算法执行成功情况下删除该算法执行日志
        else # 如果算法执行失败
                TIMEF=$(date '+%Y-%m-%d %H:%M:%S') #记录算法执行失败时间
                LOG "主程序算法 $1.ncl 执行失败,错误详见${PWD}/$1.log" # 输出算法执行失败日志路径供查错使用
                # 使用jq记录算法执行失败流程图json
                jq '.step[.step| length] |= . + {"stepname": $srcname, "stepnum": $stepnum, "status": "0",
                                                 "message": $message,"errorlog":$errorlog,"opertime":$TIMEF}' \
                        --arg srcname "执行主程序算法$1.ncl" \ # 步骤名
                        --arg stepnum "$2" \ # 步骤号码
                        --arg message "执行主程序算法$1.ncl--失败" \ # 算法执行失败吐露信息
                        --arg errorlog "${PWD}/$1.log" \ # 错误日志记录
                        --arg TIMEF "$TIMEF" $flow_file > tmp.json && mv tmp.json $flow_file # 更新算法流程json
                export STATUS${2}=0 # 返回执行失败状态0
        fi
}
```

# 集成规范

## run_{模块缩写}.sh

### Python脚本集成

```
run_python()
{
        if [ -f $1.log ];then
                rm $1.log
        fi
        TIME="$( TIMEFORMAT='%lR';time (python $1.py > $1.log 2>&1) 2>&1 1>/dev/null )"
        #TIME="$( TIMEFORMAT='%lR';time (python $1.py $2) )"
        ERRCHECK=$( grep -e "*Error*" $1.log )
        if [[ -z $ERRCHECK ]];then
        LOG "主程序算法 $1.py 执行完毕,耗时$TIME"
        TIMEF=$(date '+%Y-%m-%d %H:%M:%S')
        jq '.step[.step| length] |= . + {"stepname": $srcname, "stepnum":$stepnum, "status": "1", "messa
ge": $message,"opertime":$TIMEF}' \
                        --arg srcname "执行主程序算法$1.py" \
                        --arg stepnum "$2" \
                        --arg message "执行主程序算法$1.py--成功" \
                --arg TIMEF "$TIMEF" $flow_file > tmp.json && mv tmp.json $flow_file
                export STATUS${3}=1
    #rm $1.log
        else
        TIMEF=$(date '+%Y-%m-%d %H:%M:%S')
                LOG "主程序算法 $1.py 执行失败,错误详见${PWD}/$1.log"
        jq '.step[.step| length] |= . + {"stepname": $srcname, "stepnum": $stepnum, "status": "0", "mess
age": $message,"errorlog":$errorlog,"opertime":$TIMEF}' \
                        --arg srcname "执行主程序算法$1.py" \
