

HyperWorks Python 二次开发培训

向尧齐 2025/4/14

目录

Contents

1. Python基础
2. HyperWorks Python基础
3. HyperMesh Python
4. HyperView Python
5. HyperGraph Python
6. Ribbon

Python基础

Python基础

1. 什么是Python
2. Python 编辑器: **vs code**, sublimetext; Python解释器: Python(3.8.10), Altair Compose
3. 基础语法

```

Jupyter QtConsole 5.1.1
Python 3.8.10 (default, Apr 30 2024, 20:01:56) [MSC v.1928 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.25.0 -- An enhanced Interactive Python. Type '?' for help.

IPython profile: hwx
```

Python基础

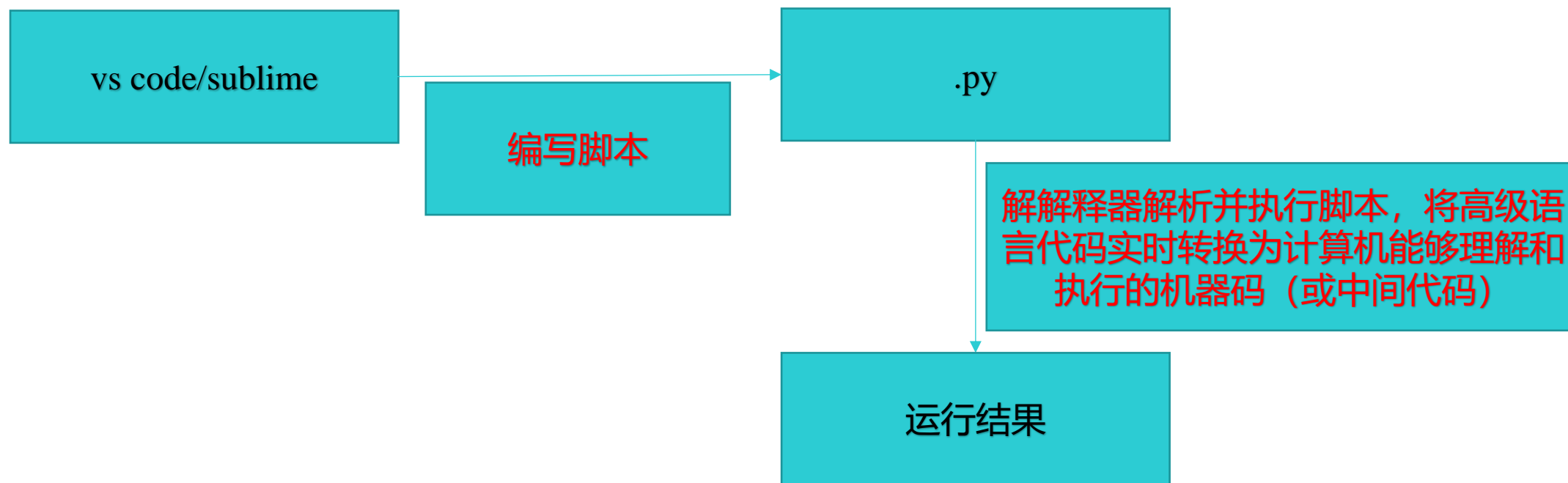
1. 什么是Python

Python是一种高级编程语言，因其简洁易读的语法而广受欢迎。它支持多种编程范式，包括面向对象、过程式和功能编程。Python有丰富的标准库和第三方模块，广泛应用于数据分析、人工智能、网络开发、自动化等领域。

Python基础

2. Python 编辑器: **vs code**, sublimetext;

Python解释器: Python(3.8.10), Altair Compose



Python基础

2. Python 编辑器: vs code, sublimetext;

Python解释器: Python (3.8.10) , Altair Compose

- 学习 Python语言自然需要一个 Python 解释器
- HyperMesh 安装后自带Python解释器
- 你也可以下载 Python 进行练习, 启动也快很多。

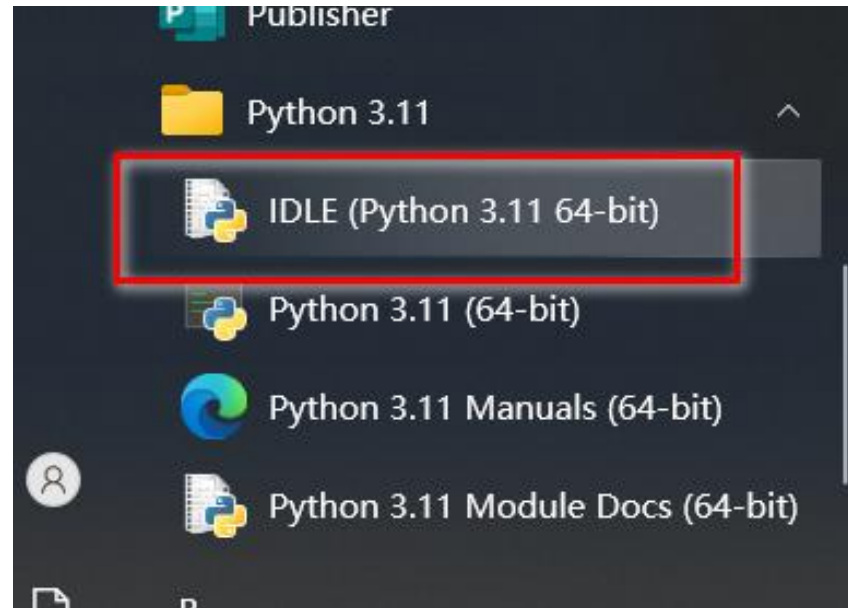
下载地址:

Welcome to Python.org

Vscode中文网 (github.net.cn)

Sublime Text - Text Editing, Done Right:<https://www.sublimetext.com/>

Python基础

A screenshot of the IDLE Shell 3.11.2 window. The title bar reads 'IDLE Shell 3.11.2'. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main text area displays the following information: 'Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32'. Below this, it says 'Type "help", "copyright", "credits" or "license()" for more information.' The prompt '>>>' is visible at the start of the next line.

```
IDLE Shell 3.11.2
File Edit Shell Debug Options Window Help
Python 3.11.2 (tags/v3.11.2:878ead1, Feb 7 2023, 16:38:35) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```


Python基础语法

1. print用法
2. 变量和数据类型
3. 控制流
4. 函数
5. 数据结构
6. 包
7. 类和对象

Python学习资料

3.8.20 Documentation

Python » zh_cn » 3.8.20 » 3.8.20 Documentation »

下载

下载这些文档

其它资源

PEP 索引

初学者指南

推荐书籍

音视频小讲座

Python 开发者指南

Python 3.8.20 文档

欢迎！这里是 Python 3.8.20 的中文文档。

按章节浏览文档：

Python 3.8 有什么新变化？

或显示自 2.0 以来的全部新变化

教程

从这里看起

标准库参考

放在枕边作为参考

语言参考

讲解基础内容和基本语法

Python安装和使用

各种操作系统的介绍都有

Python 指南

深入了解特定主题

索引和表格：

全局模块索引

快速查看所有的模块

总目录

所有的函数，类，术语

术语对照表

解释最重要的术语

安装 Python 模块

从官方的 PyPI 或者其他来源安装模块

分发 Python 模块

发布模块，供其他人安装

扩展和嵌入

给 C/C++ 程序员的教程

Python/C API 接口

给 C/C++ 程序员的参考手册

常见问题

经常被问到的问题（答案也有！）


搜索页

搜索文档

完整的内容表

列出所有的章节和部分

10



Python学习资料

python

搜索

字节跳动

如何逼自己一周学会《Python》

存下吧 很难找全的

32.3万

1416

39:58:14

【全748集】目前B站最全最细的Python零基础全套教程，2024最新版，...

Python官方课程 · 8-22

Python全套教程

小白学习(入门到精通)

存下吧 很难找全的

2967.7万

20.9万

24:41:07

花了2万多买的Python教程全套，现在分享给大家，入门到精通(Python...

Python_子木 · 2020-9-7

Python

快速入门 即学即用

存下吧 很难找全的

1569.5万

32.3万

31:40:20

黑马程序员python教程，8天python从入门到精通，学python看这套就...

黑马程序员 · 2022-8-8

清华大学

终于把Python整理成了《动画片》

存下吧 很难找全的

229.4万

4520

23:18:04

【全748集】清华大佬终于把Python做成动画片了，通俗易懂，2023最新...

图灵学院教程 · 2023-9-4

全能晋升辅导班

Python+数据分析+机器学习

名师辅导，个性化学习方案

22.7万

Python数据分析+机器学习全能辅导班

IT私塾 · 已更187课时

逼自己一周学完你的Python就牛了

全干货无废话，(568集)学完即可就业!

存下吧 很难找全的

1.5万

953

13:00:54

【全568集】强推! 2024最细自学Python全套教程! 允许白嫖，拿走不...

Python编程学院 · 9-16

test.py

20分钟学完一遍python基础

存下吧 很难找全的

54.5万

1086

20:53

20分钟学完一遍python基础

阿岳 · 2021-2-18

你敢学我就敢发

Python爬虫

从入门到入狱

47.3万

2458

73:16:23

只要你敢学我就敢发! 500集Python爬虫教程，从入门到入狱! 全程干...

Python大本营 · 2023-7-10

清华大学

终于把Python整理成了《动画片》

存下吧 很难找全的

4.3万

321

11:29:44

【全568集】清华大佬终于把Python做成动画片了，全程通俗易懂，2024...

Python零基础课程 · 9-15

清华大学

如何逼自己一周学完《Python》

存下吧 很难找全的

30.2万

13:20

【整整600集】清华大学198小时讲完的Python教程(数据分析)通俗易...

糖炒代码 · 9-19

字节跳动

终于把Python整理成了《动画片》

存下吧 很难找全的

134.4万

74

47:58:58

【全748集】字节大佬终于把python做

小白玩转Python数据分析训练营

项目可写进简历

存下吧 很难找全的

130.7万

【限时特惠】小白玩转Python数据分

逼自己一个月学完其实你很会Python

存下吧(最新版728集)学完即可就业!

存下吧 很难找全的

3.6万

414

46:39:38

【全728集】强推! 2024最细自学Pyt

Python入门到精通

千万播放 全程高能

存下吧 很难找全的

1871.5万

25.2万

55:03:06

黑马程序员Python教程_600集Python

FishC Studio

零基础入门学习Python

存下吧 很难找全的

980.8万

12万

17:35:28

【Python教程】《零基础入门学习Python

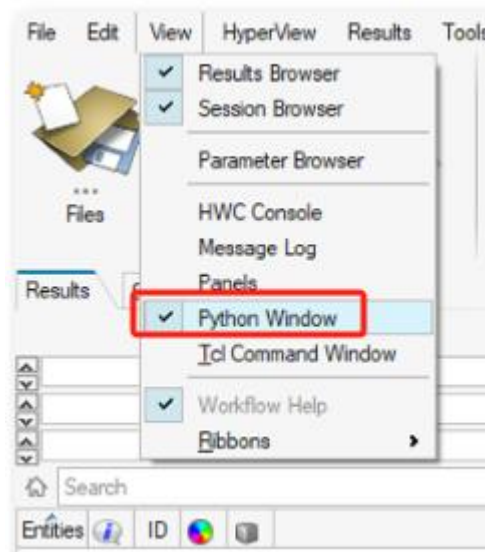
客服

顶部

HyperWorks Python基础

HyperWorks Python 控制台

- 使用Python API与应用程序客户端进行交互有多种方法。可以通过IPython控制台直接执行API调用，在" View "下拉菜单中点击" Python Window "来激活IPython控制台。



- 一旦进入控制台，您可以执行标准的Python调用，并导入您打算使用的各种包。例如，`import hw` 将使您可以访问框架类和函数（参见框架包）。

```
In [8]: import hw

In [9]: print("hello hw python")
hello hw python

In [10]:
```

HyperWorks Python 控制台

- 使用`clear`可以清除控制台中的所有显示的内容

```
In [4]: clear
```

- 要在应用程序中运行的Python脚本，可以使用内置的IPython魔术命令`run file_path`来调用它。

- ✓ `run G:\demo.py`

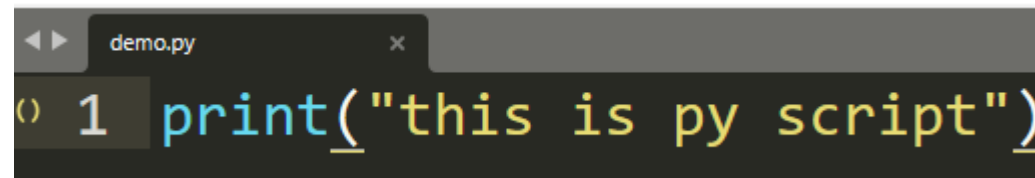
```
In [52]: run G:\demo.py  
this is py script
```

- ✓ `run "G:\demo.py"`

```
In [53]: run "G:\demo.py"  
this is py script
```

- ✓ `run G:\demo`

```
In [54]: run G:\demo  
this is py script
```



```
demo.py  
1 print("this is py script")
```

HyperWorks Python 控制台

- 如果需要传递一个变量, 使用`get_ipython().run_line_magic`

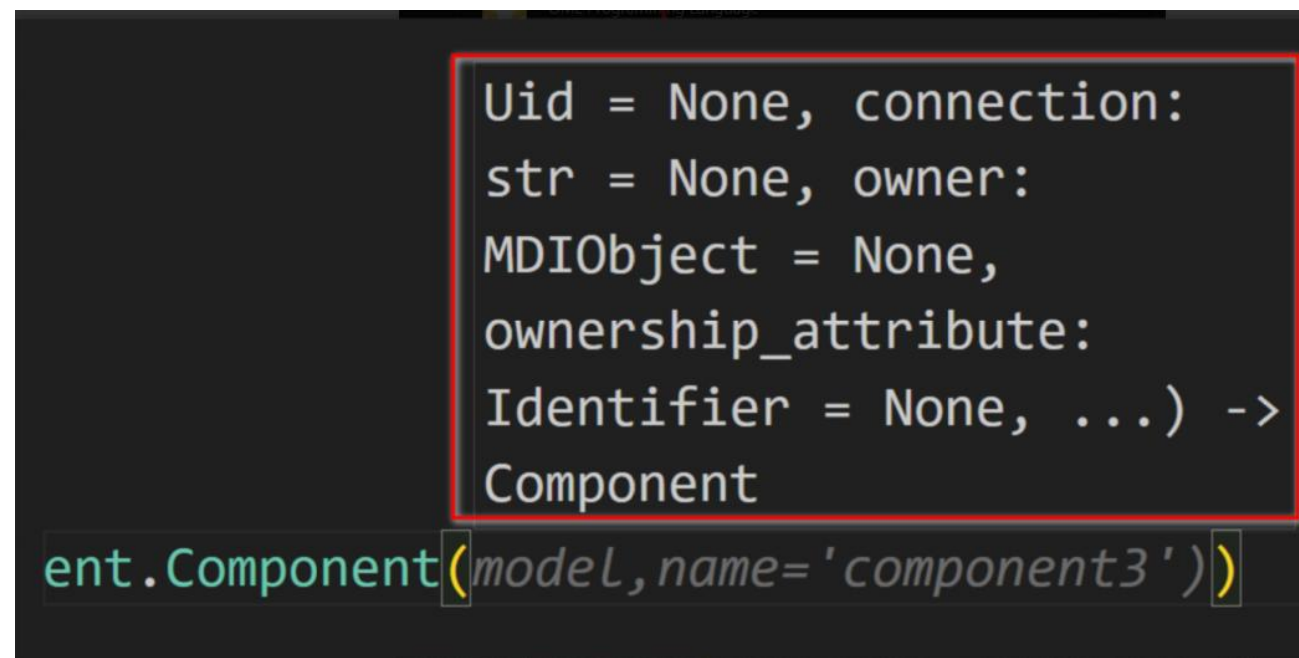
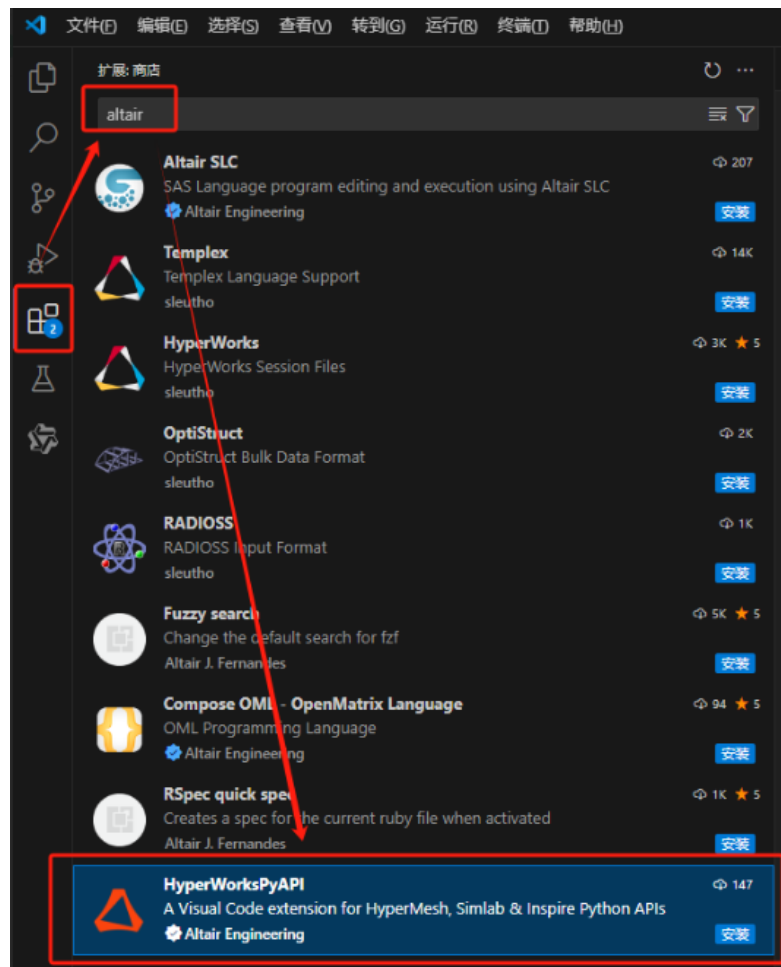
```
In [55]: file_path = r"G:\demo.py"
```

