

SDK

[SDK 介绍](#)

[安装和卸载](#)

[CloudPSS 框架介绍](#)

[框架图](#)

[仿真项目架构](#)

[CloudPSS SDK 快速入门](#)

[CloudPSS 接口文档](#)

[setToken 用户认证](#)

[Model 类](#)

- [ModelRevision 类](#)
- [ModelImplement 类](#)
- [ModelTopology 类](#)

[Runner 类](#)

- [Storage 类](#)
- [Receiver 类](#)
- [Result 类](#)

[Function 类](#)

[FunctionJob 类](#)

[Project 类](#)

[案例介绍](#)

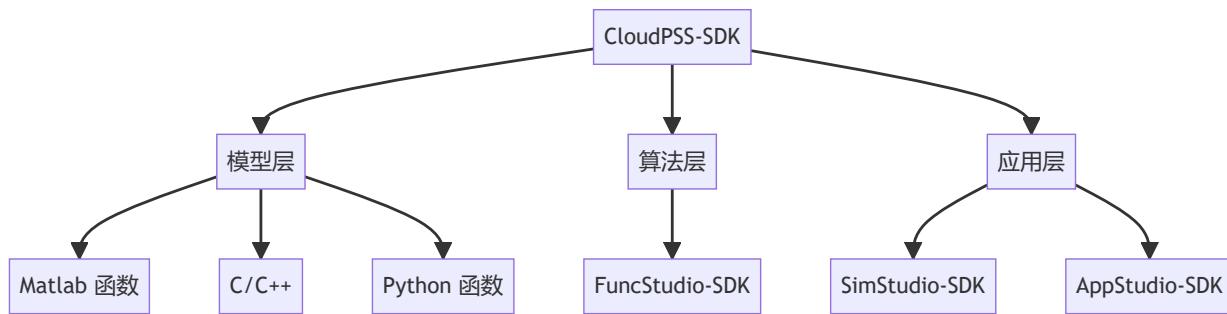
- [案例 1 通过项目 rid 运行项目](#)
- [案例 2 项目使用案例](#)
- [案例 3 运行项目并使用 matplotlib 动态绘制曲线](#)
- [案例 4 使用项目版本的 hash 运行算例文件](#)
- [案例 5 导入本地结果文件](#)
- [案例 7 使用综合能源仿真SDK](#)

SDK 简介

CloudPSS SDK 是基于 CloudPSS-API 封装的模型及软件开发套件。用户可通过编写 Python、Matlab 等脚本构建自定义模型，或是调用 CloudPSS 平台中的模型修改、仿真计算功能，实现诸如自动修改模型、批量仿真计算、自动化生成报告等复杂且繁琐的功能。用户也可在其自己的应用程序中调用 CloudPSS 仿真引擎，实现仿真驱动的高级分析应用。

CloudPSS SDK 包含模型层、算法层和应用层三种开发套件，其中：

1. **模型层开发套件** 帮助用户在 CloudPSS SimStudio 官方潮流计算、电磁暂态仿真、移频电磁暂态仿真、综合能源能量流计算等内核中开发第三方模型或用户自定义模型。目前，模型层 SDK 已开放基于 Matlab 函数的自定义控制元件接入，后续将进一步开放 Python、C/C++ 的标准元件开发套件。
2. **算法层开发套件** 帮助用户在 CloudPSS FuncStudio 中集成自己的算法内核，从而借助 CloudPSS XStudio 平台快速开发并部署自己的计算应用。
3. **应用层开发套件** 帮助用户在利用脚本的形式快速调用 CloudPSS 官方计算内核和第三方接入的计算内核，从而方便用户开发高级计算分析应用。其中，SimStudio-SDK 现已支持 SimStudio 中的模型修改和 **潮流计算**、**电磁暂态仿真**、**综合能源能量流计算** 三种计算内核。



SDK 安装与卸载

下载与安装

CloudPSS可以直接从[这里PyPi_](#)下载, 但是更方便的方式是通过[pip](#), 直接运行下面的命令进行[安装](#):

[PYHTON]

```
pip install cloudpss
```

需要[升级](#)的话, 执行下面的命令:

[PYHTON]

```
pip install --upgrade cloudpss
```

完全卸载

执行下面的命令进行[完全卸载](#):

[PYHTON]

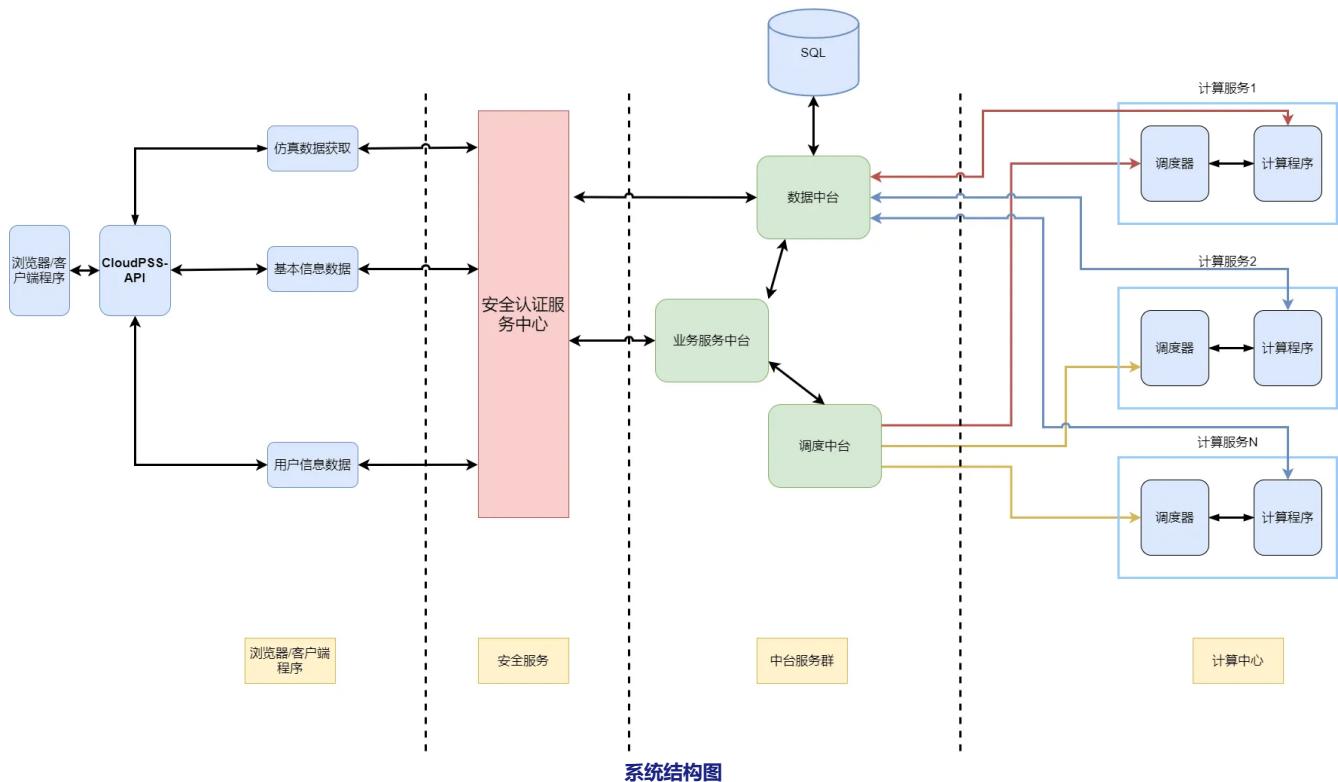
```
pip uninstall cloudpss
```

CloudPSS 框架介绍

- 框架图
- 仿真项目架构

框架图

系统结构图



浏览器/客户端程序

用户侧服务结构，一种是通过 CloudPSS 网站查询到用户的的具体信息，并提供网站对用户的资源进行访问，和执行对应权限下的操作。另一种是提供 CloudPSS 开放的 api 自己开放应用程序，从 CloudPSS 中心获取资源，和使用 CloudPSS 服务下的计算资源。

浏览器/客户端程序 浏览器：指使用浏览器访问到的 CloudPSS 网站的各个页面，客户端程序：指用户通过 CloudPSS 开放的 api 自己开发的应用程序。

CloudPSS-API 指用户通过 CloudPSS 开放的 api

仿真数据获取 指用户通过 api 从 CloudPSS 平台获取的仿真数据

基本信息数据 指用户通过 api 从 CloudPSS 平台获取的基本信息数据

用户信息数据 指用户通过 api 从 CloudPSS 平台获取的用户信息数据

安全服务

CloudPSS 的用户安全校验机制，用于校验当前用户的操作权限

安全日志服务中心 用于校验当前用户的操作权限

中台服务群

CloudPSS 后台服务器群，用于处理各个项业务逻辑

业务中台 所有中台的统称。

数据中台 数据中台用于融合整个数字孪生体与物理实体的全部数据，打通应用之间的数据隔阂，消除数据标准和口径不一致的问题。数据中台通过对来自多方面的基础数据进行清洗，按照特定主题域的概念建立多个以事务为主的主题域。本项目数据中台遵循三个一的概念，即 One ID（一个用户账号）、One Data（一个数据平台）、One Service（一个业务平台）。数据中台不仅仅是汇聚多元数据，而且让这些数据遵循相同的标准和口径，对事物的标识能统一或者相互关联，并且提供统一的数据服务接口。

调度中台 调度中台是业务中台中比较特殊的一种服务，其用于快速响应和调用各种业务功能，并根据计算量，基于现有计算资源做负载均衡

计算中心

由 CloudPSS 和用户通过调度中心共同组成的一个计算服务群，用户可以通过调度器将自己的算法程序注册到 CloudPSS 平台中，供其他用户调用。

调度器 CloudPSS 平台提供的一个程序调用服务，可以调用用户本地的计算程序

计算程序 CloudPSS 提供的计算服务，或者用户提供的计算服务

仿真项目架构

基本概念

CloudPSS 仿真的基本组织方式，一个解决方案是多个项目的集合，这多个项目没有直接的相关性，仅用于分组。

1 项目 (project)

项目是仿真计算的最小单位，存储了仿真的项目的拓扑和参数。一个系统属于某一个应用类型。

1.1 元件/子系统

组成项目的构件。每一个元件/子系统也是一个独立的项目，在由其他项目引用时被称为元件或子系统。在项目中，记录了每一个元件的项目版本 RID 或项目 RID，还保存了每个元件的参数和连接关系。

1.2 图纸/图层

对系统中元件进行组织，一个系统可以包含多个图纸，每张图纸包含多个图层。

2 参数集 (parameter set)

一个项目作为子系统使用时，其全局参数是由使用它的项目提供的。为了保持一致性，在独立使用一个项目作为主系统时，其全局参数仍旧不由系统内部提供，而是通过每个项目随附的参数集提供。

一个参数集与其关联的项目绑定，其中的参数与对应项目的全局变量声明相对应，可以提供该项目作为主系统独立仿真计算的参数。

一个项目中可以保存多个参数集，以便在多种用途下切换。

2.1 方案 (case)

记录了项目作为主系统启动应用的参数。

包括要启动的应用 RID，应用的启动参数。

一个项目可以保存多个方案。可以由参数集与方案构成计算矩阵，完成多种情况下的计算。

3 项目类型 (project type)

对项目预定义的类型，项目类型决定了在项目中能够使用的元件、子系统、全局参数等。一种应用（如电磁暂态、潮流）只能服务于特定的项目类型。

4 应用 (application)

对计算任务类型的定义，包含其名称、能够处理的应用类型、启动的任务定义 (job definition) 名称、参数等信息。

5 计算任务 (job)

使用调度中台管理的任务，是启动计算任务生成的任务实例。

6 项目/子系统及其管理

项目有自己的 RID。参照上文的描述。一个项目中实际只包含了参数集、方案相关信息和一些简单的元数据，以及系统最新版本 (project revision) 的 hash。系统的实际参数和拓扑信息由下述的版本系统进行管理。

7 项目版本 (project revision)

每次对项目进行编辑保存，就会形成一个项目版本（以下称作“版本”）。一个版本即为一个项目包含完整拓扑和参数的快照（但参数集、方案的值不包含在内）。记录每个版本文件的特征值 (hash)，作为版本的主索引。每个版本 **一经创建，不可更改**。

一个版本记录的信息主要包含：

hash：版本实际内容的特征值，也是数据库的主索引

parent：该版本的父版本 hash，可空

data：系统的参数和拓扑等信息，包含一个防止哈希碰撞的nonce。nonce 可以由用户指定，此时应对 nonce 长度加以要求，防止碰撞产生。

created：版本的创建时间

版本系统不包含权限管理相关的功能。hash 应当足够复杂，避免碰撞的产生。所有可以提供正确hash的用户即被认为拥有了相应版本的访问权限。

注意：这意味着一旦用户将某一个版本发布/共享给他人，该版本的权限将永远无法收回。即使后续删除了分享的权限，也仅限于后续发布的新版本无法被继续使用。事实上，被分享的用户甚至可以将该版本重新保存为自己的命名版本，并进行二次分发。就算是版本管理这里进行了限制，用户也完全可以通过 hash 下载对应的系统文件，并重新上传，该行为实质上无法控制。

7.1 快照版本 (snapshot revision)

基于用户保存的版本。每个版本记录其父版本的key。实现历史记录查询、历史 版本恢复等功能。

7.2 命名版本 (named revision)

给某个版本指定了名称和描述后，就形成了一个命名版本。命名版本有自己的 RID。

一个由项目的历史版本命名产生的命名版本会保存与其相关联项目的关系，并显示在项目的相关页面下。

命名版本也可以不存在对应的项目 RID，如用户直接上传的分离版本、用户删除项目后保留的命名版本均属于此类情况。

通过项目或命名版本的 RID，其他项目可以将该项目/版本作为自己的子系统使用。

7.3 最新版本 (latest revision)

通过项目保存的最新版本 hash，可以在版本库中找到一个项目的最新版本。一个项目的编辑历史由此版本出发，沿父版本向上搜寻直至某一版本不存在父版本。

其他项目可以通过项目 RID 引用一个项目的最新版本。

7.4 分离版本 (detached revision)