                        --arg stepnum "$2" \
                        --arg message "执行主程序算法$1.py--失败" \
                --arg errorlog "${PWD}/$1.log" \
                --arg TIMEF "$TIMEF" $flow_file > tmp.json && mv tmp.json $flow_file
                export STATUS${3}=0
        fi
}
```

### R语言脚本集成

```
run_R()
{
    if [ -f $1.log ]; then
        rm $1.log
    fi

    TIME="$( TIMEFORMAT='%lR';time (Rscript --no-save --no-restore --verbose $1.R $OBSINPUTPATH $OBSOUTP
UTPATH > $1.log 2>&1 ) 2>&1 1>/dev/null )"
    ERRCHECK=$( grep -e "Execution halted" -e "停止执行" $1.log )
    if [[ -z $ERRCHECK ]];then
        LOG "主程序算法 $1.R 执行完毕, 耗时$TIME"
        TIMEF=$(date '+%Y-%m-%d %H:%M:%S')
        jq '.step[.step| length] |= . + {"stepname": $srcname, "stepnum": $stepnum, "status": "1", $mess
age, "opertime": $TIMEF}'\
            --arg srcname "执行主程序算法$1.R" \
            --arg stepnum "$2" \
            --arg message "执行主程序算法$1.R--成功"\
            --arg TIMEF "$TIMEF" $flow_file > tmp.json && mv tmp.json $flow_file