```
In [56]: get_ipython().run_line_magic('run', f'{file_path}')  
this is py script
```

```
In [57]: get_ipython().run_line_magic("run", f"{file_path}")  
this is py script
```


HyperWorks Python

推荐编辑器vs code



HyperWorks Python 帮助文档

Altair® HyperWorks®

2024

- An asterisk ("*") wildcard matches one or more of any character.
- A question mark ("?") wildcard matches any one character.
- For best results, remove the wildcard (*) when searching for a con

Match: ☒ any search words ☐ all search words

Product:

☐ All

☐ Altair HyperWorks Introduction

☒ API, Reference Guide

☐ Collaboration Tools

☐ HyperGraph

☐ HyperLife

☐ HyperLife Weld Certif

☐ HyperMesh

☐ HyperMesh CFD

☐ HyperMesh NVH

☐ HyperStudy

☐ HyperView

Search results for: python

17 results found in 2 pages.

1. 2024 API Programmer's Guide

Product: API, Reference Guide

Python API The HyperMesh 2024 release introduces new Python API v
2. Task Manager Python API

Product: API, Reference Guide

The Python API documentation for the Task Manager. See the Task Ma
3. Python API

Product: API, Reference Guide

The Python API documentation for HyperMesh applications.

Altair® HyperWorks®

2024

Home / API, Reference Guides / Python API

Welcome

What's New

Get Started

Tutorials

API, Reference Guides

- > Extensions
- > External Readers and Altair Binary Format
- > Generic ASCII Reader
- > Batch Mode
- > External Resources

Python API

- > Tcl/Tk Commands
- > Translators
- > Result Math
- > HWC Console

View All Altair HyperWorks Help

Python API

The Python API documentation for HyperMesh applications.

See [Python API Reference Guide](#) for more information.

HyperMesh Python 二次开发

- Getting Start: 基础介绍
- Framework: 截图, 字体设置等与框架相关API
- **HyperMesh: 前处理**
- **HyperView: 后处理结果**
- **HyperGraph: 后处理曲线结果**
- Task Manager
- GUI Toolkit: 界面相关。



[Getting Started](#) [Framework](#) [HyperMesh](#) [HyperView](#) [HyperGraph](#) [More ▾](#)

Python API Reference Guide

This manual provides the usage information about the Python API for the following clients and tools in the HyperMesh application: **HyperMesh** client, **HyperView** client, **HyperGraph** client, **Task Manager**, **GUI Toolkit** and the application **framework**.

[Getting Started](#) contains a reference guide introduction and provides the best starting point for the user.

Each individual section contains details of the classes included in the particular module it is covering, as well as a set of practical examples.

Contents:

[Getting Started](#)

[Framework](#)

[HyperMesh](#)

[HyperView](#)

[HyperGraph](#)

[Task Manager](#)

[GUI Toolkit](#)

HyperMesh Python

hm module

model = hm.Model()

Model Class Modify Functions

- model.translatemark(collection,vector,distance)
- model.deletemark(collection)
- model.defaultmeshsurf

Model Class Query Functions

- model.hm_
- model.hm_getentitylist
- model.hm_getmass

```
res = model.hm_getmass(collection)
status,dct= res
结果值并不是字典, 是hw中的一种特殊的数据类型
lst_key = dct.keys
key = lst_key[0]
dct[key]
```

- 1. 命令名称与tcl中命令一致;
- 2. 参数为collection/entity和hwstring

hm.entities

- hm.entities.Analysisparameter...
- hm.entities.Element

hm.entities.Component

- hm.entities.Component(model)
- hm.entities.Component(model,id)
- hm.entities.Component(model,name=comp_name)

创建或获取对象

1. 无参数, 创建组件;
2. 参数id, 返回指定id的对象;
3. 参数名称, 创建指定名称的组件

comp = hm.entities.Component(model)

- 查询comp.name
- 修改comp.name = ""
- 查询comp.id
- comp.id= 不能修改id
- comp.propertyid = ent_prop

hm.entities.Property

prop_new = hm.entities.Property(model)

hm.Collection

- hm.Collection(model, hm.entities.Component)
- hm.Collection(model, hm.entities.Component,'id=1')
- hm.Collection(model, hm.entities.Component,[1,2,3])
- hm.Collection(model, hm.entities.Component,'name=comp_name')
- hm.CollectionByInteractiveSelection(model,hm.entities.Element)
- hm.Collection(model, filter, collection)
- hm.Collection(model, hm.entities.Component,collection')

集合的常见的4种基础用法

1. 无参数, 标记所有组件;
2. 指定id, 标记指定id组件;
3. 指定数列, 标记相应组件;
4. 指定名称, 标记指定组件

若参数错误, 有可能标记所有的对象

界面交互选择

comps = hm.Collection(model,ent.Component)
comp = list(comps)[0]
comp是hm.entities.Component类型的数据
hm.Collection可以直接转hm.entities

hm.hw

- hm.hwString
- hm.hwStringList
- hm.hwTriple
- hm.hwTripleList

参数转化

hw module

hw.evalTcl

调用tcl命令

hm.Model Class

- hm Module

- hm.Model

- Model Class Query Functions

- Model Class Modify Functions

- hm.Collection class

- hm.entities Module

- hm.hw

- hm.hwString

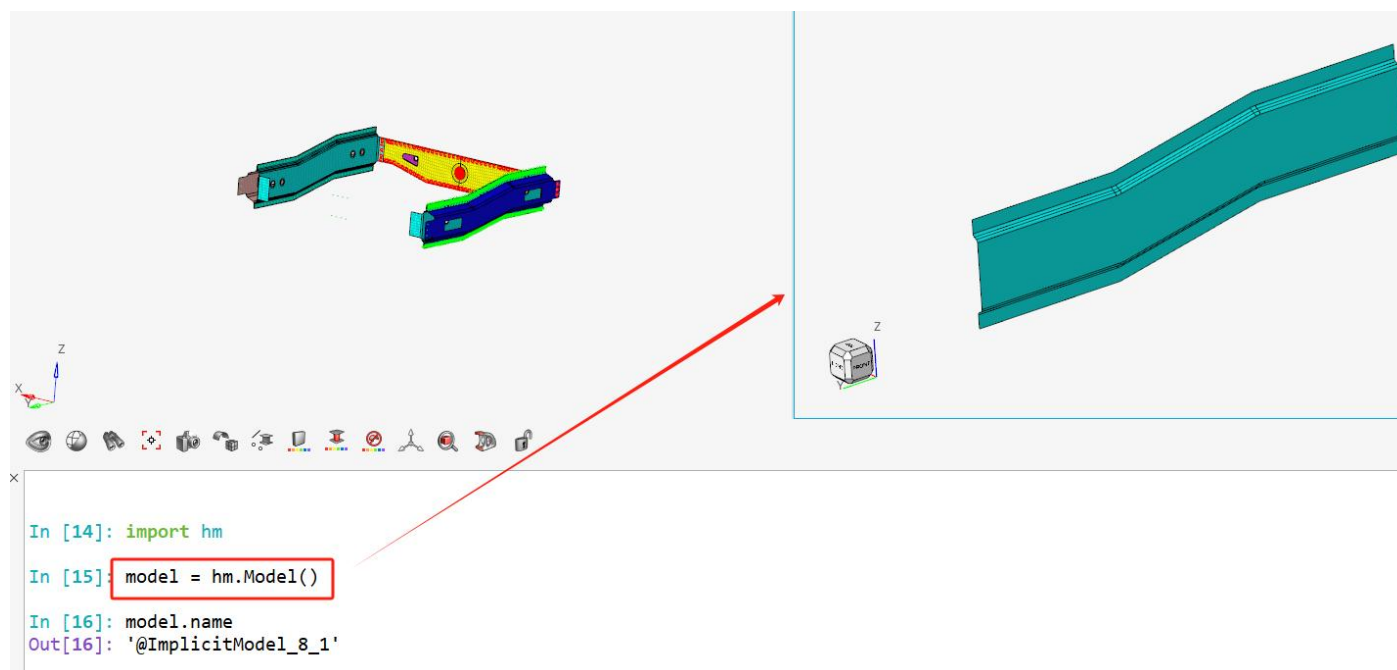
- hm.hwStringList

hm.Model()

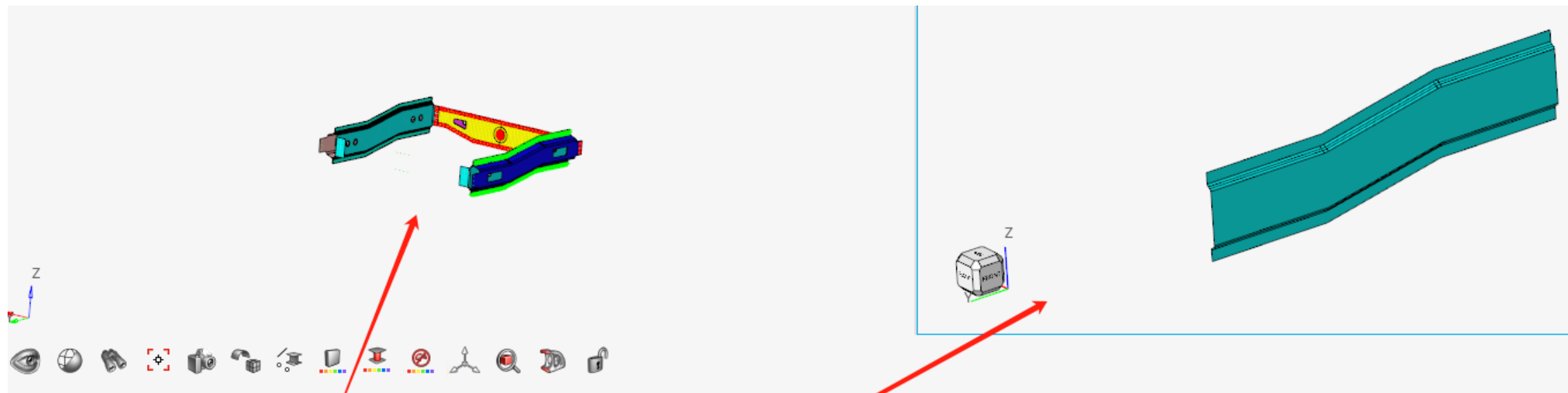
Model 类表示一个 HyperMesh 模型。从 Python API 的角度来看，这是在处理模型时的主要访问点。在执行大多数操作时，需要模型对象 - 创建新实体、查询现有实体、创建集合等等...

```
1 import hm
2 model = hm.Model()
```

通过类构造函数创建模型实例时，可以传递要使用的模型的名称。如果未提供名称，则 Model 对象将默认链接到活动模型。



hm.Model()



```
In [18]: import hm
```

```
In [19]: model_1 = hm.Model()
```

```
In [20]: model_1.name
```

```
Out[20]: '@ImplicitModel_3_1'
```

```
In [21]: model_2 = hm.Model()
```

```
In [22]: model_2.name
```

```
Out[22]: '@ImplicitModel_8_1'
```

Model Class Query Functions

实例化model后，“.”+ tab 可以提示有哪些方法，主要分为Query Functions 和 Modify Functions

```
In [37]: import hm
...: model = hm.Model()
```

```
In [38]: model.  
AE_AttachmentAbsorb  
AE_AttachmentControlConvert  
AE_AttachmentControlCreateFromAnother  
AE_AttachmentControlCreateFromDefault  
AE_AttachmentControlDefaultCreate  
AE_AttachmentCreateWithOptions  
AE_ConvertBoltLinkToAttachment
```

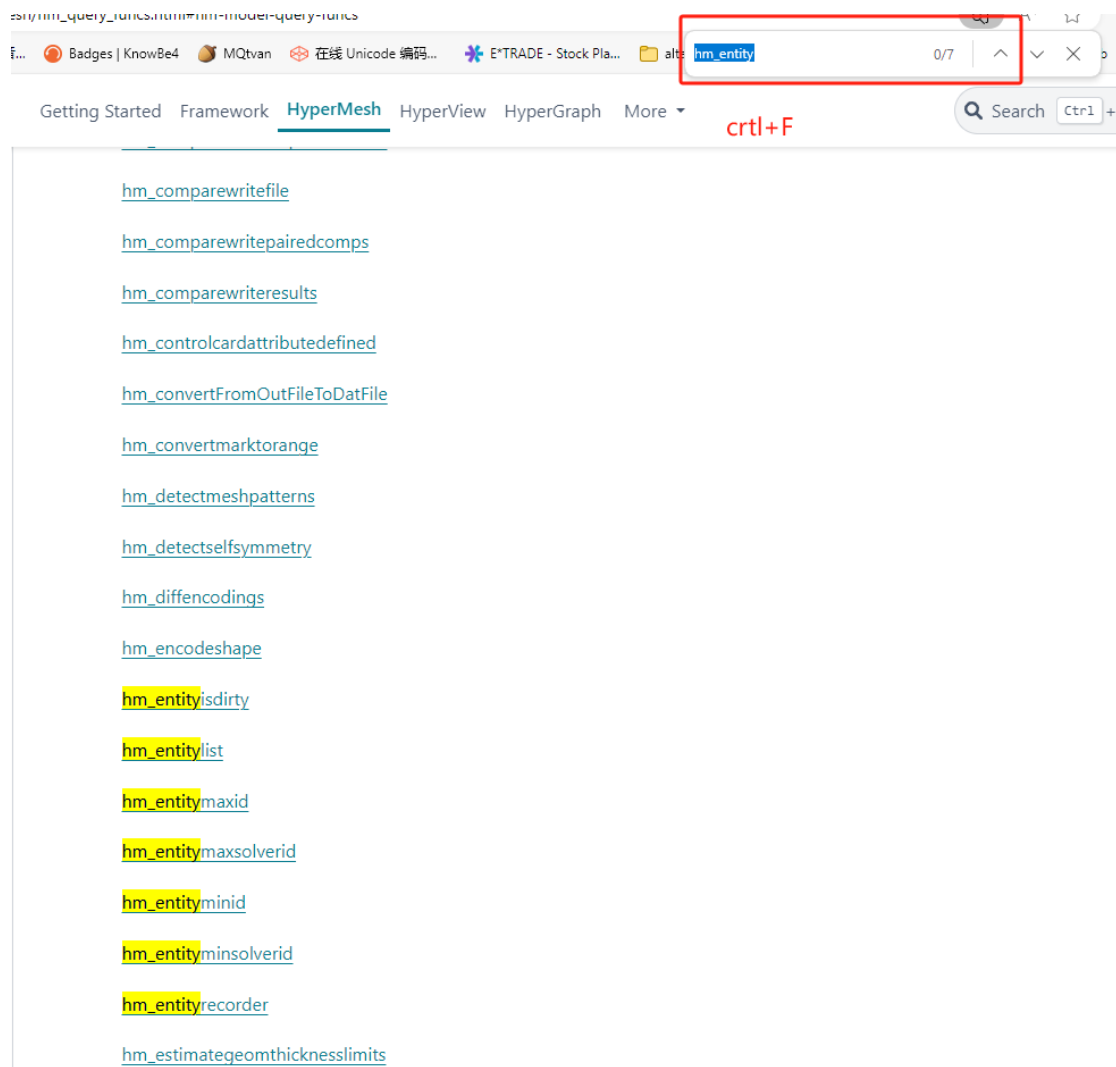
输入不同的字符串，提示的关键词内容会减少，见下图