满足以下条件的版本：

1. 这不是一个命名版本，或
2. 这不是一个最新版本或其直接或间接父级版本（即不属于任何项目的历史记录）

被称为分离版本。一般由于用户主动上传或删除了对应的系统/命名版本产生。

用户也可以在将项目保存为命名版本时主动选择生成分离版本，此时后台会将对应版本选取不同的 nonce 后保存，以便生成新的 hash。

7.5 关于CLOUDPSS官方元件

CloudPSS 官方元件也以子系统的方式发布，每一个元件有自己的项目 RID。但一般不会发布命名版本，即强制所有用户使用 CloudPSS 的最新版本元件。尽管格式与用户创建的项目一致，这些 RID 实质只是一个标签，在版本库中并不存在对应的版本数据。

7.6 版本的删除

用户无法主动对版本进行删除操作。通过一个定时运行的垃圾回收程序，生成时间距今大于一个指定阈值的分离版本将会被自动删除。

8 计算任务发布流程

8.1 确定需要计算的版本

要发布一个计算任务，首先需要获取其版本 key。

对于用户手动上传的分离版本，API 直接将版本保存进数据库，并返回对应的 hash。用户可以直接使用此 hash 开始计算任务。也可以通过此 hash 建立一个命名版本，以供后续使用。甚至可以将一个 hash 导入为新的项目，进行后续开发。

对于一个项目或命名版本，可以通过 API 获取其对应的 hash。这两者都是拥有自己 RID 的资源，自然权限校验也在该步骤完成。

8.2 启动任务

向调度中台发布任务。任务定义为需要进行的计算任务定义。任务参数为版本 key、参数组、以及该计算任务所需的其他参数，如结果保存用的数据中台 RID。通过生成的任务 RID，可以对任务的运行状况进行监测和管理。

8.3 获取结果

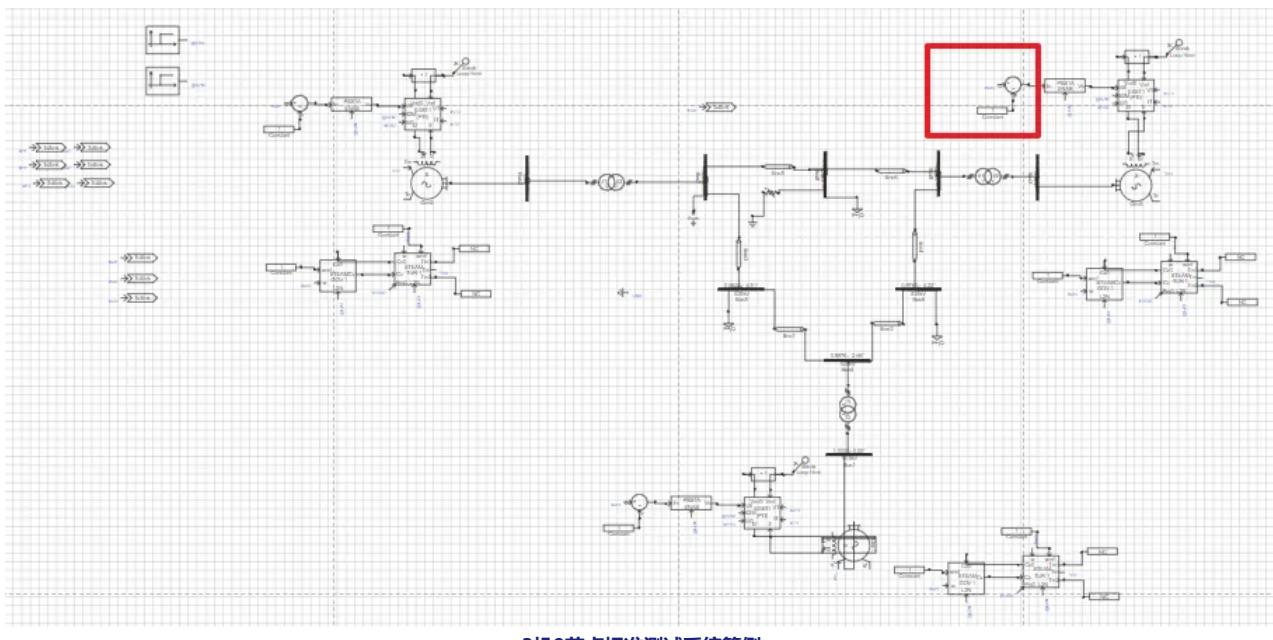
通过数据中台获取计算结果。

CloudPSS SDK 快速入门

本例以通过 Python 脚本对 3 机 9 节点标准测试系统算例进行潮流计算和电磁暂态仿真为例，帮助用户快速入门 CloudPSS SDK 的使用。

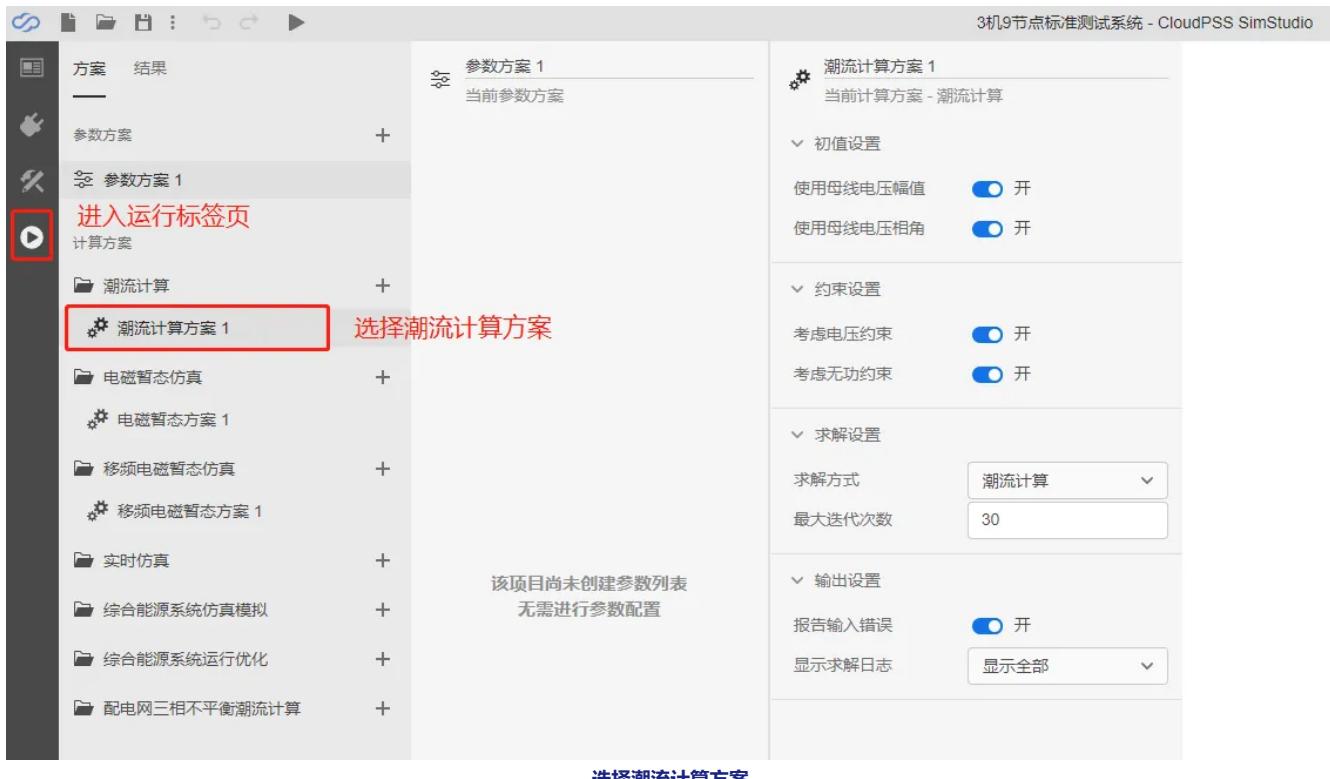
1. 潮流计算内核调用

首先，在 CloudPSS Simstudio 中打开3 机 9 节点标准测试系统算例。



3机9节点标准测试系统算例

点击**运行**标签页，在计算方案中选择默认的**潮流计算方案1**。



选择潮流计算方案

点击**启动任务**运行仿真，在**结果**页面会生成潮流计算结果。

启动潮流计算中

计算任务进入队列，等待运行

*

Power flow case solved

Buses

	Bus	Node	V _m / pu	V _a / deg	P _{gen} / MW	Q _{gen} / MVar	P _{load} / MW	Q _{load} / MVar	P _{shunt} / MW	Q _{shunt} / MVar	P _{res} / MW	Q _{res} / MVar
1	newBus	SyncGen-	1.0000	8.0926	150.0000	10.5512	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	newBus		0.9978	2.7014	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3	newBus		0.9878	0.3239	0.0000	0.0000	100.0000	35.0000	0.0000	0.0000	0.0000	0.0000
4	newBus		1.0042	1.8733	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5	newBus	SyncGen-	1.0000	4.8839	90.0000	-4.7844	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
6	newBus		0.9633	-4.5105	0.0000	0.0000	125.0000	50.0000	0.0000	0.0000	0.0000	0.0000
7	newBus		0.9763	-4.2170	0.0000	0.0000	90.0000	30.0000	0.0000	0.0000	0.0000	0.0000
8	newBus		0.9876	-2.6565	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
9	newBus	SyncGen-	1.0000	0.0000	79.4670	23.3829	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

Branches

	Branch	From bus	P _{ij} / MW	Q _{ij} / MVar	To bus	P _{ji} / MW	Q _{ji} / MVar	P _{loss} / MW	Q _{loss} / MVar
1	TLine_3p-1	newBus_3p-2	72.2887	0.7300	newBus_3p	-71.8382	-12.3705	0.4505	-11.6405
2	TLine_3p-2	newBus_3p-6	-75.7639	-15.3605	newBus_3p	77.7113	-4.3108	1.9473	-19.6714
3	TLine_3p-3	newBus_3p-3	-28.1618	-22.6295	newBus_3p	28.2765	2.8332	0.1147	-19.7963
4	TLine_3p-4	newBus_3p-7	-60.2679	-16.3604	newBus_3p	61.7235	-12.3775	1.4556	-28.7379
5	TLine_3p-5	newBus_3p-6	-49.2361	-34.6394	newBus_3p	49.5714	20.1581	0.3353	-14.4813
6	TLine_3p-6	newBus_3p-8	29.8956	-0.7276	newBus_3p	-29.7321	-13.6396	0.1635	-14.3673
7	newTransfo	newBus_3p-1	150.0000	10.5512	newBus_3p	-150.0000	3.5808	0.0000	14.1321
8	newTransfo	newBus_3p-4	-90.0000	9.5444	newBus_3p	90.0000	-4.7844	0.0000	4.7600
9	newTransfo	newBus_3p-8	-79.4670	-19.4305	newBus_3p	79.4670	23.3829	0.0000	3.9524

潮流计算结果

若要分析算例中 **Gen2** 发电机的有功出力与 **Bus2** 的电压相角之间的关系，常规方法是手动修改参数执行多次仿真，绘制出有功出力与电压相角的描点图。这个方法操作复杂且效率低下。借助 CloudPSS SDK，利用 Python 脚本修改参数，批量调用潮流计算内核，可以快速完成上述功能。

示例代码

配置好 **Python** 开发环境，输入以下脚本。

PYTHON

```
import sys,os
import cloudpss
import json
import time

if __name__ == '__main__':
    cloudpss.setToken('{token}')
    os.environ['CLOUDPSS_API_URL'] = 'https://cloudpss.net/'

    # 获取指定 rid 的项目
    model = cloudpss.Model.fetch('model/songyankan/3_Gen_9_Bus')

    # 修改 Gen2 发电机有功功率
    comp = model.getComponentByKey('canvas_0_757')
    print(comp.args)
    comp.args['pf_P'] = '180'

    # 启动计算任务
    config = model.configs[0] # 若未设置，则默认用model的第一个config (参数方案)
    job = model.jobs[0] # 若未设置，则默认用model的第一个job (计算方案)
    runner = model.run(job,config)
    while not runner.status():
```

```

logs = runner.result.getLogs()
for log in logs:
    print(log)
time.sleep(1)
print('end')

# 打印结果
print(runner.result.getBranches())
print(runner.result.getBuses())

```

地址与Token替换

PYTHON

```
os.environ['CLOUDPSS_API_URL'] = 'https://cloudpss.net/'
```

使用时需要将 `'https://cloudpss.net/'` 替换为用户当前使用的平台网址地址。

PYTHON

```

cloud-
pss.setToken('eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJhZG1pbjIsInNjb3BlcyI6WyJ1bm
ub3duI10sInR5cGUiOiJTREsiLCJleHAiOjE2NTg1NjgzNDYsImhdCI6MTYyNjk0MTQ1MX0.hDRBisqpd2bXzg5HZVoTVnxw2Gm0AihY5
HHALNpFs_gcLCL45Xt8rYKrCUq3CZKq-iM-
mYfQvPgWIn2B_QCmUezHtUuRQw_nmBBLb5NMpIAiFJJiBFDGjBvzwBAINCbBFnr8zDxUvwHZMoAb3ed9VNJDqI_CTzB8Q3udTb10-TXs')

```

同时需要申请和修改 Token，替换 `cloudpss.setToken` 后的内容。Token 的申请和注销详见 [setToken用户认证](#) 帮助文档。

指定算例

PYTHON

```
model = cloudpss.Model.fetch('model/UserName/ModelRID')
```

使用时需要将 `model/UserName/ModelRID` 替换为用户当前使用的算例。如在当前算例中，`model/songyankan/3_Gen_9_Bus` 表示用户 `songyankan` 下的 `3_Gen_9_Bus` 算例。

例如：若算例的 URL 为 `https://cloudpss.net/model/user/example#`，则代码中对应部分应替换为 `model/user/example`。