        export STATUS${2}=1

    else

       TIMEF=$(date '+%Y-%m-%d %H:%M:%S')
        LOG "主程序算法 $1.R 执行失败,错误详见${PWD}/$1.log"

        jq '.step[.step| length] |= . + {"stepname": $srcname, "stepnum": $stepnum, "status": "0", $mess
age,"errorlog":$errorlog,"opertime":$TIMEF}' \
            --arg srcname "执行主程序算法 $1.R" \
            --arg stepnum "$2" \
            --arg message "执行主程序算法 $1.R--失败"\
            --arg errorlog "${PWD}/$1.log" \
            --arg TIMEF "$TIMEF" $flow_file > tmp.json && mv tmp.json $flow_file
        export STATUS${2}=0
    fi
}
```

# 集成规范

## run_{模块缩写}.sh

### 输出产品写入output.json

根据各个ncl程序执行返回状态判定该模块是否执行成功，根据成功失败状态分别写入流程json和输出json文件

1. 此节的输出产品ls部分的文件格式*.xxx需要根据算法执行的具体产出文件格式更改
2. 此节的执行步骤号码（n=5）需根据前面算法程序的数量决定,该SSFX算法示例中有3个子程序因此步骤为：

1. 解析参数
2. 执行算法1
3. 执行算法2
4. 执行算法3
5. 输出产品和json，流程等文件

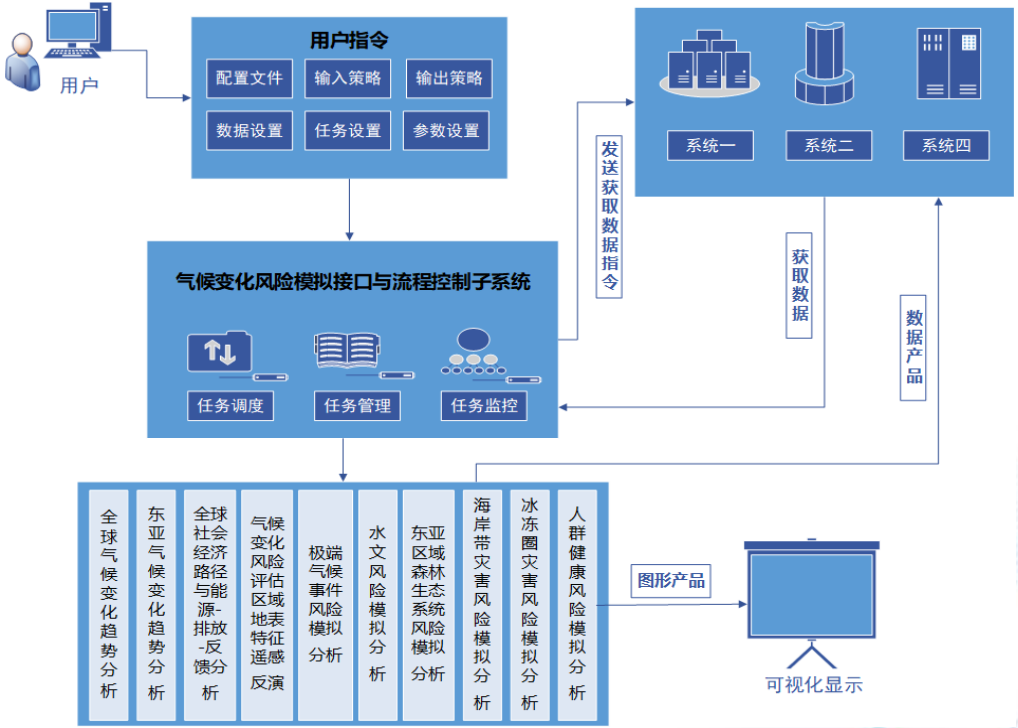
```
if [[ ($STATUS2 -eq 1) && ($STATUS3 -eq 1) && ($STATUS4 -eq 1) ]];then # 若三个ncl算法同时执行成功，输出json记录算法执行状态1
    STATUS=1