```
In [37]: import hm
...: model = hm.Model()
```

```
In [38]: model.hm_entity  
hm_entityincollector_byentity  
hm_entityincollector_byname  
hm_entityisdirty  
hm_entitylist  
hm_entitymaxid  
hm_entitymaxsolverid  
hm_entityminid
```

Model Class Query Functions

在帮助文档中使用ctrl+F，可以根据关键字查找当前界面内容



Model Class Query Functions

Model Class Query Functions 命令**一般以hm_**开始，该类方法执行后有两个返回值，**第一个返回值记录的是命令返回的状态信息，第二个值记录的是查询的数据**，它是hm中HmQueryResult数据类型。

```
In [37]: import hm
...: model = hm.Model()
```

```
In [38]: res = model.hm_entitylist(ent.Component, 'id')
```

```
In [39]: status, array = res
```

```
In [40]: print(status)
<hwdescriptor.hwReturnStatus; proxy of <Swig Object of
0x00000206A3504720> >
```

```
In [41]: array.keys
Out[41]: ['entityList']
```

```
In [42]: array['entityList']
Out[42]: array([1, 2], dtype=uint32)
```

Model Class Query Functions

```
import hm
```

```
import hm.entities as ent
```

```
model = hm.Model()
```

```
# ent.Component与tcl中的component对应
```

```
res = model.hm_entitylist(ent.Component, 'id')
```

```
# 结果是一个列表，第一个参数记录命令的返回状态，
```

```
# 第二个参数记录返回的结果
```

```
status, dct = res
```

Model Class Modify Functions

Model Class Modify Functions, 只有一个返回值, 记录的是命令返回的状态信息。

创建一条直线示例

```
In [129]: status = model.linecreatestraight(1,1,1,2,2,2)
```

```
In [130]: status.message
```

```
Out[130]: ''
```

```
In [131]: status = model.linecreatestraight(1,1,1,1,1,1)
```

```
In [132]: status.message
```

```
Out[132]: 'Cannot create zero length segment.'
```

hm.entities

hm.entities 模块中的每个 HyperMesh 实体类型都由相应的类表示。因此，每个 HyperMesh 实体都可以表示为 Python 对象。这种面向对象的方法为用户提供了更多的功能和灵活性。在 Tcl 中，实体主要通过它们的内部 ID 来引用，而 Python API 则使用实体对象。

```

1  import hm
2  import hm.entities as ent
3
4  # 1. 没有参数
5  # 创建一个新的对象
6  new_prop = ent.Property(model)
7
8  # 2. 整数
9  # 为id=10的属性创建对象,若不存在id=10的属性则报错
10 model = hm.Model()
11 prop_10 = ent.Property(model,10)
12
13 # 3. 字符串
14 # 创建一个指定名称的entity,必须不存在指定名称的entity才可完成创建
15 ent.Property(model, name='new_prop')
```

```

import hm
import hm.entities as ent
model = hm.Model()
```

```

# 1. 没有参数
# 创建一个新的对象
new_prop = ent.Property(model)
```

```

# 2. 整数
# 为id=10的属性创建对象,若不存在id=10的属性则报错
model = hm.Model()
prop_10 = ent.Property(model,10)
```

```

# 3. 字符串 # 创建一个指定名称的entity,必须不存在指定名称的entity才可完成创建
ent.Property(model, name='new_prop')
```


hm.entities

实例化对象后，可以直接修改对象的属性，下图是创建材料属性tcl和python中的代码示例

```

1  *createentity props cardimage=PSHELL includeid=0 name="new_prop"
2  *setvalue props id=1 STATUS=1 PSHELL_T=5.5
3
4  *createentity mats cardimage=MAT1 name="new_mat"
5  *setvalue mats id=1 STATUS=1 E=210000
6  *setvalue mats id=1 STATUS=1 Nu=0.3
7  *setvalue mats id=1 STATUS=1 Rho=7.85e-09
8
9  *setvalue props id=1 materialid={mats 1}
10
11 *createmark components 1 all
12 *setvalue comps mark=1 propertyid={props 1}
13
14 *createmark components 1 all
15 *scalemark components 1 1.25 1.25 1.25 1
16
17 *createmark components 1 all
18 set mass_data [ hm_getmass components 1 ]
19 puts "mass: [ lindex $mass_data 0 ]"
20 puts "volume: [ lindex $mass_data 1 ]"
21 puts "area: [ lindex $mass_data 2 ]"

```

```

import hm
import hm.entities as ent

model = hm.Model()

prop = ent.Property(model,name="new_prop")
prop.cardimage = "PSHELL"
prop.PSHELL_T = 5.5

mat = ent.Material(model,name="new_mat")
mat.cardimage = "MAT1"
mat.E = 210000
mat.Nu = 0.3
mat.Rho = 7.85e-09

prop.materialid = mat

comps = hm.Collection(model,ent.Component)
comps.set_items("propertyid",prop)

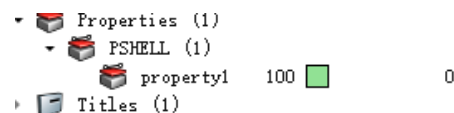
model.scalemark(comps, 1.25, 1.25, 1.25, ent.Node(model,1))

stat, res = model.hm_getmass(comps)
print(res.totalmass)
print(res.totalvolume)
print(res.totalarea)

```

hm.entities

1. 创建指定类型的对象，查询其name等属性；
2. 修改相应的属性。
3. 通过dir()，查询对象有哪些方法
4. `getattributenames()`，可以查询所有的属性名称(推荐使用)
5. **id无法通过python对象进行修改，使用hw.evalTcl**



```

In [59]: import hm
...: import hm.entities as ent
...: model = hm.Model()

```

```

In [60]: new_prop = ent.Property(model)

```

```

In [61]: new_prop.getattributenames()

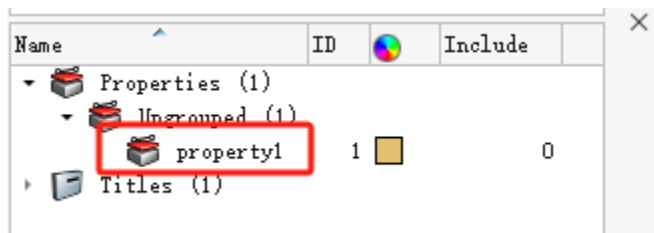
```

```

Out[61]:
['cardimage',
'color',
'definedentity',
'id',
'includeid',
'laminate',
'materialid',
'name',
'section',
'sensor']

```

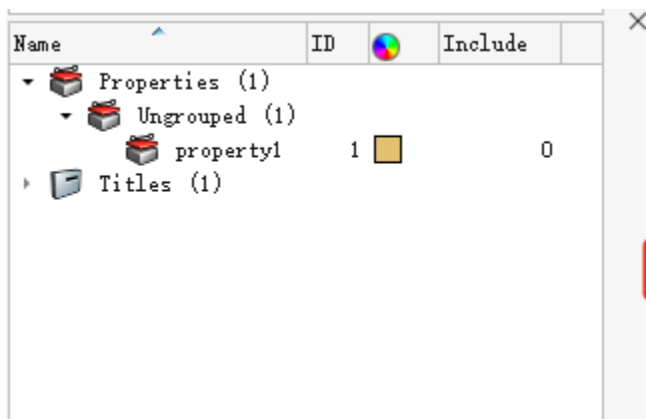
hm.entities



```
In [16]: import hm
...: import hm.entities as ent
...: model = hm.Model()

In [17]: obj_prop_1 = ent.Property(model)
```

创建新的prop



```
In [19]: obj_prop_2 = ent.Property(model, 2)

-----
RuntimeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1032\1371937042.py in <module>
----> 1 obj_prop_2 = ent.Property(model, 2)

G:\Altair_2024.1\hwdesktop\hw\python\hw\mdi\adaptors\hm\entities\mdihmmetaobject.pyc in __new__
RuntimeError: Could not find entity with id 2

In [20]: obj_prop_1 = ent.Property(model, 1)

In [21]:
```

根据id实例化prop，没有id会报错

hm.entities

实例化对象后，查询和修改对象的属性

Name	ID	Color	Include
Properties (1)			
Ungrouped (1)			
property1	1	Yellow	0
Titles (1)			

```

In [23]: obj_prop_1 = ent.Property(model,1)

In [24]: obj_prop_1.name
Out[24]: 'property1'

In [25]: obj_prop_1.color
Out[25]: 59

```

Name	ID	Color	Include
Properties (1)			
Ungrouped (1)			
new_prop_name	1	Black	0
Titles (1)			

```

In [23]: obj_prop_1 = ent.Property(model,1)

In [24]: obj_prop_1.name
Out[24]: 'property1'

In [25]: obj_prop_1.color
Out[25]: 59

In [26]: obj_prop_1.name = "new_prop_name"

In [27]: obj_prop_1.color = 1

```

```

In [28]: obj_prop_1.getattributenames()
Out[28]:
['cardimage',
'color',
'definedentity',
'id',
'includeid',
'laminate',
'materialid',
'name',
'section',
'sensor']

```

hm.entities

特殊情况：对于存在几何信息的entity，例如node, point, line, surface, solid, element

point, line, surface, solid, element无法通过ent创建，只能通过ent获取指定id的对象，创建需要使用model下的方法。

node，可以使用ent创建，可以通过设置localcoordinates属性进行坐标调整

```
In [14]: node_new = ent.Node(model)

In [15]: node_new.coordinates
Out[15]: [0.0, 0.0, 0.0]

In [16]: node_new.localcoordinates
Out[16]: [0.0, 0.0, 0.0]

In [17]: node_new.coordinates = [1,1,1]
-----
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_12300\327357521.py in <module>
----> 1 node_new.coordinates = [1,1,1]

G:\Altair_2024\hwdesktop\hw\python\hw\mdi\base.pyc in __setattr__(self, name, value)

G:\Altair_2024\hwdesktop\hw\python\hw\mdi\base.pyc in set_attribute_value(self, identifier, value)

ValueError: Value of read only attribute cannot be modified
INFO:ipykernel.inprocess.ipkernel:Exception in execute request:
-----
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_12300\327357521.py in <module>
----> 1 node_new.coordinates = [1,1,1]

G:\Altair_2024\hwdesktop\hw\python\hw\mdi\base.pyc in __setattr__(self, name, value)

G:\Altair_2024\hwdesktop\hw\python\hw\mdi\base.pyc in set_attribute_value(self, identifier, value)

ValueError: Value of read only attribute cannot be modified

In [18]: node_new.localcoordinates = [1,1,1]
```

hm.entities

创建临时节点功能封装成独立的函数

```
1  import hm
2  import hm.entities as ent
3
4  def CreateTempNode(x,y,z):
5      model = hm.Model()
6      node_new = ent.Node(model)
7      node_new.localcoordinates=[x,y,z]
```

hm.Collection

在 Python API 中，实体选择是通过 Collection 类提供的。实体选择由一个集合对象表示，用户可以针对每种实体类型创建任意数量的集合。唯一的要求是单个集合中只能包含一种实体类型，并且所有实体必须属于同一个模型。**集合是可迭代的，可以对它们进行循环操作**，以在其中对单个实体对象执行操作（参见下面的代码片段）。

```
1  import hm
2  import hm.entities as ent
3
4  model = hm.Model()
5  comps = hm.Collection(model,ent.Component)
6
7  # Querying element IDs from a collection
8  for c in comps:
9      print(c.id)
```

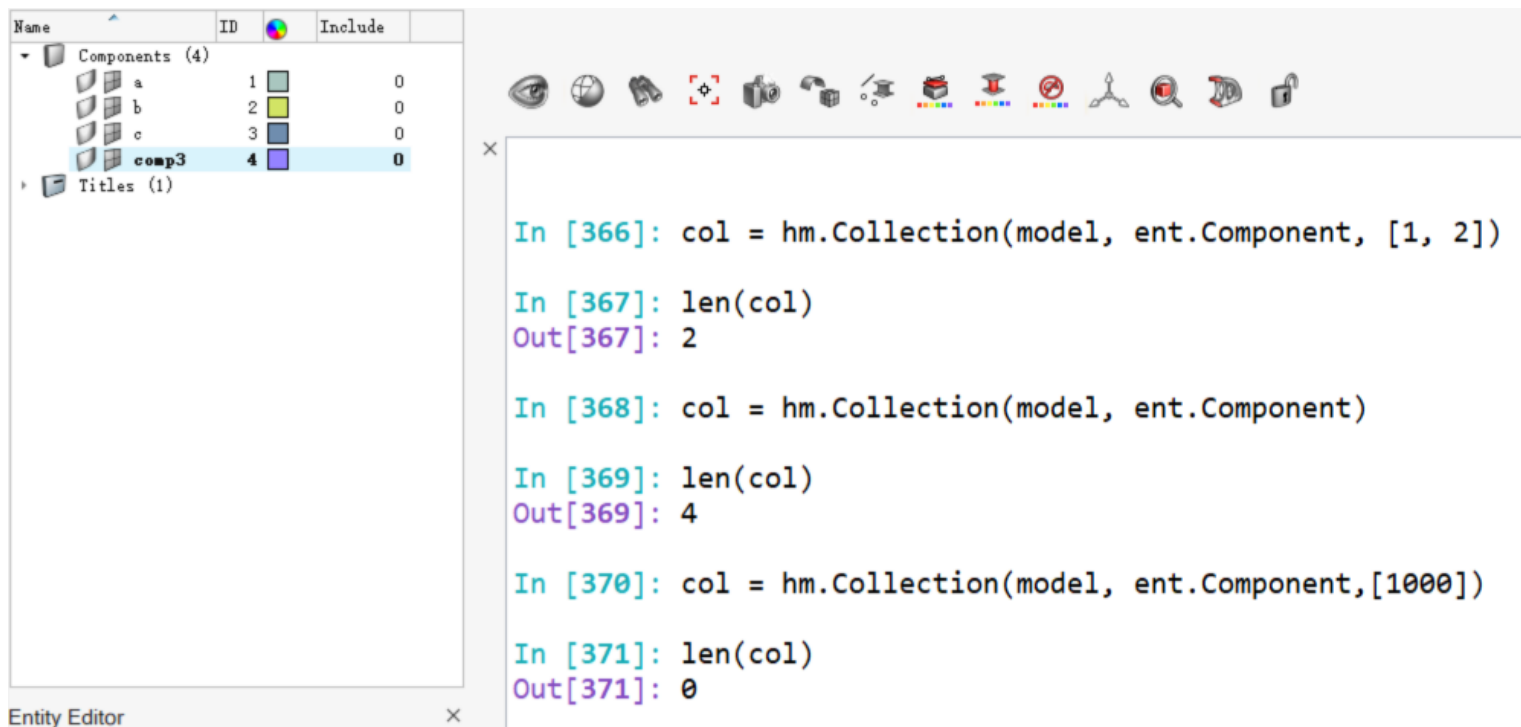

hm.Collection

该类构造函数支持多种方式来创建集合对象：

Collection(model, type)

Collection(model, type)参数：

- model (Model) – hm.Model 类的实例。
- type (Entity) – 来自 hm.entities 的 HyperMesh 实体类。



The screenshot displays the Altair HyperMesh software interface. On the left, the 'Entity Editor' window shows a tree view of components. The 'Components (4)' folder is expanded, listing 'a', 'b', 'c', and 'comp3'. 'comp3' is selected, showing its ID as 4 and its 'Include' status as 0. Below the tree, there is a table with columns 'Name', 'ID', and 'Include'.

Name	ID	Include
a	1	0
b	2	0
c	3	0
comp3	4	0

On the right, a Python console window shows the following code and output:

```
In [366]: col = hm.Collection(model, ent.Component, [1, 2])
In [367]: len(col)
Out[367]: 2

In [368]: col = hm.Collection(model, ent.Component)
In [369]: len(col)
Out[369]: 4

In [370]: col = hm.Collection(model, ent.Component, [1000])
In [371]: len(col)
Out[371]: 0
```

hm.Collection

该类构造函数支持多种方式来创建集合对象：

Collection(model, type)

Collection(model, type)参数：

- model (Model) – hm.Model 类的实例。
- type (Entity) – 来自 hm.entities 的 HyperMesh 实体类。

```
1  # 不指定id或者name, 则标记模型中的所有组件
2  col = hm.Collection(model, ent.Component)
3
4  # 标记id为1,2 的组件
5  col = hm.Collection(model, ent.Component, [1, 2])
6
7
8  # 通过名字标记, 错误示范, 字符串列表目前只支持hm.entities.Part
9  hm.Collection(model, ent.Component, ['a', 'b'])
```

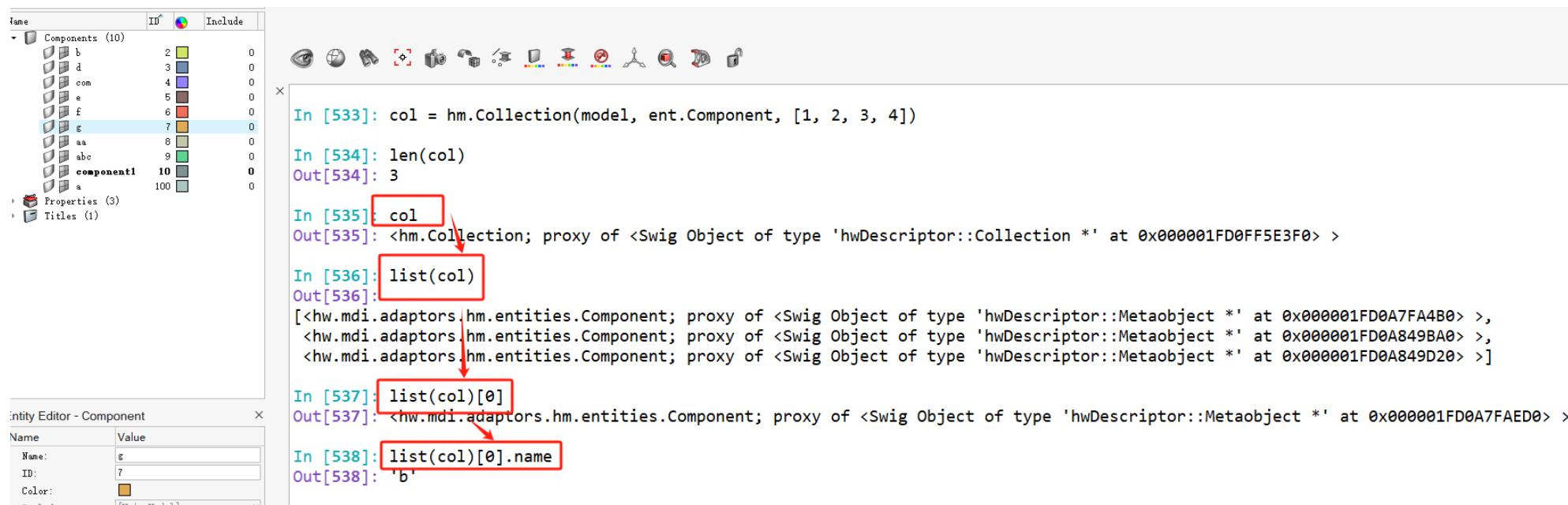
hm.Collection

集合 (Collections) 不像 Python 列表那样使用。它们没有顺序, **不能直接从集合中提取项:**

`c1 = comps[0]` # 不支持这种写法

要从集合中提取单个项, 需要先将集合转换为 Python 列表。这一操作对于大型集合可能代价较高, 因为它需要为集合中的每个实体生成一个 Python 实体对象:

`c1 = list(comps)[0]`



```

In [533]: col = hm.Collection(model, ent.Component, [1, 2, 3, 4])
In [534]: len(col)
Out[534]: 3
In [535]: col
Out[535]: <hm.Collection; proxy of <Swig Object of type 'hwDescriptor::Collection *' at 0x000001FD0FF5E3F0> >
In [536]: list(col)
Out[536]: [<hw.mdi.adaptors.hm.entities.Component; proxy of <Swig Object of type 'hwDescriptor::Metaobject *' at 0x000001FD0A7FA4B0> >,
<hw.mdi.adaptors.hm.entities.Component; proxy of <Swig Object of type 'hwDescriptor::Metaobject *' at 0x000001FD0A849BA0> >,
<hw.mdi.adaptors.hm.entities.Component; proxy of <Swig Object of type 'hwDescriptor::Metaobject *' at 0x000001FD0A849D20> >]
In [537]: list(col)[0]
Out[537]: <hw.mdi.adaptors.hm.entities.Component; proxy of <Swig Object of type 'hwDescriptor::Metaobject *' at 0x000001FD0A7FAED0> >
In [538]: list(col)[0].name
Out[538]: 'b'

```

hm.Collection

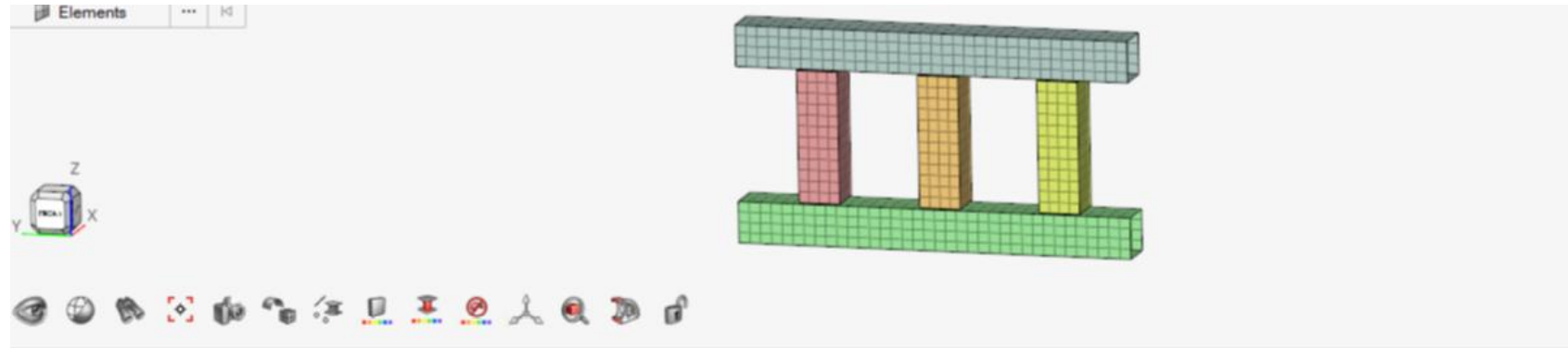
集合支持字符串的标记，不支持字符串列表的标记

Name	ID	Include
Components (10)		
b	2	0
d	3	0
com	4	0
e	5	0
f	6	0
g	7	0
aa	8	0
abc	9	0
component1	10	0
a	100	0

```
In [580]: col = hm.Collection(model, ent.Component, 'name = abc')
In [581]: list(col)[0].id
Out[581]: 9
In [582]: list(col)[0].name
Out[582]: 'abc'
```

hm.Collection

集合通过组件标记单元



```
In [91]: col_comp_1 = hm.Collection(model,ent.Component,'id=1')

In [92]: col_comp_5 = hm.Collection(model,ent.Component,'id=5')

In [93]: col_elem_1 = hm.Collection(model,ent.Element,col_comp_1)

In [94]: col_elem_5 = hm.Collection(model,ent.Element,col_comp_5)

In [95]: len(col_elem_1)
Out[95]: 192

In [96]: len(col_elem_5)
Out[96]: 576
```

hm.Collection

集合可以直接进行加减运算

```
In [105]: len(col_elem_1)
```

```
Out[105]: 192
```

```
In [106]: len(col_elem_5)
```

```
Out[106]: 576
```

```
In [107]: new_col = col_elem_1 + col_elem_5
```

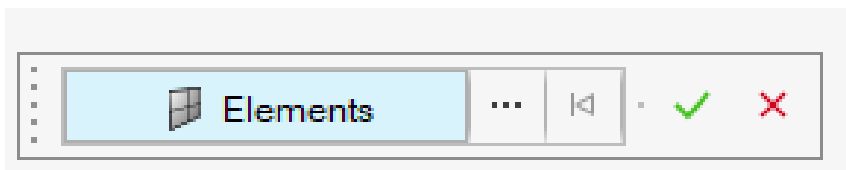
```
In [108]: len(new_col)
```

```
Out[108]: 768
```

hm.Collection

通过界面交互选择 **hm.CollectionByInteractiveSelection**

```
elems = hm.CollectionByInteractiveSelection(model, ent.Element)
```



hm.hw

Python函数的已经被调整以适应Python概念，例如使用集合而不是标记，使用实体对象而不是ID等。参数名称也已经被调整为技术上准确，但函数名称保持不变，以便用户可以轻松在文档中找到参考资料。

```
1  # tcl 命令
2  *createvector 1 0.0 1.0 0.0
3  *createmark elems 1 1
4  *translatemark elems 1 1 5.0
5
6  # python 命令
7  elems = hm.Collection(model,ent.Element)
8  vector = hm.hwTriple(0.0,0.0,1.0)
9  model.translatemark(elems,vector,50)
```


hm.hw

Python函数的签名在涉及特定HyperMesh数据类型的参数时与Tcl命令签名有所不同。字符串、整数、浮点数等在两个接口中以相同的方式暴露。下表突出了Tcl和Python函数签名在参数类型上的主要区别。

Tcl Argument Type	Python Argument Type
entity_type + mark_id	Collection
mark_id (when only one entity type is supported)	Collection
mark_id (multiple entity types are supported)	CollectionSet
entity_type + entity_id	Entity object
entity_type + list_id	EntityList
list_id (when only one entity type is supported)	EntityList
entity_id (when only one entity type is supported)	Entity object
entity_type	Entity class

hm.hw

在Tcl中，某些参数使用辅助命令来定义所需的输入。例如，上面提到的`translatemark`使用`createvector`来定义向量参数。在Python中，这些类型的参数由HyperMesh模块中提供的专用数据类型表示。在执行主命令之前不需要执行额外的函数。用户需要定义一个`hwTriple`对象，而不是使用`*createvector`。下表列出了在Python API中被HyperMesh数据类型替代的最常见的Tcl命令。

Tcl	Python
<code>*createvector</code>	<code>hwTriple</code>
<code>*createplane</code>	<code>2x hwTriple</code>
<code>*createarray</code>	<code>hwIntList</code>
<code>*createintarray2d</code>	<code>hwIntList2</code>
<code>*createdoublearray</code>	<code>hwDoubleList</code>
<code>*createdoublearray2d</code>	<code>hwDoubleList2</code>
<code>*createstringarray</code>	<code>hwStringList</code>

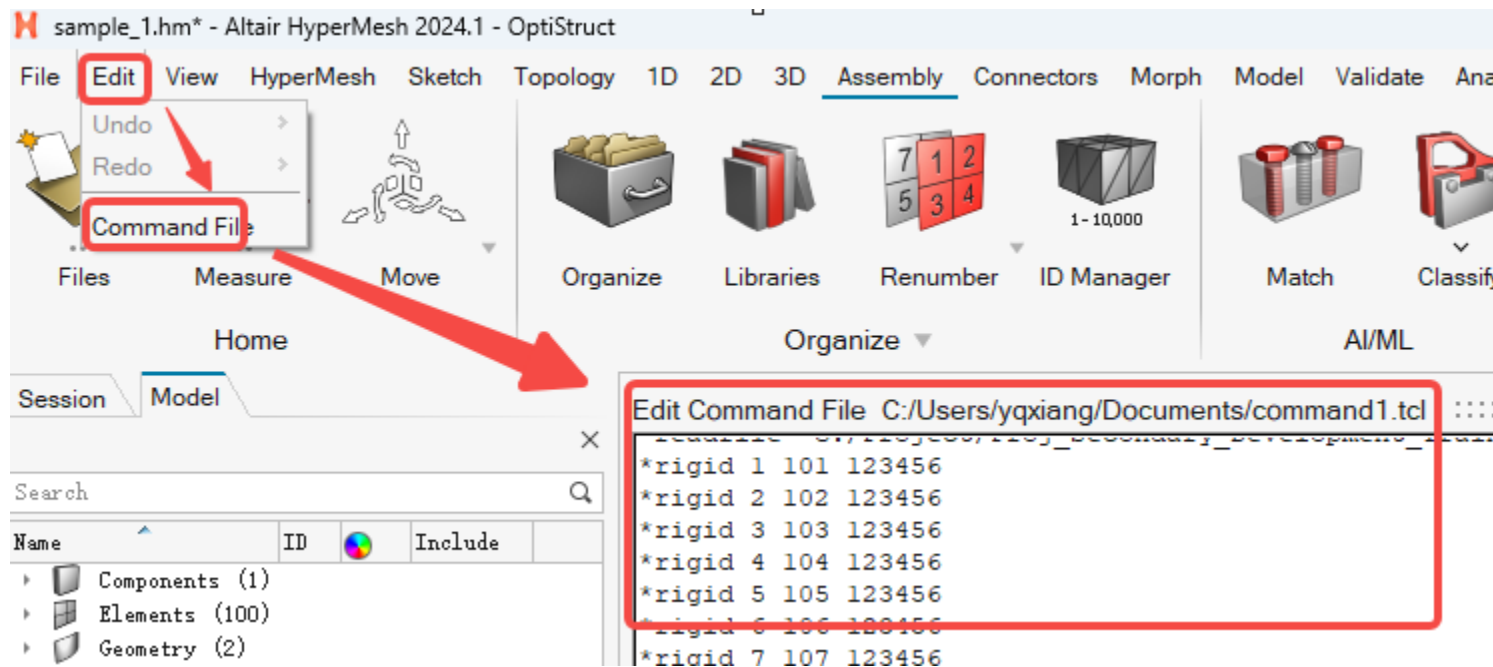
二次开发基本流程

HyperWork2025版本之前命令记录只有tcl文件

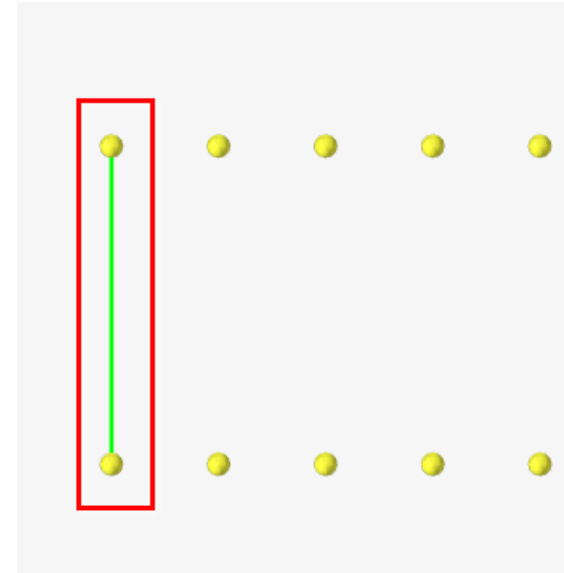
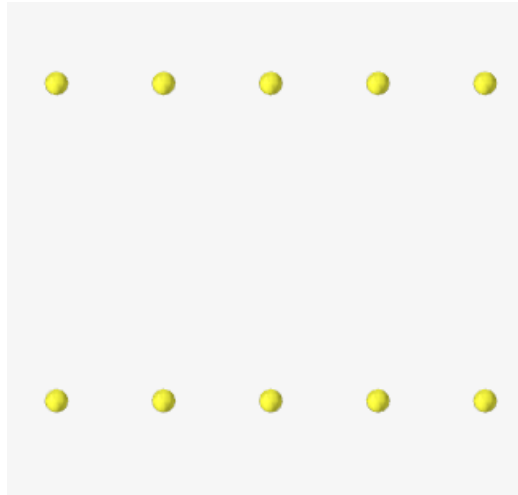
- 界面操作
- 打开command文件，获取相关命令
- 编写相关命令
- 测试

关闭右键视角变化命令记录

hm_writeviewcommands 0



Tcl vs Python