定位元件

PYTHON

```
comp = model.getComponentByKey('canvas_0_757')
```

CloudPSS SDK 中，提供了 `getComponentByKey` 函数，用户可以通过元件的 `Key` 来定位算例中的某个元件。在 SimStudio 中，选中算例中的发电机 `Gen2`，此时浏览器地址栏变为

`https://cloudpss.net/model/songyankan/3_Gen_9_Bus#/design/diagram/cells/canvas_0_757`，`canvas_0_757` 即为发电机 `Gen2` 的 Key。在每个算例中，元件 Key 是唯一的。

获取或修改参数 在定位元件后，即可获取并任意修改该元件的参数 `args`。此处，通过输出元件 `comp` 的全部参数 `args` 可知，元件参数是以字典的形式存储的。

PYTHON

```
print(comp.args)
{'Name': 'Gen2', 'P': '=4', 'Smva': '=325', 'V': '$Bus_2_Vbase / sqrt(3)', 'freq': '=50', 'R0': '=10000',
'ParamType': '0', 'ModelType': '0', 'Rs': '=0.000301', 'Xls': '=0', 'Xq': '=0.283875', 'Xd': '=0.283875',
'Rfd': '=0.000117219', 'Xlfd': '=0.047921256', 'Rkd': '=0.009822918', 'Xlkd': '=0.097868236', 'Rkqg':
'=50000', 'Xlkqg': '=50000', 'RkqQ': '=0.005334267', 'XlkqQ': '=0.059027851', 'Rs_2': '=0.000301',
'Xls_2': '=0', 'Xd_2': '=0.283875', 'Xdp_2': '=0.041', 'Xdpp_2': '=0.028895', 'Xq_2': '=0.283875',
'Xqp_2': '=0.056603', 'Xqpp_2': '=0.028895', 'Td0p_2': '=9.01', 'Td0pp_2': '=0.045', 'Tq0p_2': '=0.956',
'Tq0pp_2': '=0.069', 'Control': '1', 'Tj': '=5', 'Dm': '=0', 'StartupType': '4', 'RampingTime': '=0.06',
'V_mag': 1, 'V_ph': 8.092582389805873, 'AP': 150, 'RP': 10.551261791343547, 's2m': '@S2M', 'l2n': '@L2N',
'BusType': '1', 'pf_P': '=150', 'pf_Q': '=100', 'pf_V': '=1', 'pf_Theta': '=0', 'pf_Vmin': '=0.001',
'pf_Vmax': '=10', 'pf_Qmin': '-200', 'pf_Qmax': '=200', 's2m_o': '#initEx2', 'l2n_o': '#initGv2',
'Ef0_o': '#Ef02', 'Tm0_o': '#Tm02', 'wr_o': '#wr2', 'theta_o': '', 'loadangle_o': '', 'loadangle_so': '',
'VT_o': '#VT2', 'IT_o': '#IT2', 'PT_o': '#P2', 'QT_o': '#Q2', 'IT_inst': ''}
```

其中，`'pf_P': '=150'` 即代表 **Gen2** 元件 **Power Flow Data** 参数组中 **Injected Active Power** 的值，表示此 PV 节点输入系统的有功功率为 150MW。此处的赋值支持数字和表达式字符串。

PYTHON

```
comp.args['pf_P'] = '180'
comp.args['pf_P'] = '=180'
comp.args['pf_P'] = 180
```

通过设置此 `pf_P` 参数即可修改 **Gen2**潮流计算中的注入有功功率。上述 3 条语句执行的效果相同。

运行

PYTHON

```
config = model.configs[0]
job = model.jobs[0]
runner = model.run(job, config)
```

上述语句中的 `model.configs` 即为 SimStudio 中相关算例的全部参数方案（从上到下从0开始编号），`model.jobs` 为全部计算方案（从上到下从 0 开始编号）。通过选择指定的参数方案和计算方案，执行 `model.run(job, config)`，即可生成并执行以参数方案 `config` 和计算方案 `job` 的相应计算任务。

运行过程中，不断请求 `runner.status` 即可获得当前任务的计算状态（`False` 对应进行中，`True` 对应运行结束）。

结果输出

潮流计算结果均保存在 `runner.result` 中。用户可查看 `result` 类的接口说明文档获取更多帮助。

PYTHON

```
print(runner.result.getBranches())
print(runner.result.getBuses())
```

通过上述两条语句可得到母线（Buses）和线路（Branches）的潮流计算，其与 SimStudio 的结果页面展示的结果一致。

若需要输出表格中的某些单元格的数值，使用 Python 的切片操作获取即可。例如：

PYTHON

```
Busesresult = runner.result.getBuses()
Vavalue = Busesresult[0]['data']['columns'][3]['data'][0]
print(Vavalue)
```

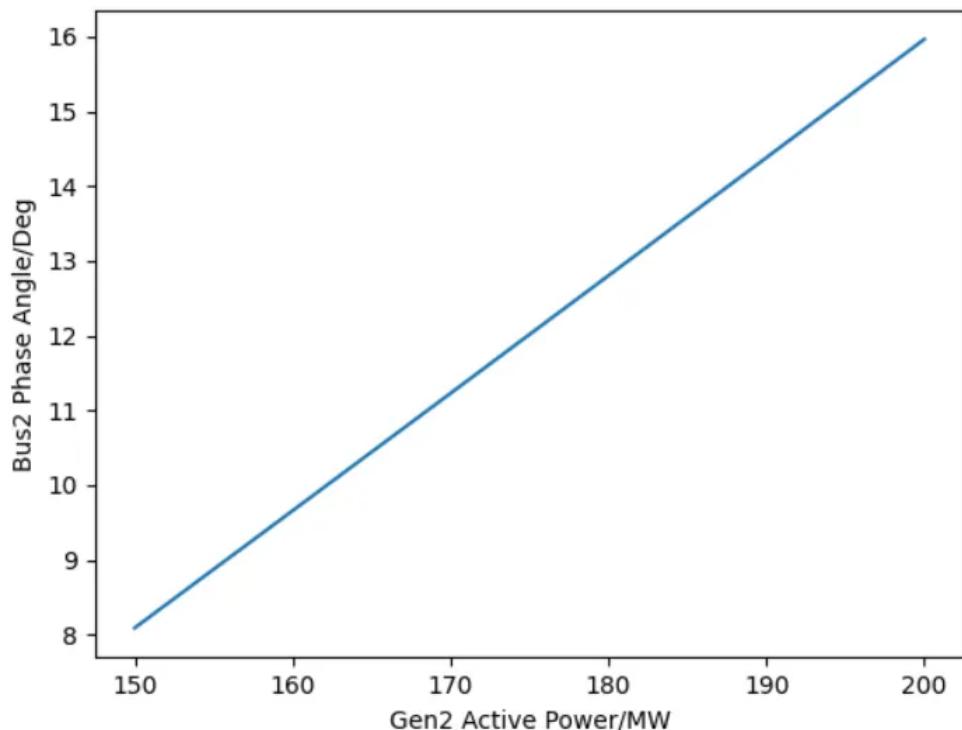
此时输出的数值即为此算例在 **Gen2** 输入功率为 **180MW** 时对应的 **Bus2** 的相角。

Buses									
	Bus	Node	V _m / pu	V _a / deg	P _{gen} / MW	Q _{gen} / MVar	P _{load} / MW	Q _{load} / MVar	P _{shunt} / MW
1	newBus	SyncGen-	1.0000	12.8027	180.0000	16.5118	0.0000	0.0000	0.00
2	newBus		0.9961	6.3176	0.0000	0.0000	0.0000	0.0000	0.00
3	newBus		0.9860	3.5666	0.0000	0.0000	100.0000	35.0000	0.00
4	newBus		1.0031	4.4914	0.0000	0.0000	0.0000	0.0000	0.00
5	newBus	SyncGen-	1.0000	7.5053	90.0000	-2.8576	0.0000	0.0000	0.00
6	newBus		0.9588	-2.7967	0.0000	0.0000	125.0000	50.0000	0.00
7	newBus		0.9729	-2.7024	0.0000	0.0000	90.0000	30.0000	0.00
8	newBus		0.9845	-1.7091	0.0000	0.0000	0.0000	0.0000	0.00
9	newBus	SvncGen-	1.0000	0.0000	50.9783	27.6336	0.0000	0.0000	0.00

输出单元格数值

批量运行潮流

若使用 **for循环** 反复调用以上操作，在 150MW 到 200MW 之间改变 **Gen2** 的有功出力，可以得到对应母线 **Bus2** 的相角的变化曲线。



批量运行绘制曲线

2. 电磁暂态计算内核调用

与调用潮流计算内核的方式相同，调用电磁暂态仿真计算内核时，需要指定计算方案（Job）为电磁暂态计算方案。在该案例中，`model.jobs[1]` 即为电磁暂态计算方案。通过修改故障元件的故障结束时间，来执行不同的电磁暂态仿真案例。

示例代码

配置好 **Python** 开发环境，输入以下脚本。

```
import sys,os
import cloudpss
import json
import time
```

PYTHON

```
if __name__ == '__main__':
    cloudpss.setToken('{token}')
    os.environ['CLOUDPSS_API_URL'] = 'https://cloudpss.net/'

    # 获取指定 rid 的项目
    model = cloudpss.Model.fetch('model/songyankan/3_Gen_9_Bus')

    # 修改故障元件的故障结束时间
    comp = model.getComponentByKey('canvas_0_1150')
    print(comp.args)
    comp.args['fe'] = 3.2

    # 启动计算任务
    config = model.configs[0] # 若未设置，则默认用 model 的第一个 config (参数方案)
    job = model.jobs[1] # 若未设置，则默认用 model 的第一个 job (计算方案)，此处选择 jobs[1]，为电磁暂态仿真任务
    runner = model.run(job,config)
    while not runner.status():
        logs = runner.result.getLogs()
        for log in logs:
            print(log)
        time.sleep(1)
    print('end')

    # 打印结果
    plots = runner.result.getPlots() #获取全部输出通道
```

结果输出

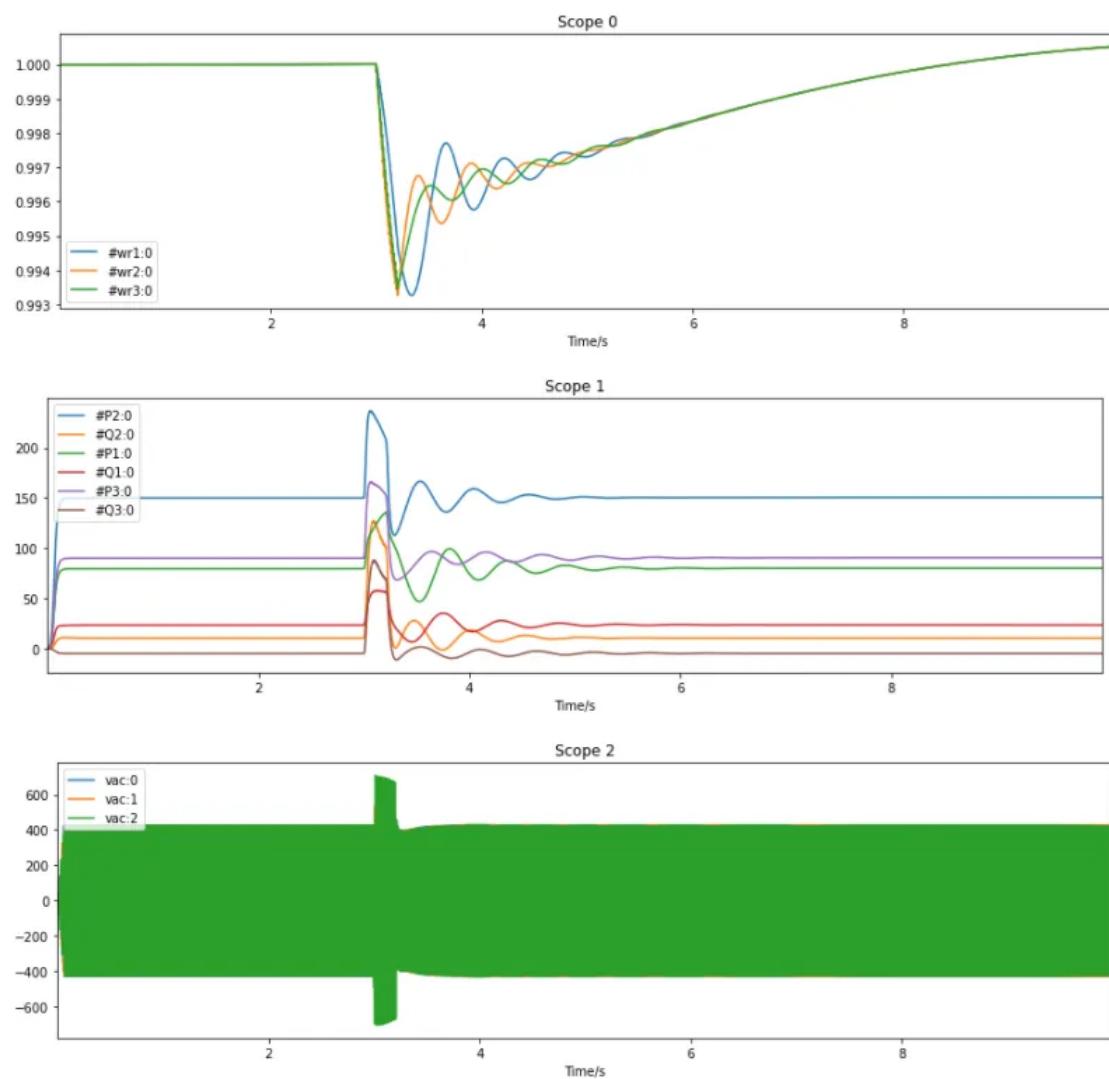
电磁暂态计算结果均保存在 `runner.result` 中。用户可查看 `result` 类的接口说明文档获取更多帮助。此处可以使用 `matplotlib` 库或 `plotly` 库绘制结果曲线。