    jq -n '{status:'$STATUS',
    message: $message,
    starttime: $starttime,
    endtime: $endtime,
    figresult:{filePath:$fpath1,fileNames:$flist1,productType: "'${MODULE}'",
                                "productName":"'${MODULENAME}'","productFormat":"png"},
    dataresult:{filePath:$fpath2,fileNames:$flist2,productType: "'${MODULE}'",
                                "productName":"'${MODULENAME}'","productFormat":"txt"}}' \
        --arg message "$MODULE success!" \  # 吐露信息 算法执行成功
        --arg starttime "$starttime" \ # 开始执行时间
        --arg endtime "$endtime" \ # 执行完成时间
        --argjson flist1 "$(cd ${OBSOUTPUTPATH}/ && ls *.png | jq -R '[.]' | jq -s -c 'add')"\
        # 列出产品执行产出的图片产品 写入json
        --argjson flist2 "$(cd ${OBSOUTPUTPATH}/index/ && ls *.txt | jq -R '[.]' | jq -s -c 'add')"\
        --arg fpath2 "${OBSOUTPUTPATH}/index" \ # 列出产品产出的txt数据结果 写入json
        --arg fpath1 "$OBSOUTPUTPATH" > $json_file # 输出到执行结果文件

        TIMEF=$(date '+%Y-%m-%d %H:%M:%S')
        # 保存执行结果流程