```
*rigid 1 101 123456
```

Tcl 命令

```
model = hm.Model()  
ent_1 = ent.Node(model,1)  
ent_2 = ent.Node(model,101)  
dofs = 123456  
hw_boollist_dofs = hm.hwBoolList(dofs)  
  
model.rigid(ent_1,ent_2,hw_boollist_dofs)
```

Python 命令

Tcl vs Python

- Tcl 命令: 关键字 + id/markid + 参数
- Python命令: 关键字 + entity对象/collection对象 + hw参数

Tcl 命令

```
*createentity comps includeid=0 name=component1
```

Python 命令

```
import hm  
import hm.entities as ent  
model = hm.Model()  
ent.Component(model, name='component1')
```

Tcl vs Python Command and Function

- Data Names

HM 中数据类型

- HM Tcl GUI command

HM GUI 交互相关

- HM Tcl Modify Commands

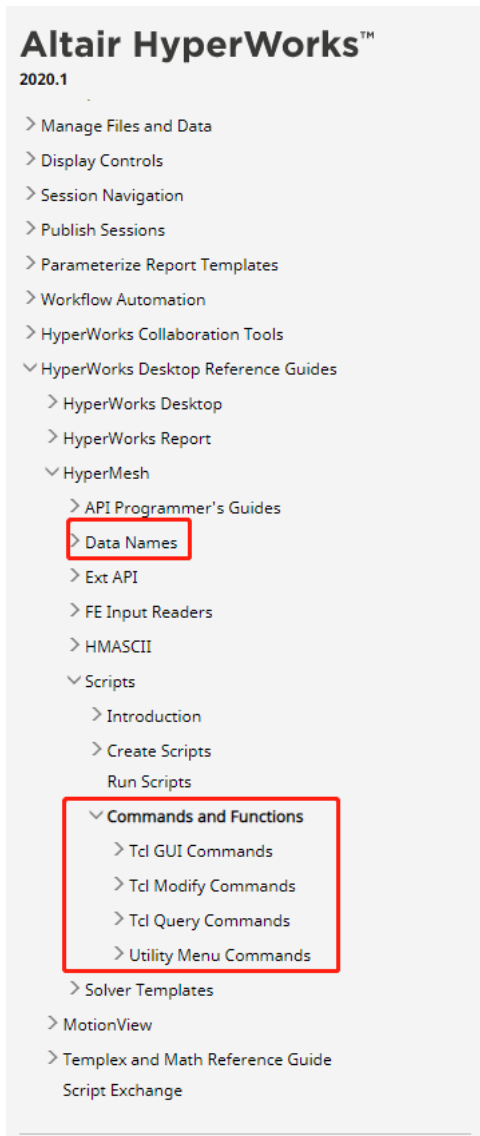
修改 HM内数据

- Hm Tcl Query Commands

获取HM内数据

- Utility Menu Commands

宏按钮等相关



- hm.entities

HM 中数据类型

- Frame command

HM GUI 交互相关

- Model Class Modify Commands

修改 HM内数据

- Model Class Query Commands

获取HM内数据

- ~~Utility Menu Commands~~

~~宏按钮等相关~~

Tcl vs Python Command and Function

内存容器命令

hm_createmark

hm_getmark

hm_marklength

*createlist

hm_getlist

HyperMesh实体交互命令

*createmarkpanel

*createlistpanel

核心命令

hm_getvalue

*setvalue

To delete components with names FRONT and SIDE:

```
hm_createmark comps 1 "FRONT SIDE"  
*deletemark comps 1
```

To delete components with names "Name with spaces" and SIDE:

```
hm_createmark comps 1 "{Name with spaces} SIDE"  
*deletemark comps 1
```

or

```
set names [list {Name with spaces} SIDE]  
hm_createmark comps 1 $names  
*deletemark comps 1
```

To mark the last 3 components that were created:

```
hm_createmark comps 1 "by id only" "-1 -2 -3"
```

To delete all elements in the database:

```
hm_createmark elems 1 "advanced" "all"  
*deletemark elems 1
```

To delete displayed elements:

```
hm_createmark elems 1 "advanced" "displayed"  
*deletemark elems 1
```

Python 对应关系

内存容器命令

hm.entities

hm.Collection

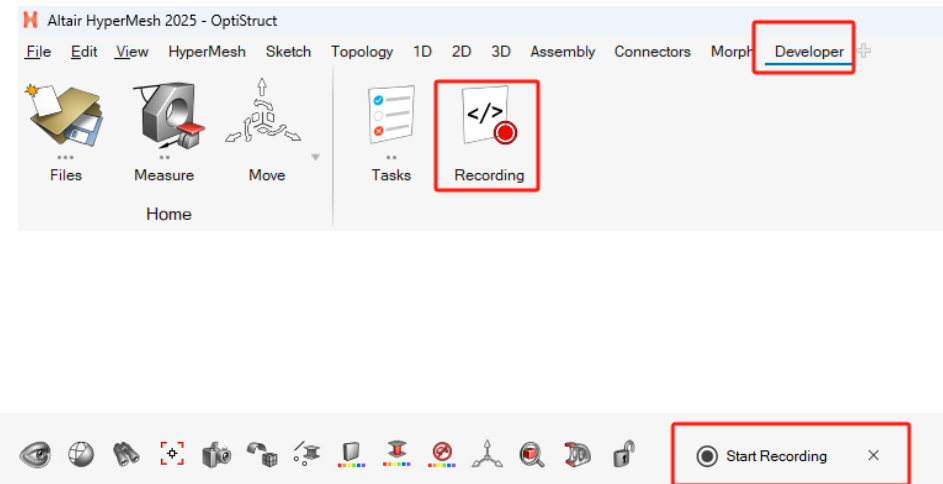
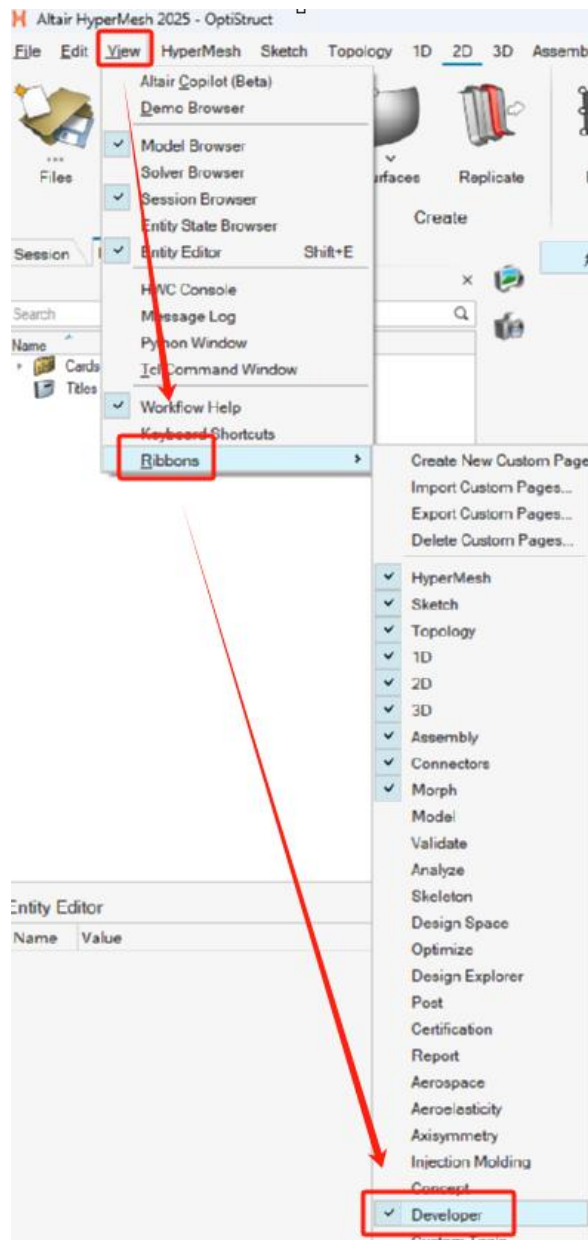
HyperMesh实体交互命令

hm.CollectionByInteractiveSelection()

二次开发基本流程

HyperWork2025中获取Python命令记录

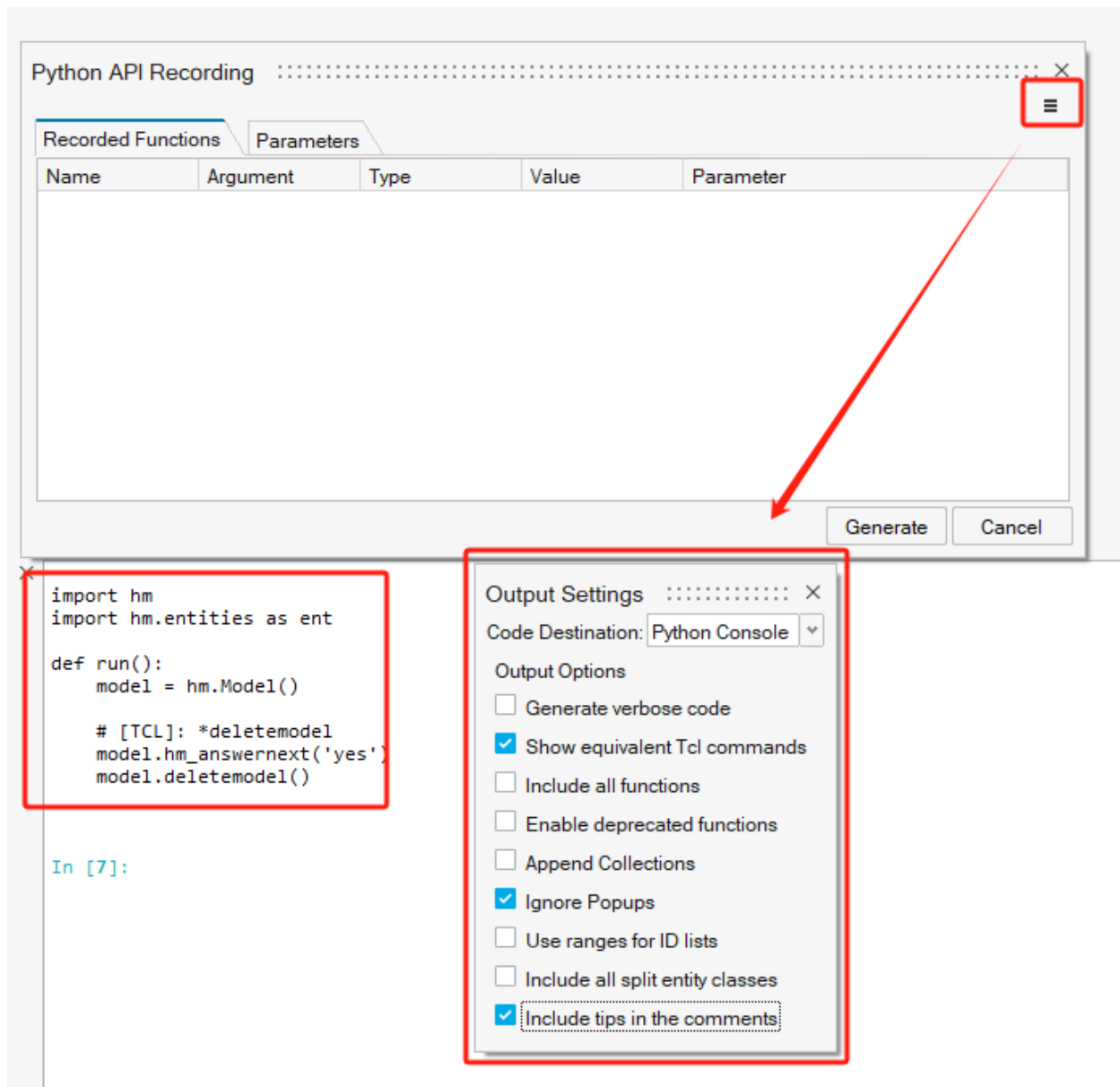
1. 在菜单栏
View>Ribbons>Developer
2. 在Developer中点击Recoding
3. 点击下方状态栏中的Start
Recoding
4. 执行界面操作，点击End
Recoding
5. 弹出Python API Recoding窗口
6. 点击Generate将在底部的
python window生成命令



二次开发基本流程

HyperWork2025中获取Python命令记录

1. 在菜单栏
View>Ribbons>Developer
2. 在Developer中点击Recoding
3. 点击下方状态栏中的Start
Recoding
4. 执行界面操作，点击End
Recoding
5. 弹出Python API Recoding窗口
6. 点击Generate将在底部的
python window生成命令



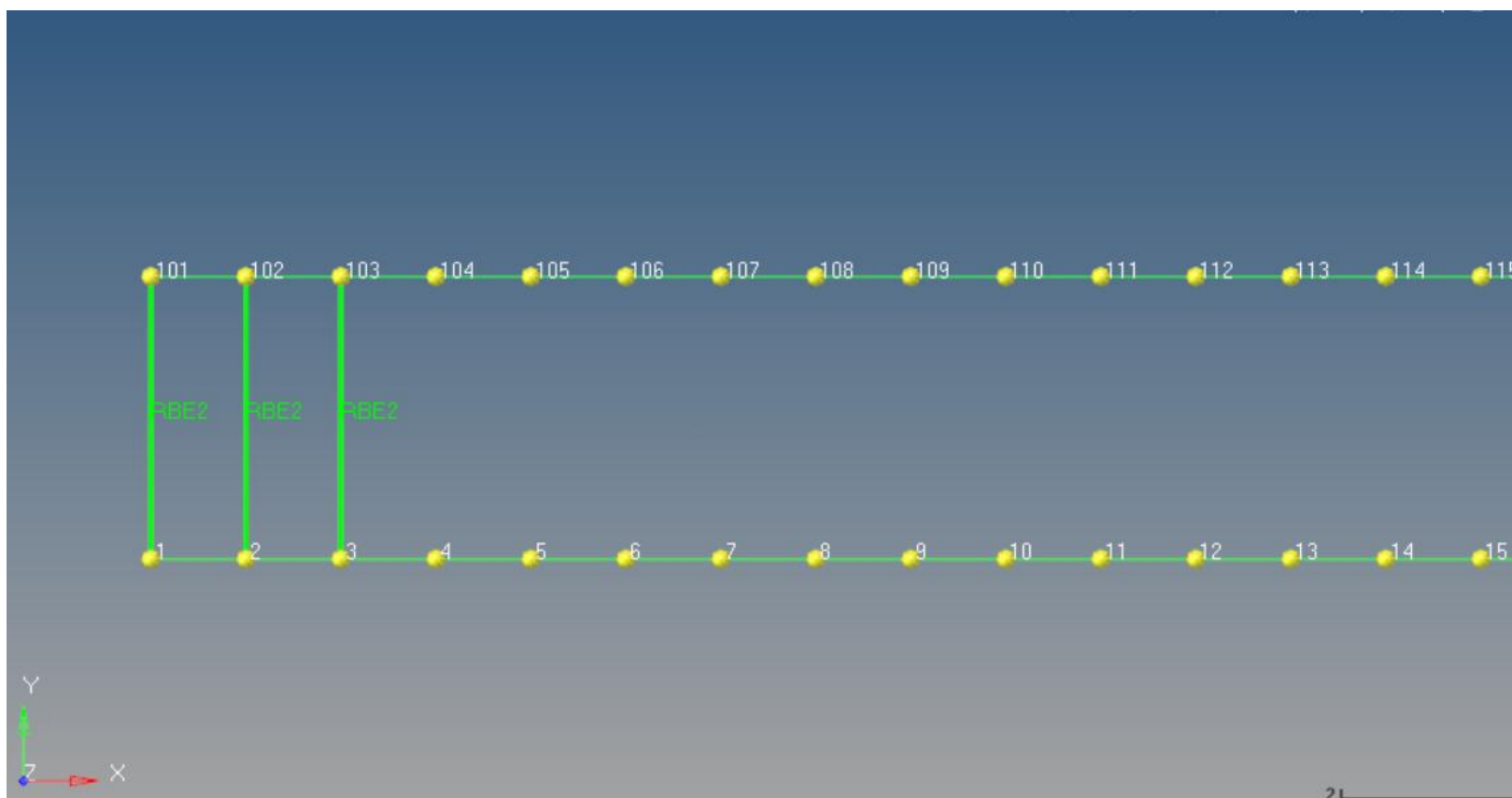
二次开发基本流程

HyperWork2025中获取Python命令记录

英文原文	选项	说明
Code destination	代码输出位置	定义生成代码的输出路径。
Generate verbose code	生成详细代码	为每个参数创建独立变量以展开代码。
Show equivalent Tcl commands	显示等效Tcl命令	以注释形式包含等效的Tcl命令。
Include all functions	包含所有功能函数	包含视图、撤销/重做等类似功能函数。
Enable deprecated functions	启用废弃函数	创建调试模型对象以访问已废弃的函数。
Append collections	追加集合	在通过附着/相邻方式创建集合时采用追加模式。
Ignore popups	忽略弹窗	在可能触发弹窗的函数前插入 model.hm_answernext("yes") 指令。
Use ranges for ID lists	对ID列表使用范围表示	使用 range 函数压缩完整的ID列表。
Include all split entity classes	包含所有分割实体类	泛化代码以作用于所有分割实体类。
Include tips in the comments	在注释中包含提示	在注释中添加可用选项的相关提示。

实例一：巧用 for 循环（Day1）

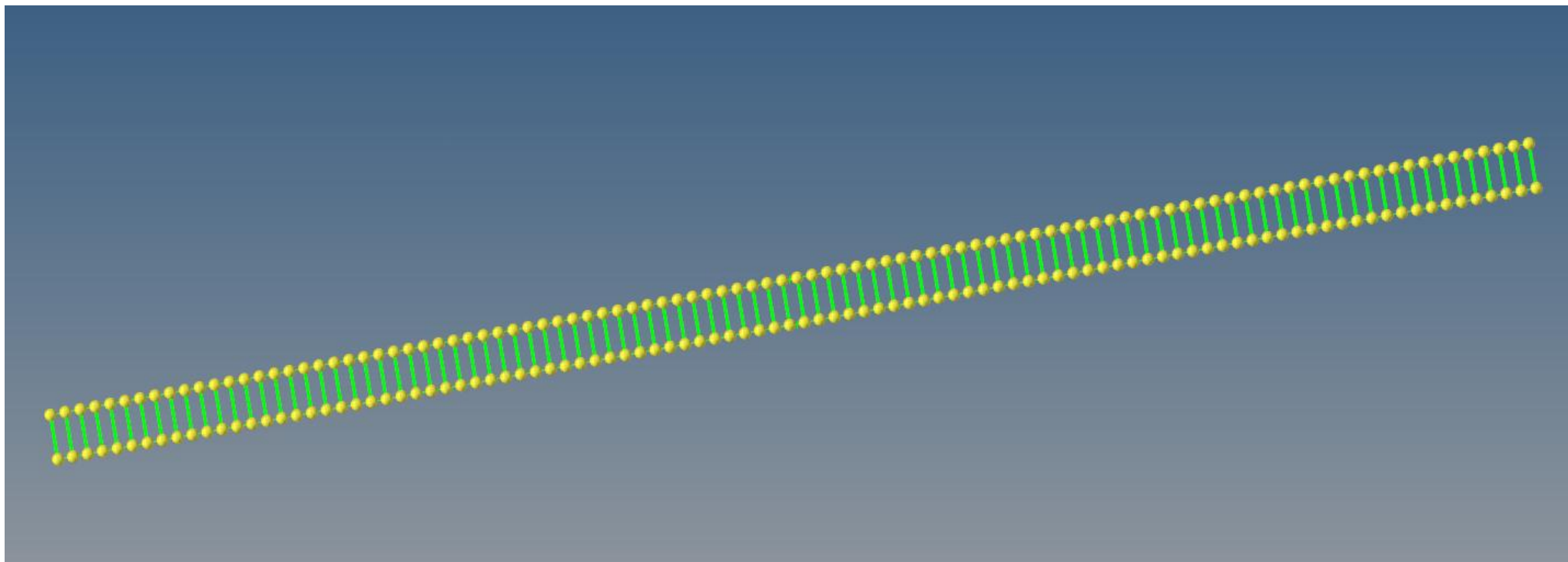
内存容器命令如果有两排分别100个节点，如何对这两排节点一一使用rigid进行快速焊接；
(提示：手动焊接两次，在command.tcl中找出规律)



实例一：巧用 for 循环（Day1）

解题思路：

- 1、在command.tcl中记录两条rigid的生成命令；
- 2、结合第一步的记录，使用for循环将这100个rigid单元生成



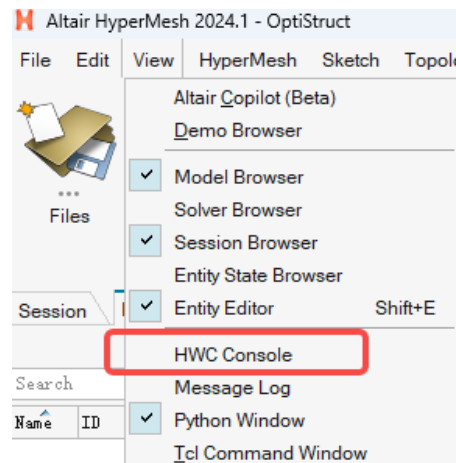
HyperView Python

HWC介绍:

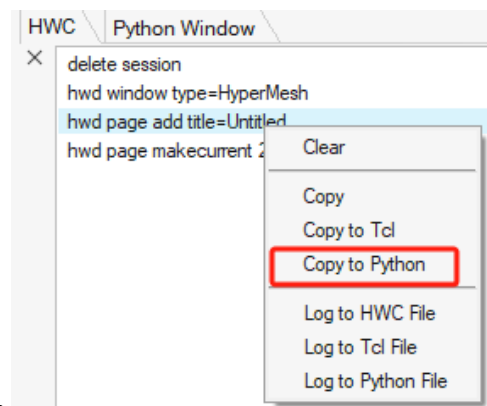
- 2019之后版本可以通过HWC命令实现图形界面的快速调整;
- 用户操作图形界面, 例: 载入结果, 加载云图, 创建note或者measure;
- 在HWC面板中将命令选中右键复制为tcl/python命令;
- 在tcl/python中运行HWC命令;
- 主要支持的功能: 载入结果, 加载云图, 创建注释, 视角调整, 截图, 组件/单元显示或隐藏。

HWC介绍:

- 在view标签中勾选HWC Console，底部命令框出现HWC Console；



- 在HWC面板中将命令选中，右键复制为python命令；



- 在python 中运行HWC命令；