使用 matplotlib 绘制曲线

PYTHON

```
import matplotlib.pyplot as plt
for it in range(0,len(plots)):
    legend = runner.result.getPlotChannelNames(it)
    print('示波器分组',it,':',legend)
    plt.figure('示波器分组'+str(it),figsize = (15,4))
    for jt in range(0,len(legend)):
        plot = runner.result.getPlotChannelData(it,legend[jt])
        plt.plot(plot['x'],plot['y'])
        plt.title('Scope '+str(it))
        plt.xlim([min(plot['x']),max(plot['x'])])
        plt.legend(legend)
        plt.xlabel('Time/s')
```

在 Jupyter Notebook 中的效果如下。

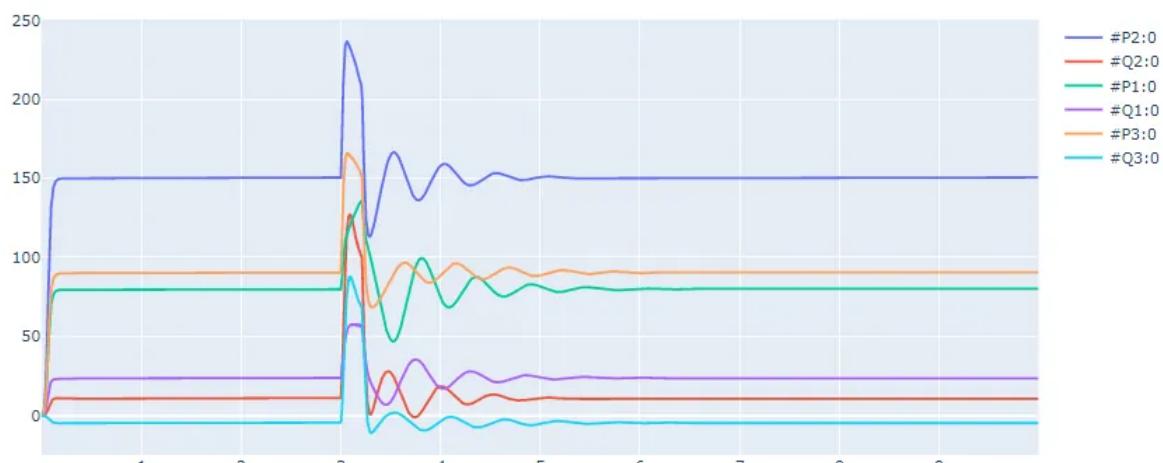


使用 plotly 绘制曲线

PYTHON

```
import plotly.graph_objects as go
for i in range(len(plots)):
    fig = go.Figure()
    channels= runner.result.getPlotChannelNames(i)
    for val in channels:
        channel=runner.result.getPlotChannelData(i,val)
        fig.add_trace(go.Scatter(channel))
    fig.show()
```

在 Jupyter Notebook 中的效果如下。



CloudPSS 接口文档

- Token 申请与设置
- Model 类
- Runner 类
- Function 类
- FunctionJob 类
- Project类

Token 申请与设置

申请token

首先点击 **主页** 标签页进入 CloudPSS 主页，然后点击左下角的 **设置** 按钮进入设置页面。

The screenshot shows the CloudPSS homepage with various application icons and news sections. A red box highlights the '点击设置' (Click Settings) button located at the bottom left of the main content area.

设置页面

点击 **SDK Token 申请** 标签进入 Token 申请界面，并选择 **时效**。

The screenshot shows the 'SDK Token Application' interface. A red box highlights the 'SDK Token Application' button under '个人信息设置' (Personal Information Settings). Another red box highlights the 'Duration' dropdown menu under '时效' (Validity Period), which is currently set to '1天' (1 day). The dropdown also includes options for 3 days, 30 days, 60 days, 90 days, 180 days, and 360 days. A blue arrow points to the '主销' (Main Sales) option in the dropdown menu.

基本设置界面

然后点击 **生成** 按键进行 Token 申请。



点击**复制**按钮将所申请的Token复制到剪贴板。



Token将不会保存在**用户中心** -> **Token申请**页面。离开页面后数据将丢失。用户需将Token记录下来，防止丢失。如不慎丢失，需重新申请。

设置token

params: token token

[PYHTON] [SETTOKEN]

```
cloudpss.setToken('{token}')
```

Model 类

```
class cloudpss.model.model.Model(model: dict = {})
```

CloudPSS **工程类**，用于处理加载后的工程数据

实例变量说明：

rid: 项目的 rid

name: 项目的名称

description: 项目的描述

revision: 当前项目的版本信息

configs: 当前项目的所有参数方案

jobs: 当前项目的所有计算方案

context: 当前项目的上下文相关信息

addConfig(config)

将**参数方案**添加到工程中

Params config: 参数方案dict

PYTHON

```
config = model.createConfig('my config')
model.addConfig(config)
```

addJob(job: dict)

将**计算方案**添加到工程中

Params job: 计算方案 dict

PYTHON

```
job = model.createJob('emtp', 'emtp job')
model.addJob(job)
```

static create(project)

新建项目

Params: project 项目

返回：保存成功/保存失败

PYTHON

```
Model.create(model)
保存成功
```

createConfig(name)

创建一个参数方案，根据项目的第一参数方案生成一个方案。

创建出的方案默认不加入到项目中，需要加入请调用 `addConfig :params name:参数方案名称`

返回: 返回一个参数方案 dict

PYTHON

```
job = model.createConfig('my config')  
参数方案
```

createJob(jobType: str, name)

创建一个计算方案, 创建出的方案默认不加入到项目中，需要加入请调用 `addJob`

Params jobType: 方案类型，包括电磁暂态仿真方案emtp、移频电磁暂态仿真方案sfemt、潮流计算方案powerFlow

返回: 返回一个指定类型的计算方案

PYTHON

```
model.createJob('emtp', 'emtp job')  
计算方案
```

static dump(model, file)

下载项目文件

Params model: 项目

Params file: 文件路径

返回: 无

PYTHON

```
Model.dump(model, file)
```

static fetch(rid)

获取项目

Params rid: 项目 rid

返回: 返回一个项目实例

PYTHON

```
model=Model.fetch('model/Demo/test')
```

static fetchMany(name=None, pageSize=10, pageOffset=0)

获取用户可以运行的项目 **列表**

Params name: 查询名称，模糊查询

Params pageSize: 分页大小

Params pageOffset: 分页开始位置

返回: 按分页信息返回项目列表

PYTHON

```
data=Model.fetchMany()  
[  
    {'rid': 'model/admin/share-test', 'name': '1234', 'description': '1234'}  
    ...  
]
```

fetchTopology(implementType=None, config=None, maximumDepth=None)

通过项目信息，[获取](#)当前项目对应的[拓扑数据](#)

Params implementType: 实现类型

Params config: config 项目参数, 不指定将使用算例保存时选中的参数方案

Params maximumDepth: 最大递归深度, 用于自定义项目中使用 diagram 实现元件展开情况

返回: 一个拓扑实例

PYTHON

```
topology=model.fetchTopology()  
topology=model.fetchTopology(implementType='powerFlow',config=config) # 获取潮流实现的拓扑数据  
topology=model.fetchTopology(maximumDepth=2) # 获取仅展开2层的拓扑数据
```

getAllComponents()

[获取](#)实现

返回: 所有元件信息

PYTHON

```
model.getAllComponents()  
{  
    'canvas_0_2': Component 实例  
}
```

getComponentByKey(componentKey: str)

通过元件的[key](#)获取对应的元件

Params key: key 元件的key

Return: Component 实例

PYTHON

```
model.getComponentByKey('canvas_0_757')  
Component 实例
```

getComponentsByRid(rid: str)

通过指定[元件类型](#)获取元件

Params str: 元件类型

返回: 按照元件的 rid 过滤后的 dict<>

PYTHON

```
model.getComponentsByRid('model/CloudPSS/newInductorRouter')
{
    'canvas_0_2': Component 实例
}
```

getModelConfig(name)

获取 指定名称的 参数方案

Params name: 参数方案名称

返回: 同名的方案数组

PYTHON

```
model.getModelConfig('参数方案 1')
```

getModelJob(name)

获取 指定名称的 计算方案

Params Name: 参数名称

返回: 返回同名计算方案数组

PYTHON

```
model.getModelJob('电磁暂态方案 1')
```

static load(filePath)

加载 本地项目文件

Params file: 文件目录

返回: 返回一个项目实例

PYTHON

```
model = Model.load('filePath')
```

run(job=None, config=None, name=None, **kwargs)

调用 仿真

Params job: 调用仿真时使用的计算方案，不指定将使用算例保存时选中的计算方案

Params config: 调用仿真时使用的参数方案，不指定将使用算例保存时选中的参数方案

Params name: 任务名称，为空时使用项目的参数方案名称和计算方案名称

返回: 返回一个运行实例

```
runner=model.run(job,config,'')
runner
```

save(key=None)

保存/创建 项目

key不为空时如果远程存在相同的资源名称时将覆盖远程项目。**key为空**时如果项目rid不存在则抛异常，需要重新设置key。如果保存时，当前用户不是该项目的拥有者时，将重新创建项目，重建项目时如果参数的key为空将使用当前项目的key作为资源的key，当资源的key和远程冲突时保存失败

Params: project 项目

Params: key 资源id的唯一标识符，

返回: 保存成功/保存失败

```
model.save(model)
model.save(model,'newKey') # 另存为新的项目
```

toJSON()

类对象**序列化** 为 dict :return: dict

static update(model)

更新 项目

Params: model 项目

返回: 保存成功/保存失败

```
Model.update(model)
```

添加每个内核的运行入口用于运行指定内核程序，如下

电磁暂态内核运行入口

runEMT(config=None,jobName=None,configName=None)

运行 **emtp** 内核，如果当前 **model** 没有创建 **emtp Job** 时报错，默认使用第一个计算方案，进行仿真。

Params config: 参数方案，可选，默认使用保存时选中的参数方案

Params jobName: 计算方案名称，可选，默认使用第一个计算方案，如果同名使用最靠前一个

Params configName: 参数方案名称，可选，如果同时使用config和configName 以config 为主，同名使用最靠前一个

返回: runner Runner[EMTResult]

移频电磁暂态内核运行入口

runSFEMT(job=None,config=None)

运行移频电磁暂态内核，如果当前 model 没有创建 Job 时报错，默认使用第一个计算方案，进行仿真。

Params job: 计算方案名称，可选，字符串类型或者字典类型,默认使用第一个计算方案，如果同名使用最靠前一个

Params config: 参数方案，可选，字符串类型或者字典类型,默认使用保存时选中的参数方案

返回: runner Runner[EMTResult]

潮流计算内核运行入口

runPowerFlow(job=None,config=None)

运行潮流内核，如果当前 model 没有创建 Job 时报错， 默认使用第一个计算方案，进行仿真。

Params job: 计算方案名称，可选，字符串类型或者字典类型,默认使用第一个计算方案，如果同名使用最靠前一个

Params config: 参数方案，可选，字符串类型或者字典类型,默认使用保存时选中的参数方案

返回: runner Runner[PowerFlowResult]

配电网三相不平衡潮流内核运行入口

runThreePhasePowerFlow(job=None,config=None)

运行配电网三相不平衡潮流内核，如果当前 model 没有创建 Job 时报错， 默认使用第一个计算方案，进行仿真。

Params job: 计算方案名称，可选，字符串类型或者字典类型,默认使用第一个计算方案，如果同名使用最靠前一个

Params config: 参数方案，可选，字符串类型或者字典类型,默认使用保存时选中的参数方案

返回: runner Runner[PowerFlowResult]

ModelRevision 类

```
class cloudpss.model.revision.ModelRevision(revision: dict = {})
```

表示一个项目的 **版本数据**

实例变量说明：

implements: 项目当前版本的实现数据

parameters: 项目当前版本的参数结果

pins: 项目当前版本的引脚信息

documentation: 项目当前版本的文档信息

static create(revision, parentHash=None)

创建一个 **新版本**

Params revision: 版本数据

返回: 项目版本hash

PYTHON

```
ModelRevision.create(model.revision)
{hash:'4043acb9ce0c6174be65573c0380415bc48186c74a459f88865313743230c'}
```

fetchTopology(implementType, config, maximumDepth)

获取当前项目版本的 **拓扑数据**

Params implementType: 实现类型

Params config: 项目参数

Params maximumDepth: 最大递归深度，用于自定义项目中使用 diagram 实现元件展开情况

返回: 一个拓扑实例

PYTHON

```
topology=revision.fetchTopology()
topology=revision.fetchTopology(implementType='powerFlow',config=config) # 获取潮流实现的拓扑数据
topology=revision.fetchTopology(maximumDepth=2) # 获取仅展开 2 层的拓扑数据
```

getImplements()

获取当前版本的 **实现**

返回: 实现实例

PYTHON

```
revision.getImplements()
```

run(job, config, name=None, rid='', **kwargs)

运行某个[指定版本](#)的项目

Params job: 调用仿真时使用的计算方案, 为空时使用项目的一个计算方案

Params config: 调用仿真时使用的参数方案, 为空时使用项目的一个参数方案

Params name: 任务名称, 为空时使用项目的参数方案名称和计算方案名称

Params rid: 项目 rid, 可为空

返回: 返回一个运行实例

PYTHON

```
revision.run(revision, job, config, '')
```

toJSON()

类对象[序列化](#)为dict :return: dict

ModelImplement类

ModelImplement

```
class cloudpss.model.implements.ModelImplement(implements: dict = {})
```

实现类

getDiagram()

获取 拓扑实现，不存在返回空

返回: 示意图实例

PYTHON

```
implement.getDiagram()
```

toJSON()

类对象 序列化 为 dict :return: dict

PYTHON

```
implement.toJSON()
```

1 diagram

```
class cloudpss.model.implements.diagram.DiagramImplement(diagram: dict = {})
```

拓扑 实现

getAllComponents()

获取 所有元件

返回: dict

PYTHON

```
diagram.getAllComponents()
```

toJSON()