    jq '.step[.step| length] |= . + {"stepname": "结果保存", "stepnum": "5",
                                    "status": "1", "message": "结果保存--成功","opertime":$TIMEF}' \
        --arg TIMEF "$TIMEF" $flow_file > tmp.json && mv tmp.json $flow_file
```

# 04

# 平台使用

# 平台使用

## 应用流程


用户
用户指令
配置文件
输入策略
输出策略
数据设置
任务设置
参数设置
系统一
系统二
系统四
发送获取数据指令
获取数据
数据产品
气候变化风险模拟接口与流程控制子系统
任务调度
任务管理
任务监控
全球气候变化趋势分析
东亚气候变化趋势分析
全球社会经济路径与能源-排放-反馈分析
气候变化风险评估区域地表特征遥感反演
极端气候事件风险模拟分析
水文风险模拟分析
东亚区域森林生态系统风险模拟分析
海岸带灾害风险模拟分析
冰冻圈灾害风险模拟分析
人群健康风险模拟分析
图形产品
可视化显示

# 任务实施完成情况

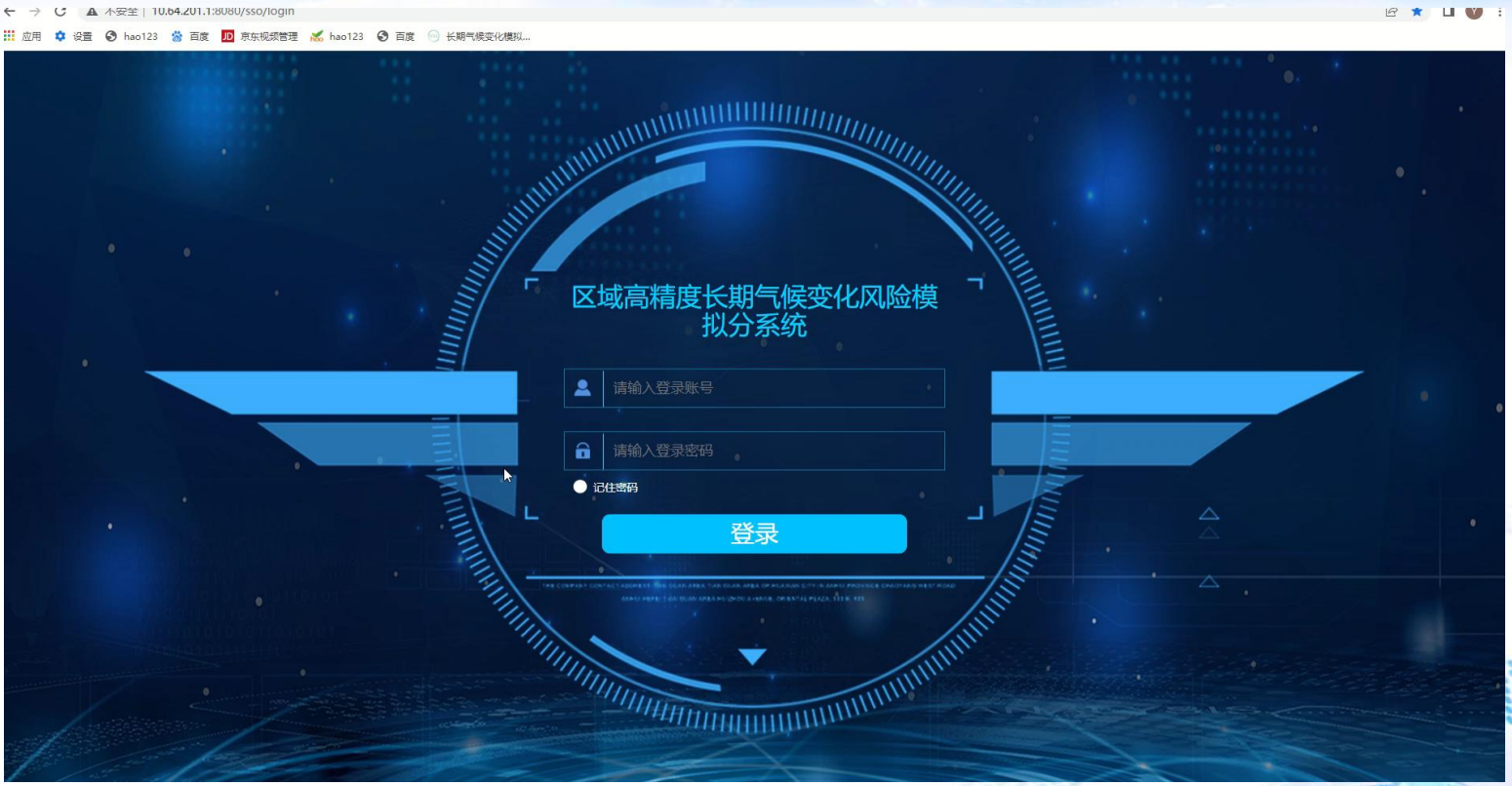
登陆界面

数据存储

产品生成

算法管理

系统管理




平台使用需要登陆大装置VPN

# 任务实施完成情况

登陆界面

**数据存储**

产品生成

算法管理

系统管理

# 任务实施完成情况

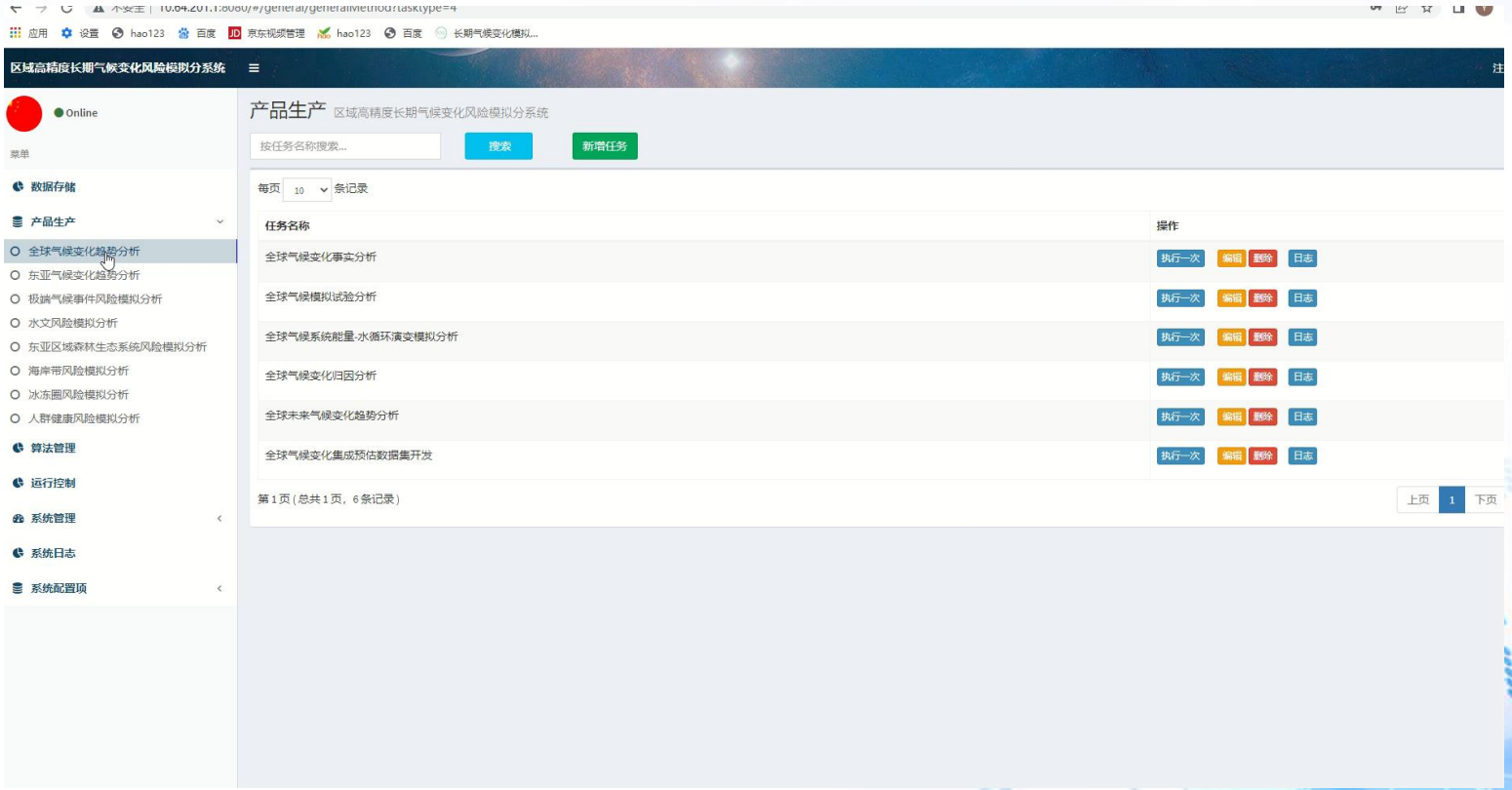
登陆界面

数据存储

**产品生成**

算法管理

系统管理



# 任务实施完成情况

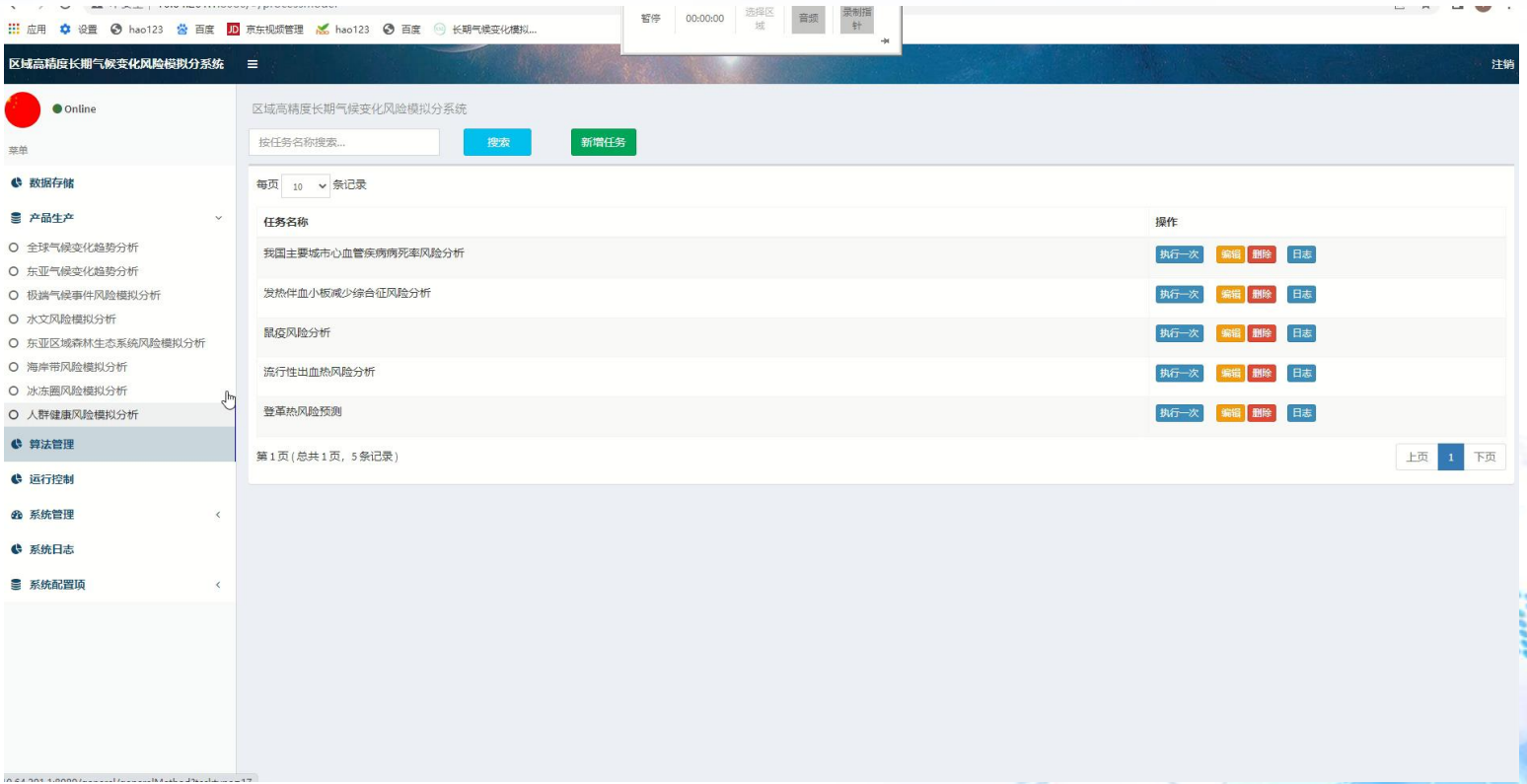
登陆界面

数据存储

产品生成

**算法管理**

系统管理



# 任务实施完成情况

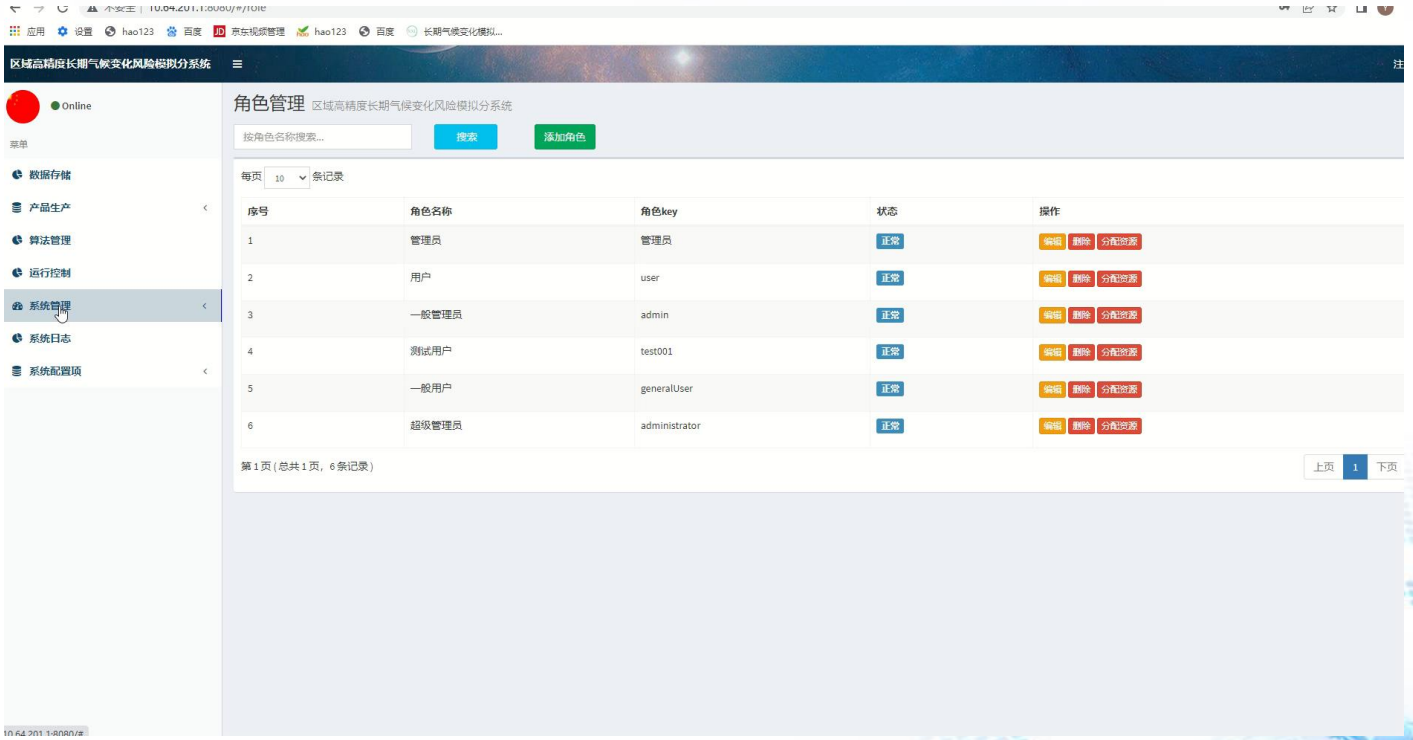
登陆界面

数据存储

产品生成

算法管理

**系统管理**



PIESAT

航天宏图信息技术股份有限公司

Piesat Information Technology Co., Ltd.

# 谢谢大家！

科技改变世界 遥感走进生活