```
In [1]: from hw import evalHWC as evalHWC  
In [2]: evalHWC('hwd page add title=Untitled')
```

HWC帮助文档:

Altair® HyperWorks®

2024.1

Search

Home / API, Reference Guides / HWC Console / HWC Commands

<

Error Messages

▼ HWC Commands

> animate

> annotation

> attribute

> color

> config

> delete

> explode

> hide

> hwd

> kpi hotspot

> mark

> open

> report

> result - HyperView - MultiCore

> result

> save

> scale

> section

> show

> sset

> system

View All Altair HyperWorks Help

HWC Commands

- animate

Animation controls.
- annotation

Annotation controls.
- attribute

Set entity attributes.
- color

Set the entity's color.
- config

Configuration options.
- delete

Delete content from the application.
- explode

Explode a model or selected components for improved visualization.
- hide

Hides all or a subset of entity types in the model.
- hwd

HyperWorks Desktop commands.
- kpi hotspot

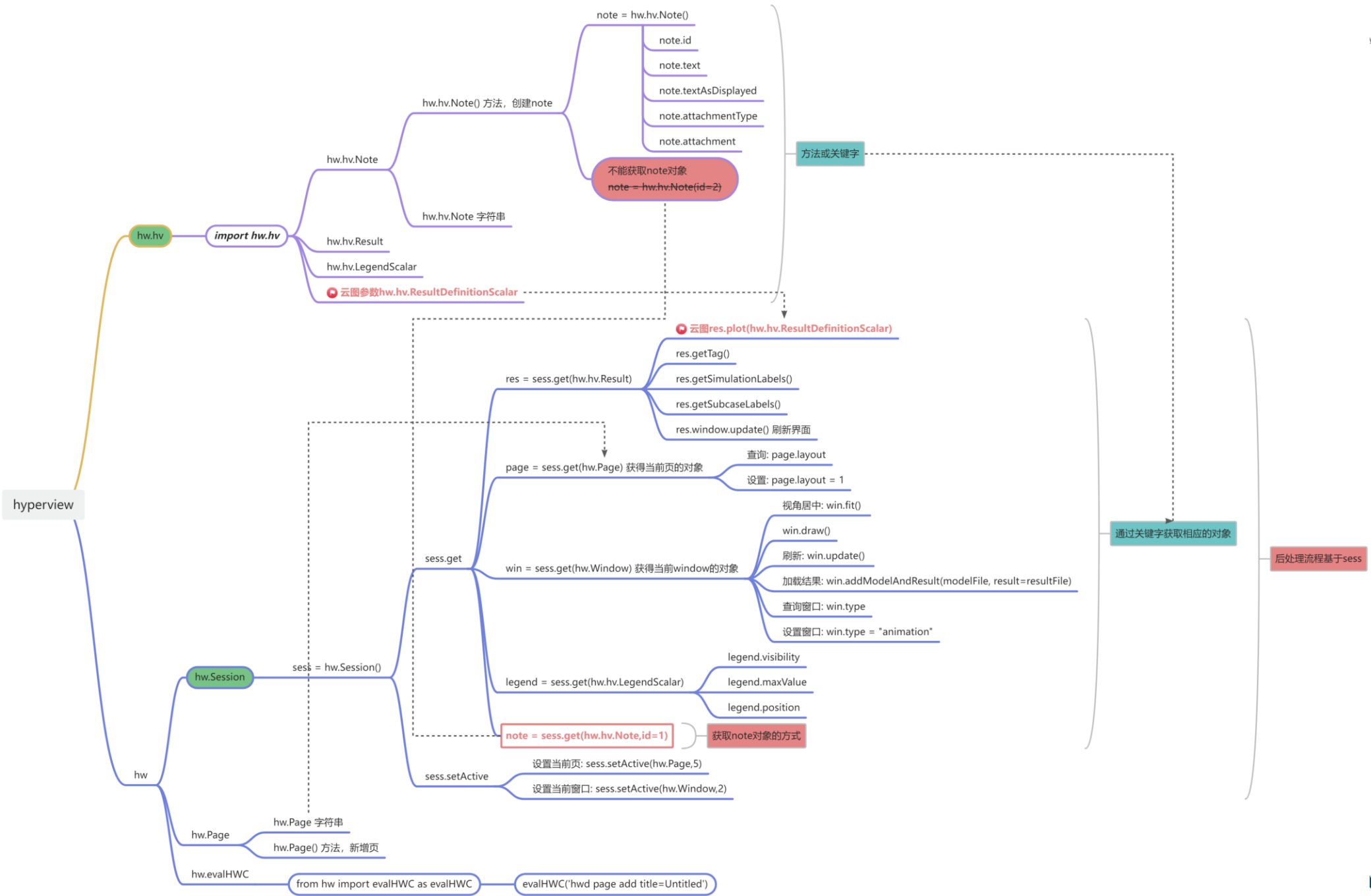
Hotspot finder commands.
- mark

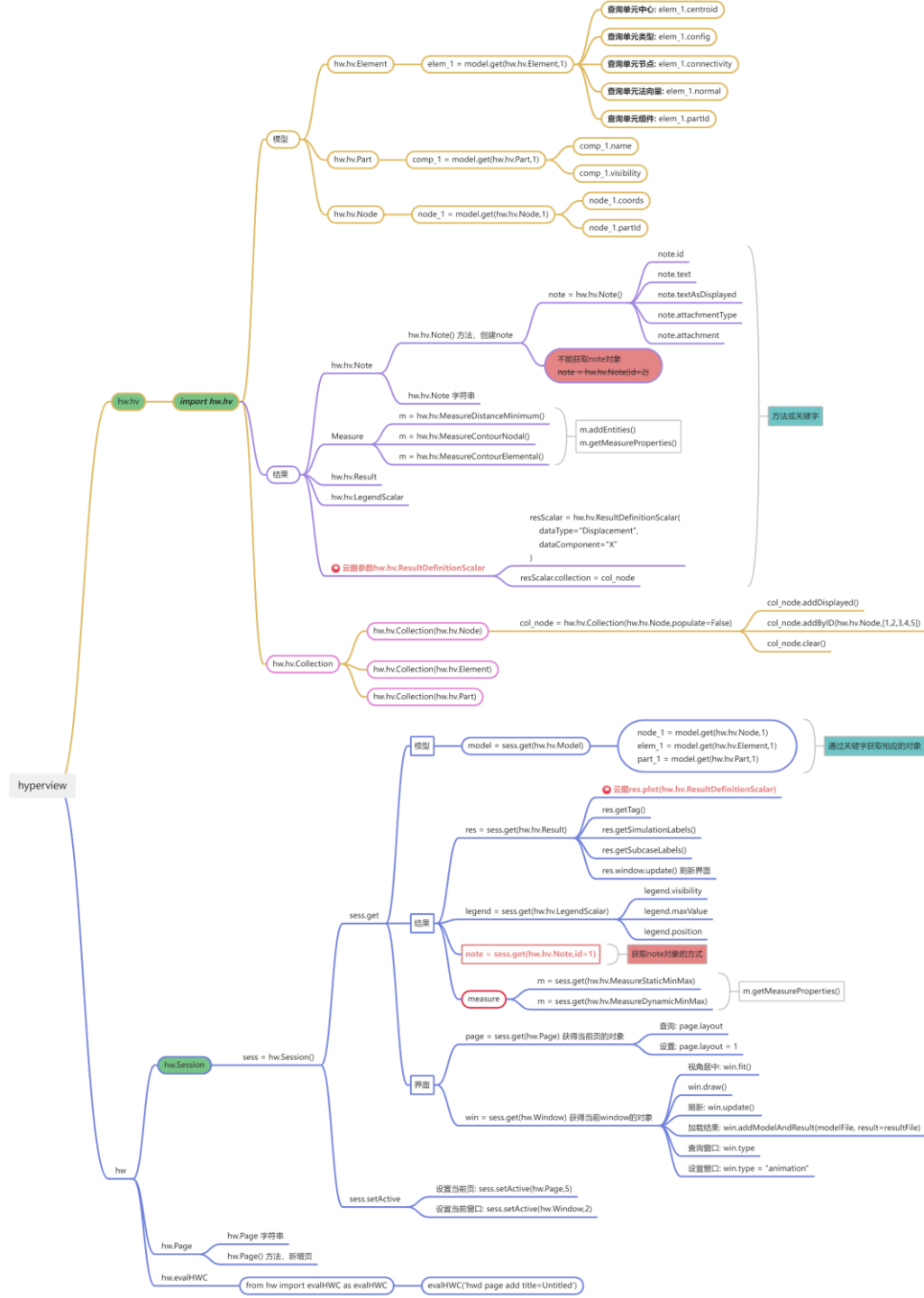
Marks entities on the active model.
- open

Open files.

后处理二次开发的流程思路:

1. **新建页面**，设置页面布局，调整窗口类型；
2. 加载结果文件；
3. Subcase, Simlotion以及Animation设置；
4. 云图设置；
5. 添加note/measure；
6. **读取数据**；
7. 截图；
8. 创建报告。





HyperView Python

- hw

 - hw.Page

 - hw.Session

 - hw.Window

- hm.hv

 - hm.hv.Note

 - hm.hv.Measure

 - hm.hv.Component

hw.Page

- hw.Page

不加括号，表示特定的字符串

- hw.Page()

表示方法，新增一页

- Tab键，可以显示出对象的方法和属性