类对象 序列化 为dict :return: dict

PYTHON

```
diagram.toJSON()
```

2 component

```
class cloudpss.model.implements.component.Component(diagram: dict = {})
```

元件类

实例变量说明：

definition: 元件定义，连接线没有definition

args: 元件参数数据，连接线没有参数数据

pins: 元件引脚数据，连接线没有引脚数据

shapes diagram-component: 表示元件

diagram-edge: 表示连接线

toJSON()

类对象 **序列化** 为dict: return:dict

PYTHON

```
comp.toJSON()
```

ModelTopology 类

class cloudpss.model.topology.ModelTopology(topology: dict = {})

项目 **拓扑** 类，通过该类的静态方法fetch获取一个拓扑实例

实例变量说明:

components: 摊平后的拓扑元件，参数和引脚不再保留表达式的形式，如果元件为拓扑实现，并有读取权限时将被展开

mappings: 拓扑分析后的一些映射数据

static dump(topology, filePath, indent=None)

以JSON格式 **保存** 数据到指定文件

Params: topology 拓扑实例

Params: file 文件路径

Params: indent json 格式缩进

static fetch(hash, implementType, config, maximumDepth=None)

获取 拓扑

Params: hash

Params: implementType 实现类型

Params: config 参数方案

Params: maximumDepth 最大递归深度，用于自定义项目中使用 diagram 实现元件展开情况

: return: 拓扑实例 拓扑实例

PYTHON

```
data = ModelTopology.fetch('','emtp',{})
```

toJSON()

将类 **转换** 为 dict 数据

Runner 类

```
class cloudpss.runner.runner.Runner(taskId, name, job, config, revision,  
modelRid, **kwargs)
```

```
static create(revisionHash, job, config, name=None, rid='', **kwargs)
```

创建一个运行任务

Params revision: 项目版本号

Params job: 调用仿真时使用的计算方案，为空时使用项目的一个计算方案

Params config: 调用仿真时使用的参数方案，为空时使用项目的一个参数方案

Params name: 任务名称，为空时使用项目的参数方案名称和计算方案名称

Params rid: 项目rid，可为空

返回: 返回一个运行实例

PYTHON

```
runner = Runner.runRevision(revision, job, config, '')
```

status()

运行 **状态** :return: 返回运行状态0/1/-1。1表示正常结束,0表示运行中， -1表示数据接收异常

PYTHON

```
runner.status()
```

Storage 类

```
class cloudpss.runner.storage.Storage(key, name, job, config, revision, model,  
messages=None, finished=None, **kwargs)
```

消息 **存储** 类，用于存储接收到的消息数据，并提供数据保存、加载两个数据本地化函数。 提供通过数据类型获取数据，通过数据位置获取数据、和获取当前数据长度的函数

static dump(result, file)

保存 结果到本地文件

Params **file**: 保存文件的目录

PYTHON

```
Result.dump(result,file)  
{...}
```

getMessage(index)

获取 指定位置的 **消息数据**

Params **index**: 数据的位置信息

返回: 消息数据

PYTHON

```
message= db.getMessage(1)
```

getMessageLength()

获取 指定消息的 **长度**

返回: 数据长度

PYTHON

```
length=db.getMessageLength()
```

getMessagesByKey(key)

获取 指定 key 的 **消息数据**

Params **key**: 数据 key

返回: 对应 key 的数据数组

PYTHON

```
message= db.getMessagesByKey('log')
```

getMessagesByType(type)

获取 指定类型的 消息数据

Params type: 数据类型

返回: 对应类型的数据数组

PYTHON

```
message= db.getMessagesByType('log')
```

static load(file)

加载 本地结果文件

Params: file 文件目录

返回: 返回一个项目实例

PYTHON

```
result = Result.load('C:\Users\dps-dm\cloudpss-sdk\result\424111.cjob')
```

storeMessage(message)

消息 存储 db.storeMessage(message)

Params message: 需要存储的消息

Receiver 类

```
class cloudpss.runner.receiver.Receiver(taskId, db, url: Optional[str] = None,  
**kwargs)
```

socket 接收服务

close(ws)

connect()

on_close(ws)

on_error(ws, error)

on_message(ws, message)

消息接收处理

Params ws: socket 实例

Params message: 接收到的数据

heartbeat ws 心跳服务 **testData**

on_open(ws)

status()

Result 类

class cloudpss.runner.result.EMTResult(*args, **kwargs)

电磁暂态 结果处理类，继承 Result

提供快捷 plot 数据的接口函数，获取到的 plot 数据为合并后的数据格式，不再是接收时分段的数据

该类只提供 EMT 仿真使用

getPlot(index: int)

获取 指定序号的 **数据分组**

Params index: 图表位置

PYTHON

```
result.getPlot(0)  
{...}
```

getPlotChannelData(index, channelName)

获取 一组输出分组下指定通道名称的 **数据**

Params index: 输出通道位置

Params channelName: 输出通道名称

返回: 通道数据, 一个trace数据

PYTHON

```
channel= result.getPlotChannelData(0, '') {...}
```

getPlotChannelNames(index)

获取 一组输出分组下的所有 **通道名称**

Params index: 输出通道位置

返回: 通道名称数组

PYTHON

```
names= result.getPlotChannelNames(0) []
```

getPlots()

获取 所有的 plots 数据

PYTHON

```
result.getPlots()  
{...}
```

class cloudpss.runner.result.IESResult(*args, **kwargs)

综合能源 结果处理类，继承 Result

提供快捷 plot 数据的接口函数，获取到的 plot 数据为合并后的数据格式，不再是接收时分段的数据

该类只提供 IES 仿真使用

getPlotData(compID, labelName, traceName='all', index=-1)

获取 元件 ID 为 compID 的元件，对应标签为 labelName、图例名称为 traceName 的 plot 数据的第 index 项

PYTHON

```
result.getPlots(compID, labelName, traceName, index)  
{...}
```

getSankey(index)

获取 第 index 个桑基图数据

PYTHON

```
result.getSankey(index)  
{...}
```

getSankeyNum()

获取 桑基图数据序列的长度

PYTHON

```
result.getSankeyNum()
```

class cloudpss.runner.result.PowerFlowResult(*args, **kwargs)

潮流 结果处理类，继承 Result

提供快速获取 buses 和 branches 的接口，并提供潮流写入项目的接口

该类只提供潮流仿真时使用

getBranches()

获取 潮流结果 branches 数据

PYTHON

```
channel = result.getBranches()  
[...]
```

getBuses()

获取 潮流结果 buses 数据

PYTHON

```
channel= result.getBuses()  
[...]
```

powerFlowModify(project)

潮流数据 写入 model

PYTHON

```
channel= result.powerFlowModify(model)
```

class cloudpss.runner.result.Result(db)

结果处理 类，从消息存储库中获取数据，并进行简单的整理

可迭代器，迭代时按接收顺序返回数据

PYTHON

```
for data in result:  
    print(data)
```

也可以从类的 db 变量，获取数据存储类实例进行操作

static dump(result, file)

保存 结果到本地文件

Params: file 保存文件的目录

PYTHON

```
Result.dump(file)  
{...}
```

getLogs()

获取 当前任务的 日志

PYTHON

```
logs= result.getLogs() {...}
```

getMessagesByType(type)

获取 指定类型的 消息数据

PYTHON

```
message= result.getMessagesByType('log')
```

classmethod load(filePath)

加载本地结果文件

Params file: 文件目录

返回: 返回一个结果实例

PYTHON

```
result = Result.load('C:\Users\dps-dm\cloudpss-sdk\result\424111.cjob')
```

modify(data, model)

通过指定消息修改算例文件

Params data: 消息字典 {}

Params model: 项目

PYTHON

```
message= result.modify(data,model)
```

Function 类

```
class cloudpss.function.function.Function
```

FunctionJob 类

class cloudpss.function.job.Args

参数类

class cloudpss.function.job.Job(kwargs)**

Function Job

abort(data)

发送中止消息

Params **data:** 消息数据

PYTHON

```
job.abort({})
```

property args

container(data, key=None, verb=None)

发送消息组 :params key: [图标索引](#) key :params verb: [操作类型](#) , 可选 'create' | 'replace' | 'append' | 'prepend' :params data: 分组数据

PYTHON

```
{
  layout?:
    | {
        /** 浮窗显示, title 为 anchor (如: `/design/diagram/cells/canvas_0_16`), 用于决定显示的内容 */
        type: 'float';
    }
    | {
        /** tab 页方式显示, title 为 tab 的标题 */
        type: 'tabs';
        position: 'left' | 'right' | 'top' | 'bottom';
    }
    | {
        /** grid 布局显示, title 为 grid-area 名称 */
        type: 'grid';
        /** CSS grid 属性, 如 'item1 . item2' 1fr 'item1 item3 item4' 1fr / 1fr auto 2fr */
        grid: string;
    };
  items: Array<{
    /** 根据 `layout.type` 具有不同的功能 */
    title: string;
    /** 对应 key 数据不存在时显示 */
    placeholder: string;
    /** placeholder 是否使用 html */
    html: boolean;
    /** 获取用于填充的数据的方式 */
    query?:
      | {
          /** 查找指定 key 的消息填充 */
      }
  }>;
}
```

```

    type: 'message';
    key: string;
    /** 当此属性存在时，如未找到指定消息，向调用方发送该 payload 进行查询 */
    signal?: MessagePayload;
}
| {
    /** 通用 web api */
    type: 'http';
    method: string;
    url: string;
    headers: Record<string, string>;
    body: unknown;
    /** 从响应中获取消息的方法 */
    picker?:
        | string[] // 从响应体的路径 pick (lodash.get)
        | ExpressionSource<unknown>; // 通过表达式获取 (可用对象 $req, $res)
}
| {
    /** 从对象存储获取指定对象填充 */
    type: 'object-storage';
    hash: string;
}
| {
    /** 从对象存储获取指定对象填充 */
    type: 'object-storage';
    hash: string;
};

};

}

```

PYTHON

```
job.container(data, "key-c")
```

static current()

获取当前任务 Returns: Job: 任务类

feedDog()

通知看门狗，当前程序还在运行，当程序 30s 内没有输出时，执行器将直接 kill 运行脚本

gridContainer(items, grid=None, key=None, verb=None)

发送类型为 grid 的消息组

Params items: 需要关联的其他图表消息 {'title':标题, 'placeholder': 默认显示内容, key: 需要关联的图表}

Params tabsPosition: 操作类型，可选 'top' | 'bottom'

Params key: 图标索引 key

Params verb: 操作类型，可选 'create' | 'replace' | 'append' | 'prepend'

PYTHON

```
job.abort({})
```

static loadArgs()

加载当前任务参数 Returns: dict: 任务参数

message(msg, level='info', key=None, verb='create')

发送日志信息

Params msg: str 消息内容

Params level: 消息类型 默认 info 可选: 'critical' | 'error' | 'warning' | 'info' | 'verbose' | 'debug'

Params key: str 目标, key 相同的数据会往同一个目标写入数据

Params verb: 操作类型, 可选 'create' | 'replace' | 'append' | 'prepend'

PYTHON

```
job.message('info message')
job.message('error message', level='error')
```

在同一个目标显示消息

PYTHON

```
job.message('message1', key='log-1')
job.message('message2', key='log-1', verb='append')
```

替换目标数据

PYTHON

```
job.message('message1', key='log-1')
job.message('message2', key='log-1', verb='replace')
```

plot(key, traces, verb='append', layout={}, **kwargs)

发送绘图消息

Params key: 目标, key 相同数据将显示到同一个图表上

Params traces: Array 曲线数据组

PYTHON

```
[{
  'name': str, 列名称
  'type': 'text' | 'html' | 'number', 列类型
  'data': unknown[] 列数据
}]
```

Params title: str 图表标题

Params xAxis: dict 坐标轴设置

Params yAxis: dict 坐标轴设置

```
{
    'title':str, 轴标题 'type' : 'linear' | 'log' | 'date',
    轴类型 'range': [number, number] | 'auto' 显示范围
}
```

Params verb: str 消息内容 可选: 'create' | 'replace' | 'append' |'prepend' | 'update'

```
job.plot('plot-1',[{'name':'t1','type':'scatter','x':[1,2],'y':[3,4]}])
```

plotLyDataToTrace(data)

print(message)

progress(val, key='progress-1', title=')

发送进度信息

Params val: float 进度值 [0-1]

Params key: str 目标, key 相同的数据会往同一个目标写入数据

Params title: str 进度 标题

```
job.progress(0.2)
```

tabContainer(items, tabsPosition='top', key=None, verb=None)

发送类型为 grid 的消息组

Params keys: 需要关联的其他图表列表

Params tabsPosition: 操作类型, 可选 'top' | 'bottom'

Params key: 图标索引 key

Params key: 操作类型, 可选 'create' | 'replace' | 'append' | 'prepend'

table(key, columns, title='', verb='append', **kwargs)

发送表格数据

Params key: str 目标, key 相同的数据会往同一个目标写入数据

Params columns: Array 表格的列数据组

```
[
    {
        'name': str, 列名称
        'type': 'text' | 'html' | 'number', 列类型
        'data': unknown[] 列数据
    }
]
```

Params title: str 图表标题

Params verb: 操作类型, 可选 'create' | 'replace' | 'append' | 'prepend'

```
job.table('info message')
job.table('error message', level='error')
```

terminate(status)

发送结束消息

Params status: 结束状态 可以选 'resolved' | 'rejected' | 'aborted' | 'timed_out'

```
job.terminate('resolved')
```

Project类

```
class cloudpss.project.project.Project(*args, **kwargs)
```

CloudPSS **工程类**，用于处理加载后的工程数据

实例变量说明：

rid: 项目的 rid

name: 项目的名称

description: 项目的描述

revision: 当前项目的版本信息

configs: 当前项目的所有参数方案

jobs: 当前项目的所有计算方案

context: 当前项目的上下文相关信息

addConfig(config)

将**参数方案**添加到工程中

Params config: 参数方案 dict

PYTHON

```
config = project.createConfig('my config')
project.addConfig(config)
```

addJob(job: dict)

将**计算方案**添加到工程中

Params job: 计算方案 dict

PYTHON

```
job = project.createJob('emtp','emtp job')
project.addJob(job)
```

static create(project)

新建项目

Params: project 项目

返回: 保存成功/保存失败

PYTHON

```
Project.create(project)
保存成功
```

createConfig(name)

创建一个参数方案，根据项目的第一参数方案生成一个方案。

创建出的方案默认不加入到项目中，需要加入请调用 `addConfig :params name:参数方案名称`

返回: 返回一个参数方案 dict

PYTHON

```
job = project.createConfig('my config')  
参数方案
```

createJob(jobType: str, name)

创建一个计算方案, 创建出的方案默认不加入到项目中，需要加入请调用 `addJob`

Params jobType: 方案类型，包括电磁暂态仿真方案emtp、移频电磁暂态仿真方案sfemt、潮流计算方案powerFlow

返回: 返回一个指定类型的计算方案

PYTHON

```
project.createJob('emtp','emtp job')  
计算方案
```

static dump(project, file)

下载项目文件

Params project: 项目

Params file: 文件路径

返回: 无

PYTHON

```
Project.dump(project,file)
```

static fetch(rid)

获取项目

Params rid: 项目 rid

返回: 返回一个项目实例

PYTHON

```
project=Project.fetch('project/Demo/test')
```

static fetchMany(name=None, pageSize=10, pageOffset=0)

获取用户可以运行的项目 **列表**

Params name: 查询名称，模糊查询

Params pageSize: 分页大小

Params pageOffset: 分页开始位置

返回: 按分页信息返回项目列表

PYTHON

```
data=Project.fetchMany()  
[  
    {'rid': 'project/admin/share-test', 'name': '1234', 'description': '1234'}  
    ...  
]
```

fetchTopology(implementType=None, config=None, maximumDepth=None)

通过项目信息，[获取](#)当前项目对应的[拓扑数据](#)

Params implementType: 实现类型

Params config: config 项目参数, 不指定将使用算例保存时选中的参数方案

Params maximumDepth: 最大递归深度, 用于自定义项目中使用 diagram 实现元件展开情况

返回: 一个拓扑实例

PYTHON

```
topology=project.fetchTopology()  
topology=project.fetchTopology(implementType='powerFlow', config=config) # 获取潮流实现的拓扑数据  
topology=project.fetchTopology(maximumDepth=2) # 获取仅展开2层的拓扑数据
```

getAllComponents()

[获取](#)实现

返回: 所有元件信息

PYTHON

```
project.getAllComponents()  
{  
    'canvas_0_2': Component 实例  
}
```

getComponentByKey(componentKey: str)

通过元件的[key](#)获取对应的元件

Params key: key 元件的key

Return: Component 实例

PYTHON

```
project.getComponentByKey('canvas_0_757')  
Component 实例
```

getComponentsByRid(rid: str)

通过指定[元件类型](#)获取元件

Params str: 元件类型

返回: 按照元件的 rid 过滤后的 dict<>

PYTHON

```
project.getComponentsByRid('project/CloudPSS/newInductorRouter')
{
    'canvas_0_2': Component 实例
}
```

getProjectConfig(name)

获取 指定名称的 参数方案

Params name: 参数方案名称

返回: 同名的方案数组

PYTHON

```
project.getProjectConfig('参数方案 1')
```

getProjectJob(name)

获取 指定名称的 计算方案

Params Name: 参数名称

返回: 返回同名计算方案数组

PYTHON

```
project.getProjectJob('电磁暂态方案 1')
```

static load(filePath)

加载 本地项目文件

Params file: 文件目录

返回: 返回一个项目实例

PYTHON

```
project = Project.load('filePath')
```

run(job=None, config=None, name=None, **kwargs)

调用 仿真

Params job: 调用仿真时使用的计算方案，不指定将使用算例保存时选中的计算方案

Params config: 调用仿真时使用的参数方案，不指定将使用算例保存时选中的参数方案

Params name: 任务名称，为空时使用项目的参数方案名称和计算方案名称

返回: 返回一个运行实例

```
runner=project.run(job,config,'')
runner
```

save(key=None)

保存/创建 项目

key不为空时如果远程存在相同的资源名称时将覆盖远程项目。 **key为空**时如果项目rid不存在则抛异常，需要重新设置key。如果保存时，当前用户不是该项目的拥有者时，将重新创建项目，重建项目时如果参数的key为空将使用当前项目的key作为资源的key，当资源的key和远程冲突时保存失败

Params: project 项目

Params: key 资源id的唯一标识符，

返回: 保存成功/保存失败

```
project.save(project)
project.save(project, 'newKey') # 另存为新的项目
```

toJSON()

类对象序列化 为 dict :return: dict

static update(project)

更新 项目

Params: project 项目

返回: 保存成功/保存失败

```
Project.update(project)
```

案例介绍

- 案例 1 通过项目 rid 运行项目
- 案例 2 项目使用案例
- 案例 3 运行项目并使用 matplotlib 动态绘制曲线
- 案例 4 使用项目版本的 hash 运行算例文件
- 案例 5 导入本地结果文件
- 案例 6 配电网三相不对称三相不对称潮流计算案例
- 案例 7 SimStudio综合能源仿真

案例 1 通过项目 rid 运行项目

本实例代码介绍通过 rid 运行项目，等待项目运行结束并获取结果数据，具体见 example-run.py,案例参数代码的流程。

PYTHON

```
import sys,os
import cloudpss
import json
import time
if __name__ == '__main__':
    cloudpss.setToken('{token}')


### 获取指定 rid 的项目
model=cloudpss.Model.fetch('model/Demo/example')


try:
    runner=model.run()
    while not runner.status() :
        print('running',flush=True)
        logs = runner.result.getLogs()
        for log in logs:
            print(log)
        time.sleep(1)

    # 打印 0 号分组数据
    print(runner.result.getPlot(0))
    filePath = 'D:\data\result\test.cjob'
    cloudpss.Result.dump(runner.result,filePath)
    print('end')
except Exception as e:
    print('error',e)
# 运行结束时间为默认时间
# 通过while循环获取任务状态判断是否运行结果
```

案例 2 项目使用案例

项目使用案例。具体见 **example-project.py**, 案例参数代码。

PYTHON

```
import sys,os
import cloudpss
import json
if __name__ == '__main__':
    cloudpss.setToken('{token}')

    # 获取项目列表
    data = cloudpss.Model.fetchMany()
    print('获取项目列表', data)

    # ### 获取指定 rid 的项目
    model = cloudpss.Model.fetch('model/Demo/SDK_TEST1')
    print('项目', model)

    # 保存项目到本地
    filePath = 'D:\data\simulation\test.cproj'
    cloudpss.Model.dump(model, filePath)

    model = cloudpss.Model.load('D:\data\simulation\test.cproj')
    print('项目', model.rid)

    # 创建一个改项目的参数方案
    config = model.createConfig('测试参数方案')
    print('参数方案', config)

    # 修改参数方案
    config['args'][ 'T_Tm_change' ] = '2'

    # 将参数方案加入到项目中
    # model.addConfig(config)

    job = model.createJob('emtp', '测试计算方案')
    print('参数方案', job)

    # 修改计算方案
    job['args'][ 'end_time' ] = '2'

    # 将参数方案加入到项目中
    # model.addJob(job)

    r = model.save()
    print('保存信息', r)
```

案例 3 运行项目并使用 matplotlib 动态绘制曲线

本实例代码介绍项目运行及使用 matplotlib 动态绘制曲线，具体见 example-run-matplot.py，本示例需要自行安装 matplotlib 依赖库，案例参数代码的方法。

PYTHON

```
import sys,os
import cloudpss
import json
import matplotlib.pyplot as plt
if __name__ == '__main__':
    cloudpss.setToken('{token}')

### 获取指定 rid 的项目
model=cloudpss.Model.fetch('model/Demo/SDK_TEST1')
print('项目',model)

config=model.configs[0] # 不填默认用model的第一个config
job=model.jobs[1] # 不填默认用model的第一个job

runner=model.run(job=job,config=config)
plotMap={}
fig, ax = plt.subplots()
while not runner.status():
    print('running')
    # 获取通道分组下的曲线名称
    print(runner.result.getLogs())
    plotKeys = runner.result.getPlotChannelNames(1)
    if plotKeys is not None:
        for val in plotKeys:

            line = plotMap.get(val,None)

            # 获取曲线数据
            channel=runner.result.getPlotChannelData(1,val)

            if line is None:

                line, = ax.plot(channel['x'], channel['y'], label=val )
                plotMap[val]=line
            else:
                line.set_data(channel['x'],channel['y'])

    plt.pause(0.1)

print('end')

plt.show()
```

案例 4 使用项目版本的 hash 运行算例文件

本实例代码介绍使用项目版本的 hash 运行算例文件，具体见 example-revision.py 案例参数代码的流程。

PYTHON

```
import sys,os
import cloudpss
import json
import time

"""
运行指定项目版本的仿真程序,
运行指定版本的项目时只能参数方案和计算方案
"""

if __name__ == '__main__':
    cloudpss.setToken('{token}')


config={'args': {'T_Tm_change': '1.5', 'T_Speed_change': '2'}, 'name': '参数方案 1', 'pins': {}}
job={'rid': 'job-definition/cloudpss/emtp', 'args': {'debug': '0', 'n_cpu': '1', 'solver': '1', 'end_time': '3', 'task_cmd': '', 'step_time': '0.00002', 'begin_time': '0', 'task_queue': '', 'initial_type': '0', 'ramping_time': '0', 'load_snapshot': '0', 'save_snapshot': '0', 'solver_option': '0', 'output_channels': [['电磁转矩', '1000', '1000', '', 'canvas_0_122,canvas_0_404'], ['a相定子电流', '1000', '1000', '', 'canvas_0_305'], ['转速', '1000', '1000', '', 'canvas_0_399,canvas_0_403']], 'max_initial_time': '1', 'load_snapshot_name': '', 'save_snapshot_name': 'snapshot', 'save_snapshot_time': '1'}, 'name': '电磁暂态仿真方案 1'}
try:

runner=cloudpss.Runner.create('4043acbddb9ce0c6174be65573c0380415bc48186c74a459f88865313743230c', job=job, config=config)

    while not runner.status():
        print('running')
        print(runner.result.getLogs())
        time.sleep(1)

    # 获取所有分组信息
    plots = runner.result.getPlots()

    print(plots)
except Exception as e:
    print(e)
```

案例 5 导入本地结果文件

本实例代码介绍导入本地结果文件，导入的结果文件支持 SDK 保存的和从网站下载的结果文件。具体见 example-load-result.py。

PYTHON

```
import cloudpss
import sys
import os
# import plotly.graph_objects as go
"""

    运行指定项目版本的仿真程序,
    运行指定版本的项目时只能参数方案和计算方案
"""

if __name__ == '__main__':
    result = cloudpss.EMTResult.load('D:\\data\\result\\test.cjob')
    print(result)
    plots = result.getPlots()

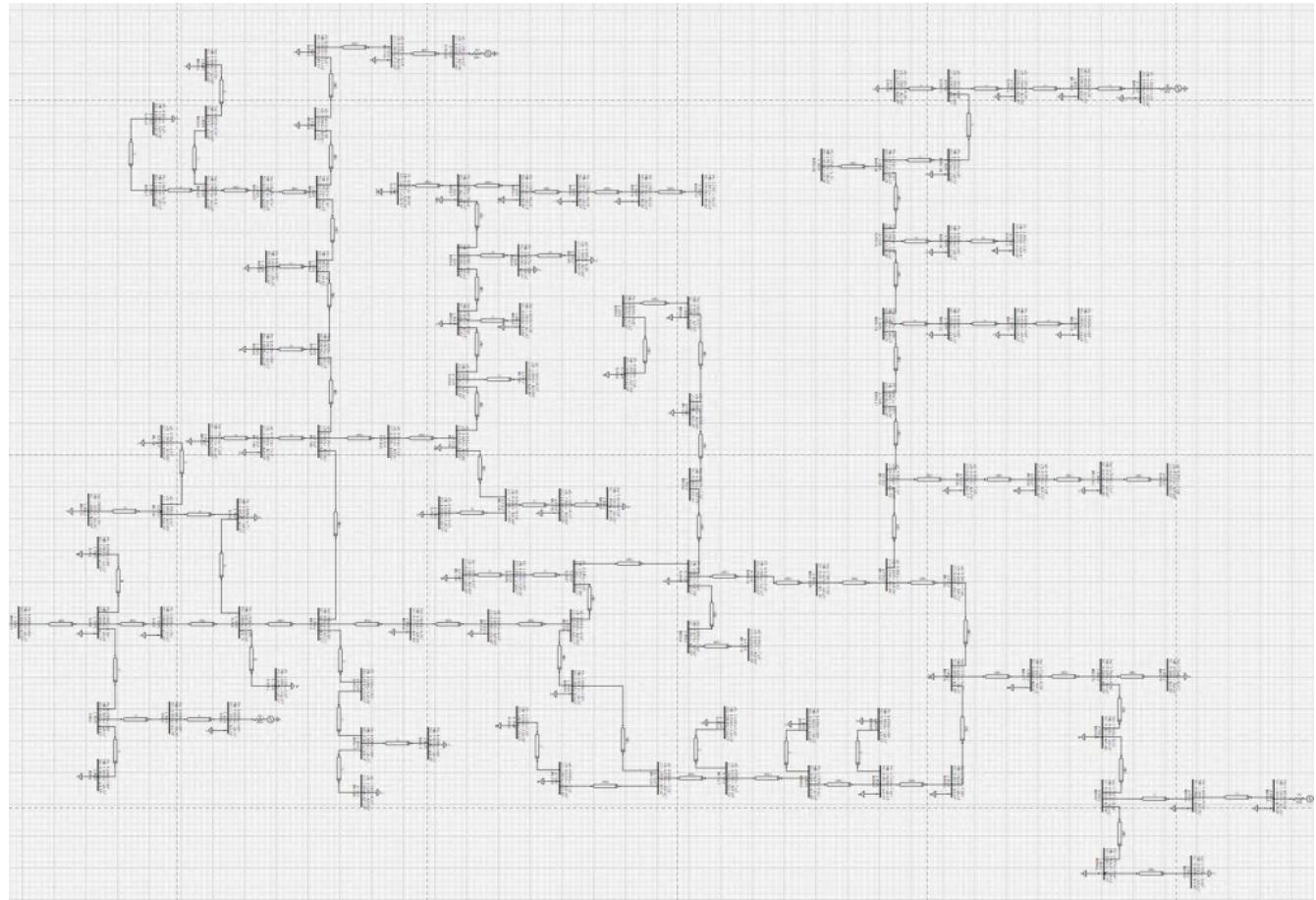
    for i in range(len(plots)):
        print(plots[i])
        # fig = go.Figure()
        # channels= result.getPlotChannelNames(i)
        # for val in channels:
        #     channel=result.getPlotChannelData(i,val)
        #     fig.add_trace(go.Scatter(channel))
        # fig.show()
```