```
In [38]: import hw
```

```
In [39]: hw.Page
```

```
Out[39]: hw.page.Page
```

```
In [40]: obj_new_page = hw.Page()
```

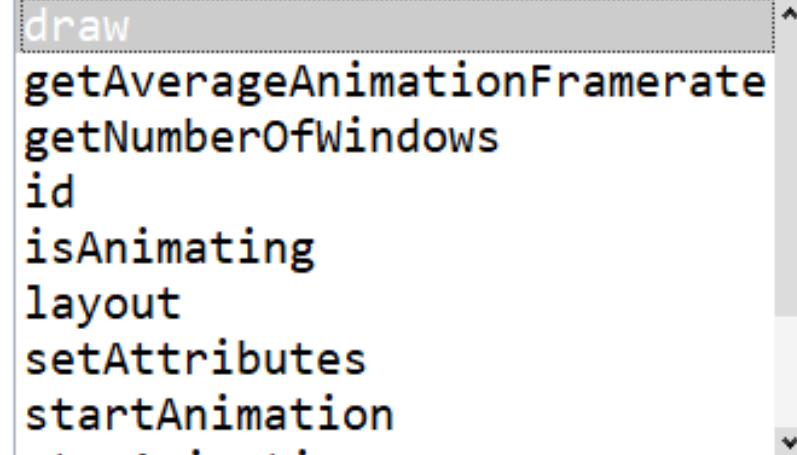
```
In [41]: obj_new_page.id
```

```
Out[41]: 4
```

```
In [42]: obj_new_page.getNumberOfWindows()
```

```
Out[42]: 1
```

```
In [44]: obj_new_page.
```



```
draw  
getAverageAnimationFramerate  
getNumberOfWindows  
id  
isAnimating  
layout  
setAttributes  
startAnimation
```

hw.Session

hw.Session与hm.Model类似，hyperview下的操作都是基于hw.Session

● 设置当前页

```
In [44]: sess = hw.Session()
```

```
In [45]: sess.setActive(hw.Page, page=4)
```

```
Out[45]: <hw.page.Page at 0x239cadda2b0>
```

```
In [46]: sess.setActive(hw.Page, id=1)
```

```
Out[46]: <hw.page.Page at 0x23a193b67c0>
```

● 获得page对象

```
In [47]: obj_page = sess.get(hw.Page)
```

```
In [48]: obj_page.id
```

```
Out[48]: 1
```

● 获得window对象

```
In [49]: obj_win = sess.get(hw.Window)
```

```
In [50]: obj_win.type
```

```
Out[50]: 'animation'
```



hw.Window

- hw.Window

不加括号，表示特定的字符串

- 一般与sess配合使用，通过sess设置当前window或者获得window对象

```
In [31]: import hw
```

```
In [32]: sess = hw.Session()
```

```
In [33]: hw.Window
```

```
Out[33]: hw.window.Window
```

```
In [34]: obj_win = sess.get(hw.Window)
```

```
In [35]: obj_win.id
```

```
Out[35]: 2
```

```
In [36]: obj_win.type
```

```
Out[36]: 'fevre'
```

```
In [37]: obj_win.type = 'animation'
```

hw.hv

- hw.hv.Note

不加括号，表示特定的字符串

- hw.hv.Note()

表示方法，新增Note

- 通过sess获得Note对象

```
In [37]: hw.hv.Note
Out[37]: hw.hv.note.Note

In [38]: obj_note = hw.hv.Note()

In [39]: obj_note.id
Out[39]: 2

In [40]: obj_note.label
Out[40]: 'note2'

In [41]: note_model_info = sess.get(hw.hv.Note,id=1)

In [42]: note_model_info.label
Out[42]: 'Model Info'

In [43]: note_model_info.textAsDisplayed
Out[43]: '1: C:\\Users\\yqxiang\\Desktop\\3dmodel.h3d (Err. 0.01%) (Err. 0.0

In [44]: note_model_info.text
Out[44]: '{for (i = 0; i != numpts(window.modeltitlelist); ++i) }\\n{window.m
```


hw.hv

- hw.hv.Measure..
不加括号, 表示特定的字符串
- hw.hv.Measure..()
表示方法, 新增Measure
- 通过sess获得Measure对象

```
In [59]: hw.hv.MeasureContourNodal
Out[59]: hw.hv.measurecontournodal.MeasureContourNodal

In [60]: obj_mea = hw.hv.MeasureContourNodal()

In [61]: obj_mea.type
Out[61]: 'contourNodal'

In [62]: obj_mea.label
Out[62]: 'measure4'

In [63]: static_mea = sess.get(hw.hv.MeasureStaticMinMax,id=1)

In [64]: static_mea.label
Out[64]: 'Static MinMax Result'

In [65]: static_mea.getMeasureProperties()
Out[65]:
{'properties': ['<model_id>',
                '<part_pool>',
                '<part_id>',
                '<entity_pool>',
                '<entity_id>',
                '<value>'],
 'ids': ['Min', 'Max'],
 'Min': [1, 'Part', 2, 'Node', 108, -15.53]}
```

HyperView Python 函数封装

```
In [1]: import hw
```

```
In [2]: def AddNewPage():
...:     sess = hw.Session()
...:     new_page = hw.Page()
...:     new_page_id = new_page.id
...:     sess.setActive(hw.Page, page=new_page_id)
...:     return new_page_id
...:
```

```
In [3]: AddNewPage()
```

```
Out[3]: 2
```

```
In [4]: import hw
...: import hw.hv
...: from hw import evalHWC as evalHWC
```

```
In [5]: def UpdataWin():
...:     sess = hw.Session()
...:     res = sess.get(hw.hv.Result)
...:     win = res.window
...:     win.update()
...:
...:     def GetNodeValueByNote(node_id):
...:         note = hw.hv.Note()
...:         note_id = note.id
...:         note.attachmentType = "Node"
...:         note.attachment = [1, 'node', node_id]
...:         note.text = '{entity.contour_val}'
...:         UpdataWin()
...:         value = note.textAsDisplayed
...:         evalHWC(f'annotation note delete note{note_id}')
...:         return value
...:
```

```
In [6]: GetNodeValueByNote(1)
```

```
Out[6]: '1298.24'
```

实例二： Python代码自动加载h3d结果 (Day2)

使用Python API, 在hyperworks自动载入h3d结果, 并加载位移云图, 提取位移最大点的id以及对应的位移;

(提示: 打开hwc window, 先手动完成h3d文件位移云图的加载, 拿到hwc python 代码, 然后优化代码)

实例二： Python代码自动加载h3d结果 (Day2)

使用Python API, 在hyperworks自动载入h3d结果, 并加载位移云图, 提取位移最大点的id以及对应的位移;

(提示: 打开hwc window, 先手动完成h3d文件位移云图的加载, 拿到hwc python 代码, 然后优化代码)

```
× delete session  
hwd window type=HyperMesh  
hwd window type = HyperMesh  
hwd page add title=Untitled  
hwd page makecurrent 2  
hwd page current activewindow=1  
hwd window type=HyperView  
open animation modelandresult G:/Secondary_Development_Trai  
result scalar load type=Displacement  
show legends  
animate transient time 0.1
```

HyperGraph Python

HyperGraph Python

- `hm.hg`

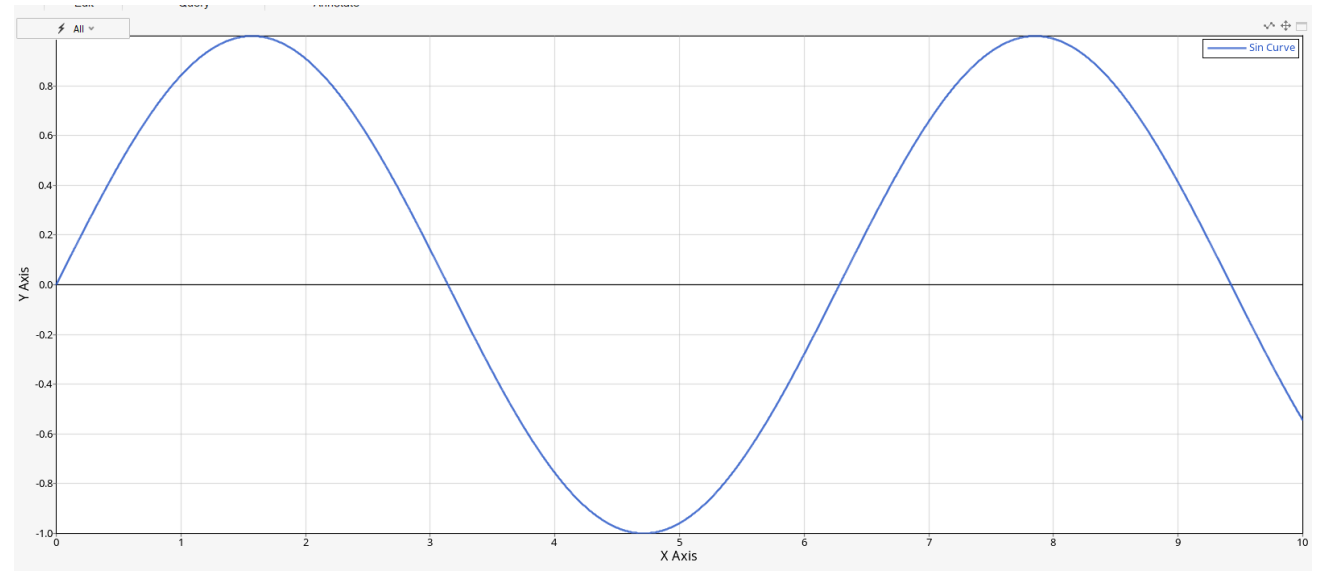
- `hm.hv.CurveXY`

- `hm.hv.Note`

hm.hv.CurveXY

● hm.hv.CurveXY (), 新增曲线

```
In [11]: import hw
...: import hw.hg
...: def PlotSinCurve():
...:
...:     # 新增一页
...:     AddNewPage()
...:
...:     # 创建sess
...:     sess = hw.Session()
...:
...:     # 获得win对象
...:     win_obj = sess.get(hw.Window)
...:
...:     # 调整win类型
...:     win_obj.type = "xy" ;
...:
...:     # sin curve
...:     sincurve_obj = hw.hg.CurveXY()
...:     sincurve_obj.xSource = "math"
...:     sincurve_obj.ySource = "math"
...:     expression_X = "0:10:0.1"
...:     expression_Y = "sin(x)"
...:     sincurve_obj.xExpression = expression_X
...:     sincurve_obj.yExpression = expression_Y
...:     sincurve_obj.label = "Sin Curve"
...:     win_obj.update()
...:
```



75 In [12]: PlotSinCurve()

hm.hv.CurveXY

- `hm.hv.CurveXY(id=1)` , 获得指定id曲线的对象

```
In [89]: curve_1 = hw.hg.CurveXY(id=1)
```

```
In [90]: curve_1.xMinimum
```

```
Out[90]: 0.0
```

```
In [91]: curve_1.yValues
```

```
Out[91]:
```

```
(0.0,  
 0.0998334166468282,  
 0.198669330795061,  
 0.29552020666134,  
 0.389418342308651,  
 0.479425538604203,  
 0.564642473395035,  
 0.644217687237691,  
 0.717356090899523,  
 0.782306000000000)
```


hm.hg.Note

- hw.hg.Note
不加括号，表示特定的字符串
- hw.hg.Note()
表示方法，新增Note
- 通过sess获得Note对象

```
In [93]: hw.hg.Note
Out[93]: hw.hg.note.Note

In [94]: obj_note = hw.hg.Note()

In [95]: obj_note.id
Out[95]: 1

In [96]: obj_note.text
Out[96]: ''

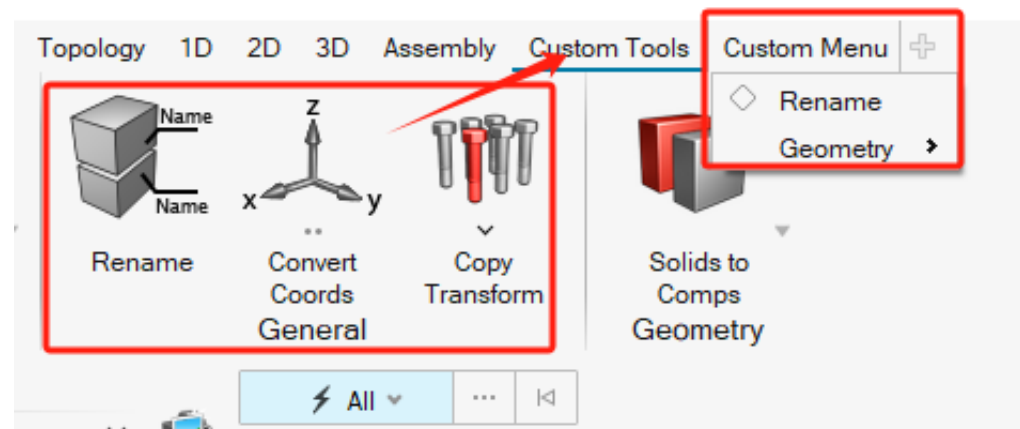
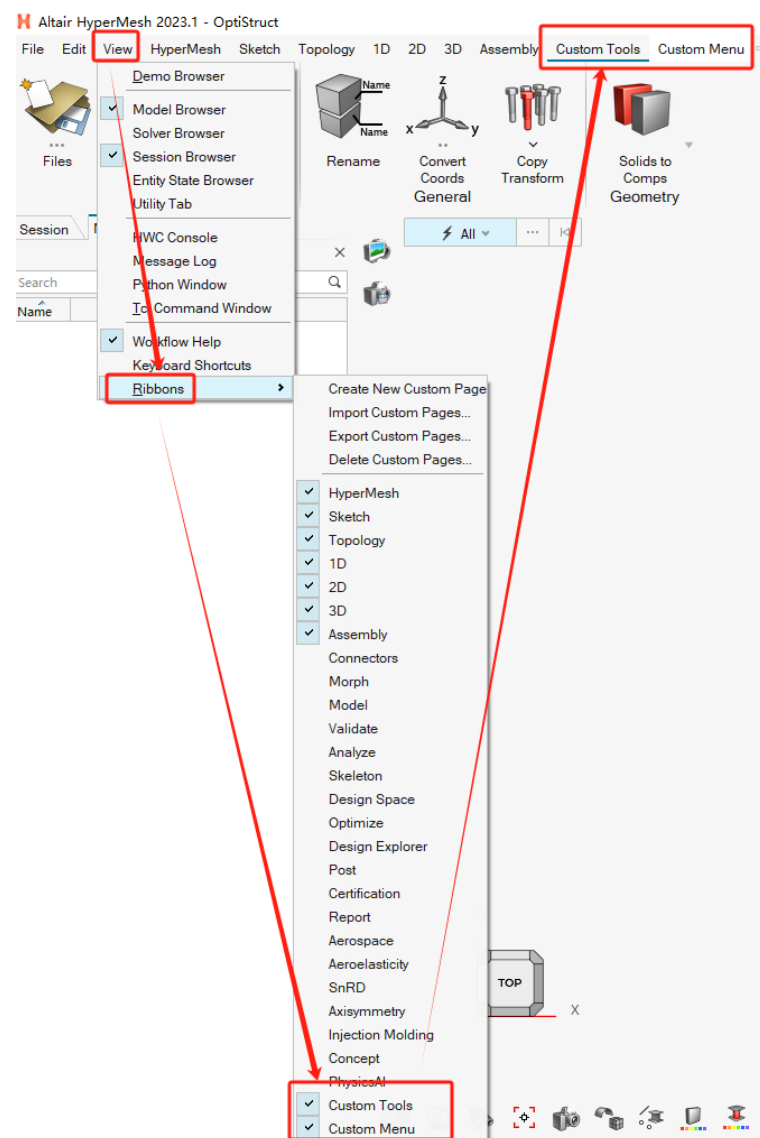
In [97]: obj_note.text = 111