案例 6 配电网三相不对称三相不对称潮流计算案例

本例以通过 Python 脚本对 IEEE-123 节点标准测试系统算例进行三相不对称潮流计算为例，帮助用户快速入门 CloudPSS SDK 的使用。

1. 配电网三相不对称潮流计算内核调用

首先，在 CloudPSS Simstudio 中打开 [IEEE-123 节点标准测试系统算例](#)。



点击 **运行** 标签页，在计算方案中选择默认的配电网三相不平衡潮流计算方案1。



选择潮流计算方案

点击**启动任务**运行仿真，在**结果**页面会生成三相不对称潮流计算结果。

Buses

	BusName	V/kV	Theta/deg	Pgen/kW	Qgen/kW	Pload/kW	Qload/kVar
1	BUS8A	2.1835	-4.4449	-	-	-0.0001	0.0000
2	BUS8B	2.3457	-122.6695	-	-	-0.0001	0.0000
3	BUS8C	2.3586	120.6039	-	-	0.0000	-0.0001
4	BUS97A	1.9116	-7.6048	-	-	0.0000	0.0000
5	BUS97B	2.2105	-127.9458	-	-	-0.0001	0.0000
6	BUS97C	2.3511	122.7803	-	-	0.0000	0.0000
7	BUS98A	1.9090	-7.6630	-	-	-40.0000	-20.0000
8	BUS98B	2.2079	-127.9922	-	-	0.0000	-0.0001
9	BUS98C	2.3496	122.7447	-	-	0.0000	-0.0001

Branches

	From Bus	Pij/kW	Qij/kVar	To Bus	Pij/kW	Qij/kVar
1	BUS149A	1259.8625	446.3275	BUS1A	-1229.0199	-399.2407
2	BUS149B	783.9255	538.4166	BUS1B	-787.7550	-522.6545
3	BUS149C	218.2866	318.0558	BUS1C	-213.9670	-317.5260
4	BUS14A	-60.1363	-30.1315	BUS9A	60.3814	30.3767
5	BUS14B	0.0000	0.0000	BUS9B	0.0000	0.0000
6	BUS14C	0.0000	0.0000	BUS9C	0.0000	0.0000
7	BUS99A	0.0004	-0.0077	BUS100A	-0.0002	0.0038
8	BUS99B	-0.0010	-0.0097	BUS100B	0.0004	0.0048
9	BUS99C	40.0326	20.0668	BUS100C	-40.0002	-19.9973

潮流计算结果

若要分析算例中 **Source2** 电源的A相有功出力与 **Bus25**母线的A相相角之间的关系。常规方法是手动修改参数执行多次仿真，绘制出有功出力与电压相角的描点图。这个方法操作复杂且效率低下。借助 CloudPSS SDK，利用 Python 脚本修改参数，批量调用三相不对称潮流计算内核，可以快速完成上述功能。

示例代码

配置好 **Python** 开发环境，输入以下脚本。

PYTHON

```
import time
import json
import sys
import os
sys.path.append(os.path.join(os.path.dirname(__file__), '..\\'))

if __name__ == '__main__':
    import cloudpss

cloudpss.setToken('eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9eyJpZCI6MSwidXNlcj5hbWUiOiJhZG1pbisInNjb3BlcyI6WyJ1bmtpbzduIl0sInR5cGUiOiJTREsiLCJleHAiOjE2NTg1NjgzNDYsImlhCI6MTYyNjk0MTQ1MX0.hDRBisqpd2Bxzg5HZVoTVnxw2GmOAihY5HHALNpFs_gCLCL45Xt8rYKrCUq3CZKq-IMmYfQvPgWI2B_QCmUezHtUuRQw_nmBBLb5NMpIAiFJJiBFDGjBvzwBAINCbBFnr8zDxUvwHZMoAb3ed9VNJDqI_CThB8Q3udTb10-TXs')

os.environ['CLOUDPSS_API_URL'] = 'https://internal.cloudpss.net/'
# 获取指定 rid 的项目
project = cloudpss.Model.fetch('model/lcm20/123Nodes3')

comp = project.getComponentByKey('component_dacVoltageSource_1')
comp.args['pf_PPA'] = '180'
print(comp.args)
config = project.configs[0] # 不填默认用project的第一个config
```

```

job = project.jobs[3] # 不填默认用project的第一个job
runner = project.run(job, config)
while not runner.status():
    # print('running', flush=True)
    logs = runner.result.getLogs()
    for log in logs:
        # del log['id']
        print(log)
    # 获取所有分组信息
    time.sleep(1)
print('end')
print("getBranches:", runner.result.getBranches())
print("getBuses:", runner.result.getBuses())

```

地址与Token替换

PYTHON

```
os.environ['CLOUDPSS_API_URL'] = 'https://internal.cloudpss.net/'
```

使用时需要将 `'https://internal.cloudpss.net/'` 替换为用户当前使用的平台网址地址。

PYTHON

```
cloudpss.setToken('eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidXNlcm5hbWUiOiJhZG1pbkiIsInNjb3BlcyI6WyJ1bmtub3duIl0sInR5cGUiOiJTREsiLCJleHAiOjE2NTg1NjgzNDYsImlhdi6MTYyNjk0MTQ1MX0.hDRBisqpd2bXzg5HZVoTVnxw2Gm0AihY5HHALNpFs_gcLCL45Xt8rYKrCUq3CZKq-iMmYfQvPgWIN2B_QCmUezHtUuRQw_nmBBLb5NMpIAiFJJiBFDGjBvzwBAINCbBFnr8zDxUvwHZMoAb3ed9VNJDqI_CTzB8Q3udTb10-TXs')
```

同时需要申请和修改 Token，替换 `cloudpss.setToken` 后的内容。Token 的申请和注销详见 [setToken用户认证](#) 帮助文档。

指定算例

PYTHON

```
model = cloudpss.Model.fetch('model/UserName/ModelRID')
```

使用时需要将 `model/UserName/ModelRID` 替换为用户当前使用的算例。如在当前算例中，`model/lcm20/123Node3` 表示用户 `lcm20` 下的 `123node3` 算例。

例如：若算例的 URL 为 `https://cloudpss.net/model/user/example#`，则代码中对应部分应替换为 `model/user/example`。

定位元件

PYTHON

```
comp = project.getComponentByKey('component_dacVoltageSource_1')
```

CloudPSS SDK 中，提供了 `getComponentByKey` 函数，用户可以通过元件的 `Key` 来定位算例中的某个元件。在 SimStudio 中，选中算例中的电源 配电网交流电压源2，此时浏览器地址栏变为 `https://internal.cloudpss.net/model/lcm20/123Nodes3#/design/diagram/cells/component_dacVoltageSource_1`，`component_dacVoltageSource_1` 即为电源 配电网交流电压源2的 Key。在每个算例中，元件 Key 是唯一的。

获取或修改参数 在定位元件后，即可获取并任意修改该元件的参数 `args`。此处，通过输出元件 `comp` 的全部参数 `args` 可知，元件参数是以字典的形式存储的。

PYTHON

```
print(comp.args)
{'BusType': '1', 'Dr': '0.2', 'Fault': '0', 'Func': '0', 'GeneratorType': '6', 'Grnd': '1', 'I': '',
'Init': '0', 'Irms': '', 'Name': 'Source2', 'P': '', 'Ph': '0', 'Q': '', 'R': '0', 'Tconstant': '0.05',
'Tfe': '0.4', 'Tfs': '0.2', 'Tramp': '0.05', 'V': '', 'Vm': '4.16', 'Vrms': '', 'f': '50', 'pf_PPA':
'180', 'pf_PPB': '225', 'pf_PPC': '225', 'pf_QQA': '100', 'pf_QQB': '100', 'pf_QQC': '100', 'pf_Qmax':
'200', 'pf_Qmin': '-200', 'pf_ThetaA': '0', 'pf_ThetaB': '0', 'pf_ThetaC': '0', 'pf_VVA': '1', 'pf_VVB':
'1', 'pf_VVC': '1', 'pf_Vmax': '10', 'pf_Vmin': '0.001'}
```

其中，`'pf_PPA': '180'` 即代表 **配电网交流电压源2** 元件 Power Flow Data 参数组中 Phase A Injected Active Power 的值，表示此 PV 节点输入系统的有功功率为 180kW。此处的赋值支持数字和表达式字符串。

PYTHON

```
comp.args['pf_PPA'] = '180'
comp.args['pf_PPA'] = '=180'
comp.args['pf_PPA'] = 180
```

通过设置此 `pf_PPA` 参数即可修改 **配电网交流电压源2**三相不对称潮流计算中的注入有功功率。上述 3 条语句执行的效果相同。

运行

PYTHON

```
config = model.configs[0]
job = model.jobs[3]
runner = model.run(job, config)
```

上述语句中的 `model.configs` 即为 SimStudio 中相关算例的全部参数方案（从上到下从0开始编号），`model.jobs` 为全部计算方案（从上到下从 0 开始编号）。通过选择指定的参数方案和计算方案，执行 `model.run(job, config)`，即可生成并执行以参数方案 `config` 和计算方案 `job` 的相应计算任务。

运行过程中，不断请求 `runner.status` 即可获得当前任务的计算状态（`False` 对应进行中，`True` 对应运行结束）。

结果输出

三相不对称潮流计算结果均保存在 `runner.result` 中。用户可查看 `result` 类的接口说明文档获取更多帮助。

PYTHON

```
print(runner.result.getBranches())
print(runner.result.getBuses())
```

通过上述两条语句可得到母线（Buses）和线路（Branches）的三相不对称潮流计算结果，其与 SimStudio 的结果页面展示的结果一致。

若需要输出表格中的某些单元格的数值，使用 Python 的切片操作获取即可。例如：

PYTHON

```
Busesresult = runner.result.getBuses()
Vavalue = Busesresult[0]['data'][‘columns’][2][‘data’][182]
print(Vavalue)
```

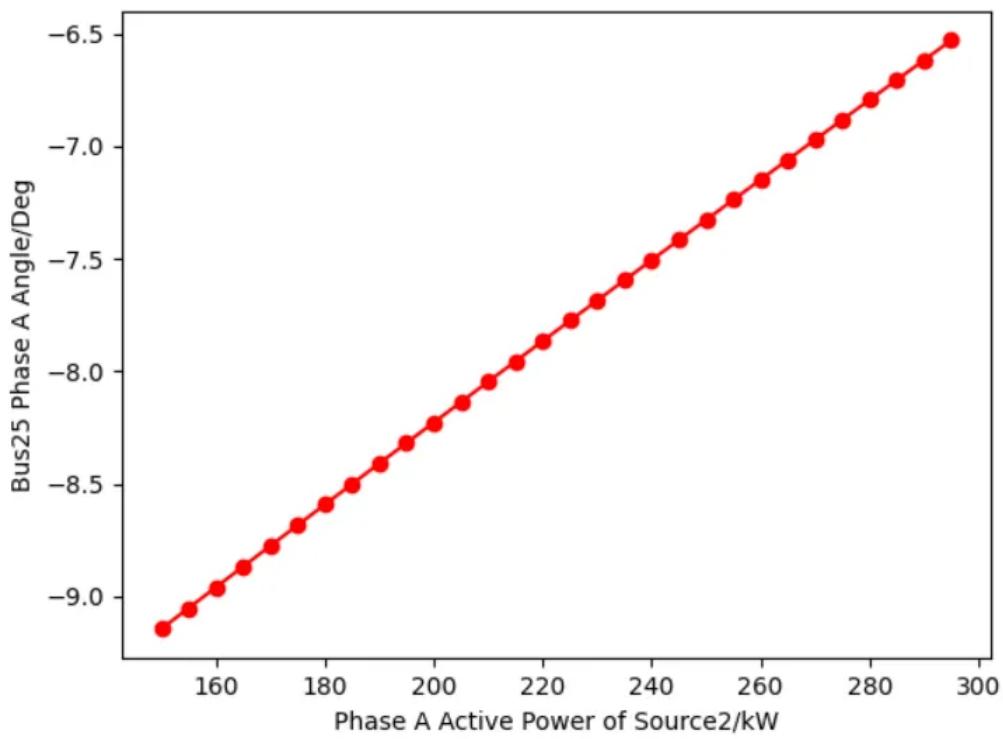
此时输出的数值即为此算例在 **配电网交流电压源2** 输入功率为 180kW 时对应的 Bus25 母线的A相相角。

Buses							
	BusName	V/kV	Theta/deg	Pgen/kW	Qgen/kW	Pload/kW	Qload/kVar
180	BUS23B	2.3398	-122.4040	-	-	0.0000	-0.0001
181	BUS23C	2.3416	119.5772	-	-	0.0000	-0.0001
182	BUS24C	2.3351	119.5230	-	-	-40.0001	-20.0000
183	BUS25A	2.2310	-8.5923	-	-	-0.0001	0.0000
184	BUS25B	2.3542	-121.8710	-	-	-0.0001	0.0000
185	BUS25C	2.3498	119.3302	-	-	-0.0001	0.0000
186	BUS25RA	2.2295	-8.6687	-	-	-0.0001	-0.0001
187	BUS25RB	2.3555	-121.8524	-	-	-0.0001	-0.0001
188	BUS25RC	2.3463	119.3115	-	-	-0.0001	0.0000

输出单元格数值

批量运行潮流

若使用 **for循环** 反复调用以上操作，在 150kW 到 200kW 之间改变 配电网交流电压源2 的A相有功出力，可以得到对应母线 Bus25 的相角的变化曲线。



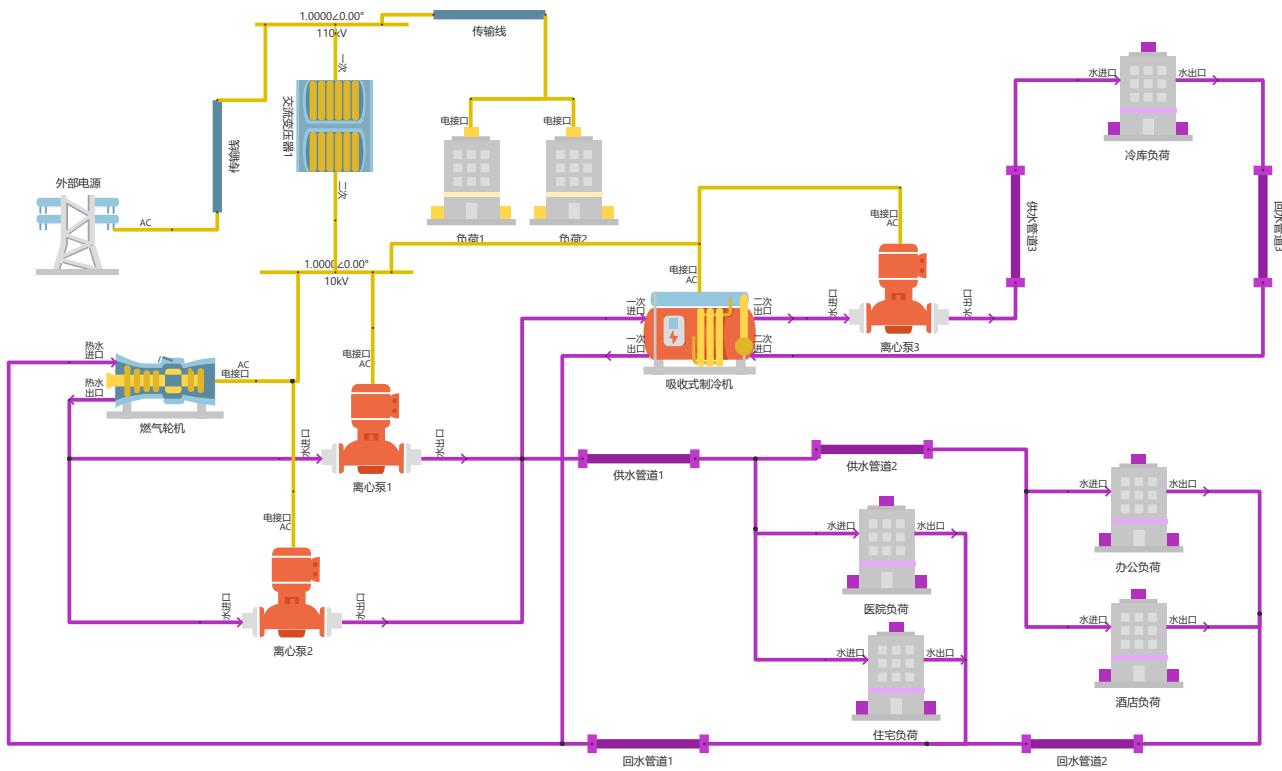
批量运行绘制曲线

案例 7 SimStudio综合能源仿真

本例以通过 Python 脚本获取SimStudio平台三联供算例计算结果为例，帮助用户快速入门综合能源 CloudPSS SDK 的使用。

算例创建和使用

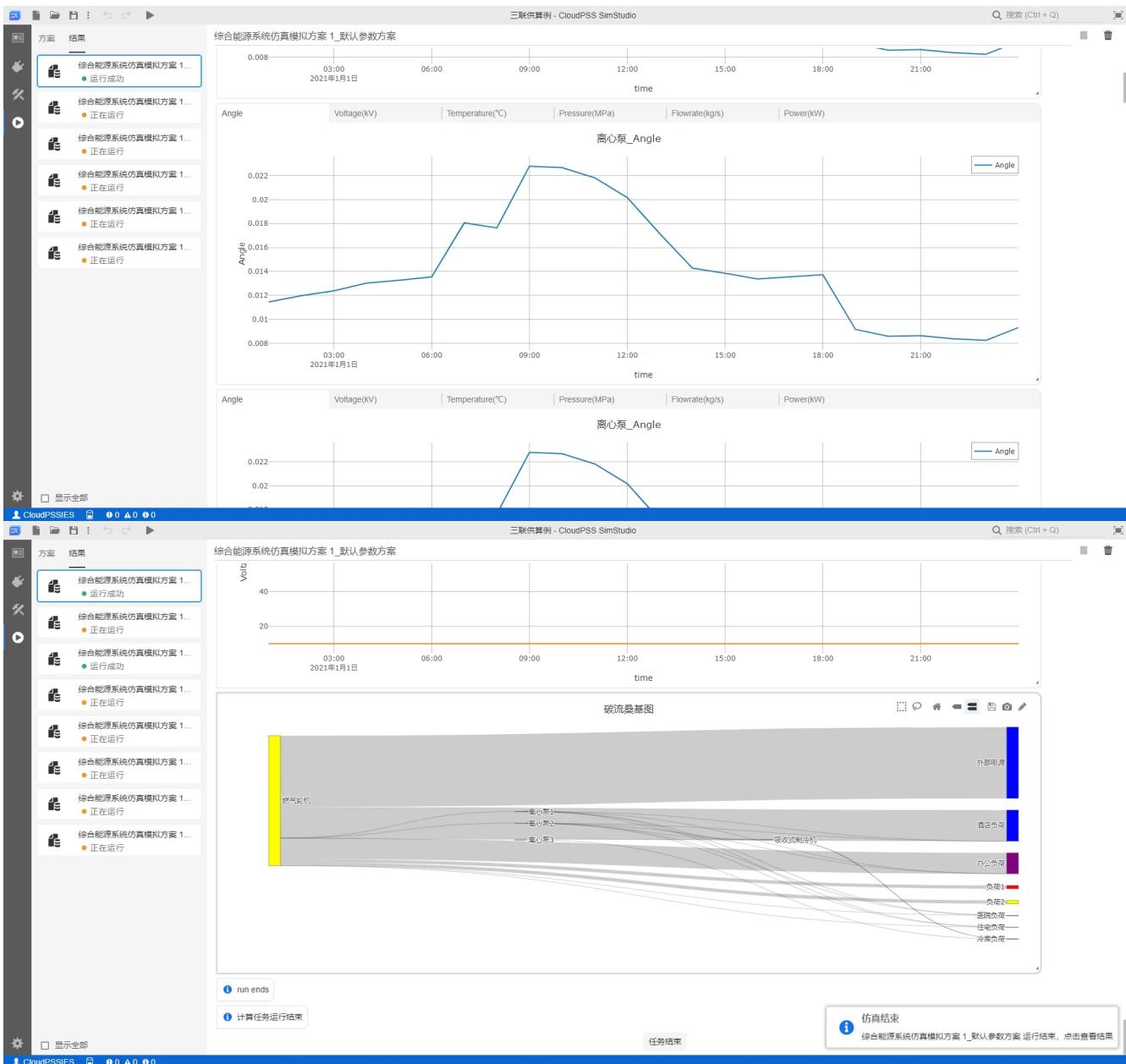
首先，在 CloudPSS Simstudio 中打开三联供算例，该算例也可以通过模板算例进行创建。



点击运行标签页，在计算方案中选择默认的综合能源系统仿真模拟方案1。



点击启动任务运行仿真，在结果页面会生成综合能源仿真计算结果。



若要批量处理综合能源的计算结果，比如将所有负荷的用能量相加来计算分时总负荷，常规的方法是在计算结果列表中找到所有的负荷元件，然后手动提取他们的负荷数据再进行相加，但由于通常综合能源系统中元件数量众多，对应的结果图表数量也非常可观，这样处理的效率十分低下。借助CloudPSS SDK，利用Python脚本获取结果数据，然后进行批量处理，可快速完成上述功能。

示例代码

配置好 Python 开发环境，这里主要用到了第三方绘图库 `Dash` 和 `JupyterDash`，用户可以使用 `pip install dash` 和 `pip install jupyter-dash` 指令进行安装。接下来执行以下脚本：

```
import time
import cloudpss
import os
import sys
from jupyter_dash import JupyterDash
from dash.dependencies import Input, Output
from dash import html, dcc
from functools import reduce

app = JupyterDash(__name__)

# Python code block
# ...
# This block contains the Python code for the example application, which
# uses the Dash library to create a user interface for the simulation results.
# It includes imports for time, cloudpss, os, sys, JupyterDash, dash.dependencies,
# dash, html, dcc, and functools.reduce. The app variable is set to JupyterDash(__name__).
# The code then proceeds to define the application logic and UI components.
# ...
# The code ends with a large block of placeholder text: '# Python code block'.
# ...
# This block contains the Python code for the example application, which
# uses the Dash library to create a user interface for the simulation results.
# It includes imports for time, cloudpss, os, sys, JupyterDash, dash.dependencies,
# dash, html, dcc, and functools.reduce. The app variable is set to JupyterDash(__name__).
# The code then proceeds to define the application logic and UI components.
# ...
# The code ends with a large block of placeholder text: '# Python code block'.
```

```

def init_html():
    tabs=[]
    for val in tabData:
        tabs.append(dcc.Tab(label=val['name'], value=val['id']))
    tabs.append(dcc.Tab(label='Tab Two', value='tab-2-example-graph'))
    app.layout = html.Div([
        html.H1('Total Thermal Load'),
        dcc.Tabs(id="tabs-example-graph", value=tabData[0]['id'], children=tabs),
        html.Div(id='tabs-content-example-graph')
    ])

@app.callback(Output('tabs-content-example-graph', 'children'),
              Input('tabs-example-graph', 'value'))
def render_content(tab):
    for data in tabData:
        if data["id"]==tab:
            return html.Div([
                dcc.Graph(
                    id=data["id"],
                    figure={
                        'data': [
                            {
                                'x': data['x'],
                                'y': data['y'],
                            }
                        ]
                    }
                )
            ])
    )

if __name__ == '__main__':
    cloudpss.setToken('{token}')

### 获取指定 rid 的项目
project = cloudpss.Model.fetch('model/CloudPSS/CHPCase')

### 通过采暖制冷负荷的rid获取所有的采暖制冷负荷元件信息
loadComponents = project.getComponentsByRid('model/CloudPSS/IES-HeatColdLoad')

try:
    # 运行仿真
    runner = project.run()
    while not runner.status():
        print('running', flush=True)
        logs = runner.result.getLogs()
        for log in logs:
            print(log)
        time.sleep(1)

    ...
    getPlotData获取key对应的采暖负荷元件的时序负荷结果,
    将每个采暖制冷负荷元件的时序负荷结果组成List
    ...
    loadList = [
        runner.result.getPlotData('/' + key, 'Power(kW)', 'Thermal Load')['Thermal Load']['y']
        for key in loadComponents.keys()
    ]
    timeSerial = runner.result.getPlotData('/' + list(loadComponents.keys())[0], 'Power(kW)', 'Thermal Load')['Thermal Load']['x']

    # 分别统计分时热负荷和分时冷负荷, 热负荷在LoadList中的数值为正, 冷负荷为负
    totalHeatLoad = [0.0]*len(loadList[0])
    totalCoolLoad = [0.0]*len(loadList[0])
    totalHeatLoad = reduce(lambda x, y:
                           [x[i] + (y[i] if y[i] > 0 else 0) for i in range(len(x))],
                           loadList,
                           totalHeatLoad)

```

```

totalCoolLoad = reduce(lambda x, y:
    [x[i] + (-y[i] if y[i] < 0 else 0) for i in range(len(x))],
    loadList,
    totalCoolLoad)

tabData = []
tabData.append({"name": "HeatLoad", "x": timeSerial, "y": totalHeatLoad, "id": "HeatLoad"})
tabData.append({"name": "CoolLoad", "x": timeSerial, "y": totalCoolLoad, "id": "CoolLoad"})
init_html()
app.run_server(mode='inline')
except Exception as e:
    print('error', e)

```

上述代码运行后，将在python终端输出如下图的结果曲线。

Total Thermal Load



Total Thermal Load



获取结果

综合能源仿真计算结果均保存在 `runner.result` 中。用户可查看 `result` 类的接口说明文档获取更多帮助。

PYTHON

```
print(runner.result.getPlotData('/component_ies_heat_cold_load_8', 'Power(kW)', 'Thermal Load'))  
print(runner.result.getPlotData('/component_ies_heat_cold_load_8', 'Power(kW)', 'Thermal Load', 0))
```

通过上述两条语句可分别获得元件 `component_ies_heat_cold_load_8` 整个仿真周期内的负荷时序结果和首个时刻的负荷结果。返回结果以字典对象的格式进行存储，如下：

PYTHON

```
{  
    'Thermal Load': {  
        'x': ['2021-01-01 01:00:00'],  
        'y': [439.3155554349029]  
    }  
}
```

如果想整个取出目标key的数值结果，可直接访问该key下的 `y` 成员。例如：

PYTHON

```
loadValue = runner.result.getPlotData('/component_ies_heat_cold_load_8', 'Power(kW)', 'Thermal Load')['Thermal Load'][['y']]  
print(loadValue)
```

此时输出的数值即为此算例的 `component_ies_heat_cold_load_8` 采暖制冷负荷（冷库负荷）的时序结果。