In [98]: obj_note.positionX
Out[98]: 5.0

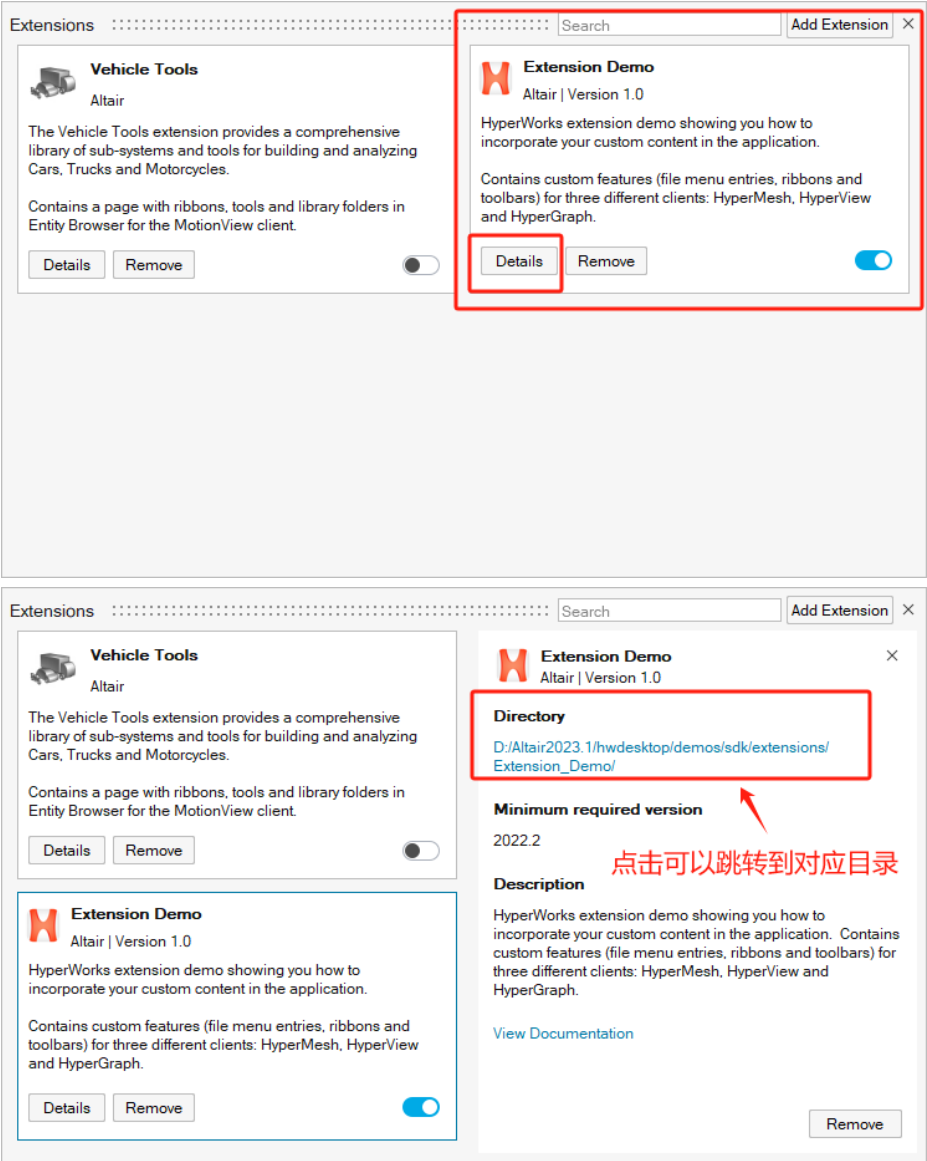
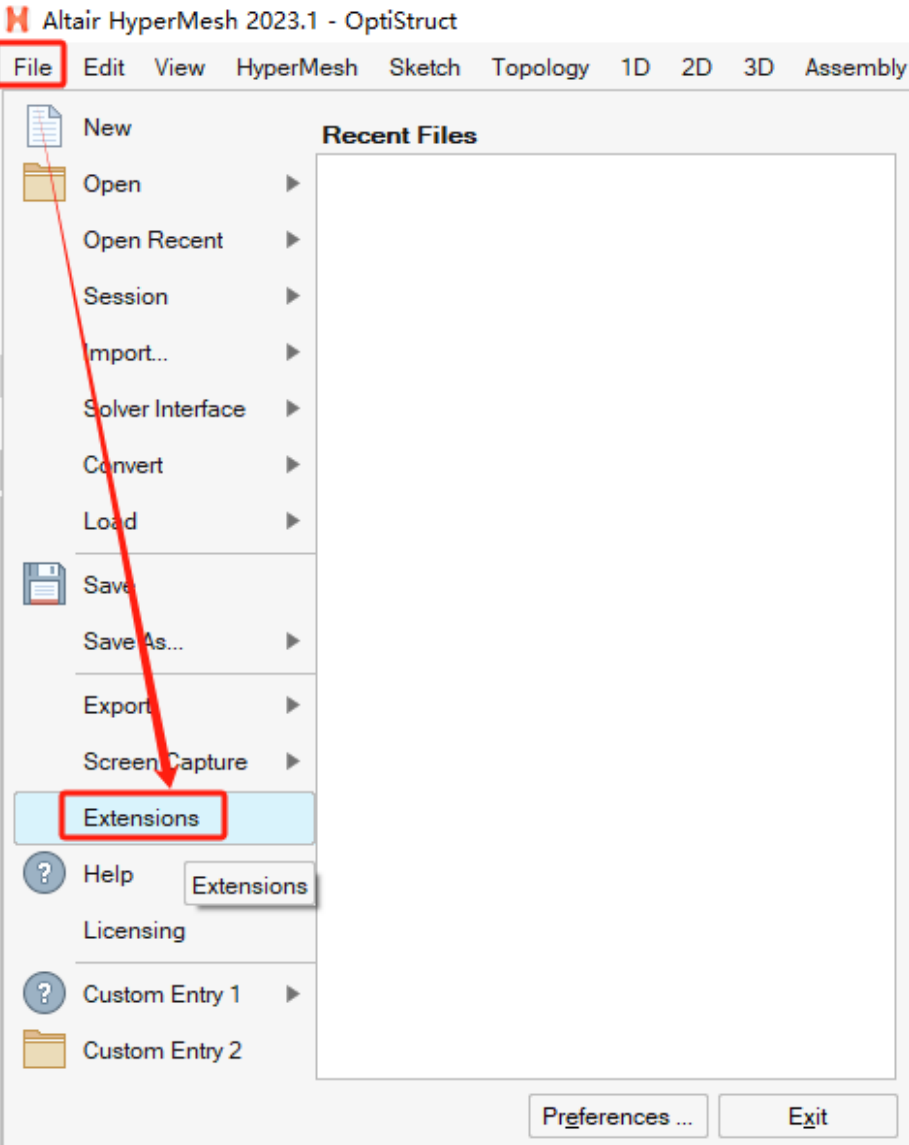
In [99]: obj_note.positionY
Out[99]: 0.006993006993007089
```

Ribbon

HyperWorks 功能区



Extension



Extension文件夹结构

- documentation文件夹，存放帮助文档
- hg, hypergraph界面相关的配置
- hm, hypermesh界面相关的配置
- hv, hyperview界面相关的配置
- images, 存放自定义的界面功能图标
- extension.xml, 必须修改的配置文件
- global-init.tcl, 自定义存放全局变量，按需修改

Altair2023.1 > hwdesktop > demos > sdk > extensions > Extension_Demo

名称	修改日期	类型	大小
documentation	2024/2/23 14:52	文件夹	
hg	2024/2/23 14:52	文件夹	
hm	2024/2/23 14:52	文件夹	
hv	2024/2/23 14:52	文件夹	
images	2024/2/23 14:52	文件夹	
extension.xml	2024/2/23 14:52	XML 文件	2 KB
global-init.tcl	2024/2/23 14:52	TCL 文件	2 KB

Extension.xml

The image shows the relationship between an `extension.xml` file and the Altair Extensions user interface. Red annotations explain key parts of the XML and their corresponding UI elements.

extension.xml Code:

```
<section name="Extension">
  <entry name="name" value="Extension Demo" />
  <entry name="displayName" value="Extension Demo" />
  <entry name="resources" value="images" />
  <entry name="minProductVersion" value="2022.2" />
  <entry name="version" value="1.0" />
  <entry name="author" value="Altair" />
  <entry name="description" value="HyperWorks extension demo showing you how to incorporate your custom content in the application.&#xD;&#xD;Contains custom features (file menu entries, ribbons and toolbars) for three different clients: HyperMesh, HyperView and HyperGraph." />
  <entry name="supportedClient" value="HyperWorksDesktop" />
  <entry name="documentation" value="documentation/ExtensionDemo.htm" />
  <entry name="tclscript" value="global-init.tcl" />

  <section name="profile" value="HyperMesh">
    <entry name="ribbonxml" value="hm/hm-ribbon.xml" />
    <entry name="tclscript" value="hm/hm-init.tcl" />
    <entry name="toolbars" value="hm/toolbars" />
    <entry name="contexts" value="hm/contexts" />
    <entry name="workflowhelp" value="hm/contexts/workflowhelp.xml" />
    <entry name="filemenu" value="hm/hm-filemenu.xml" />
  </section>

  <section name="profile" value="HyperView">
    <entry name="ribbonxml" value="hv/hv-ribbon.xml" />
    <entry name="tclscript" value="hv/hv-init.tcl" />
    <entry name="toolbars" value="hv/toolbars" />
    <entry name="filemenu" value="hv/hv-filemenu.xml" />
  </section>
</section>
```

Annotations:

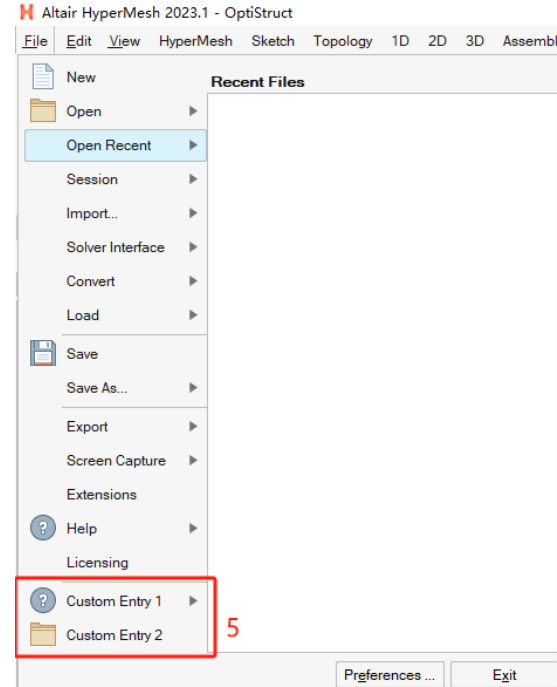
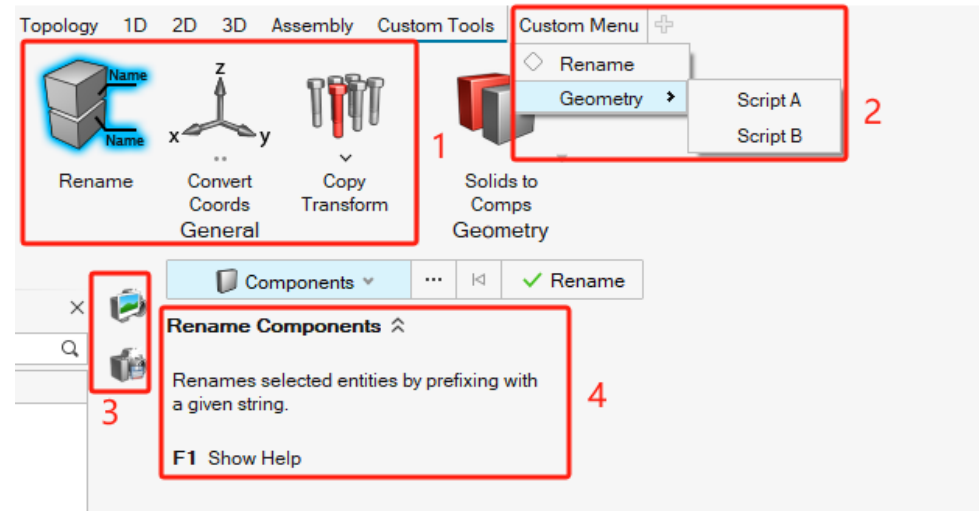
- 新创建的功能块名称, 必须修改** (Newly created function block name, must be modified): Points to the `name` and `displayName` attributes.
- 根据当前版本修改** (Modify according to the current version): Points to the `minProductVersion` attribute.
- 第一个运行的tcl脚本** (First running tcl script): Points to the `tclscript` attribute in the main section.
- hypermesh界面下的功能区** (功能区 - Function Area): Points to the `ribbonxml` attribute in the HyperMesh profile.
- hypermesh界面下的菜单** (菜单 - Menu): Points to the `filemenu` attribute in the HyperMesh profile.

Altair Extensions UI:

- The **Extension Demo** extension is shown with version 1.0.
- The description matches the XML: "HyperWorks extension demo showing you how to incorporate your custom content in the application. Contains custom features (file menu entries, ribbons and toolbars) for three different clients: HyperMesh, HyperView and HyperGraph."
- The extension is currently enabled (toggle switch is on).

Extension.xml

1. Custom HyperMesh Ribbon (defined in hm/hm-ribbon.xml)
2. Custom HyperMesh Menu (defined in hm/hm-ribbon.xml)
3. Custom HyperMesh Toolbar (defined in hm/toolbars/toolbar.xml), 没有可以删除该行
4. Custom HyperMesh Workflow Help for Custom Contexts (defined in hm/contexts/workflowhelp.xml), 没有可以删除该行
5. Custom HyperMesh File Menu Entries (defined in hm/hm-filemenu.xml), 没有可以删除该行



创建新的ribbon

新建一个ribbon时，需要将Extension_Demo文件夹下的内容全部复制到一个新的文件夹下，按需修改相应的文件

1.修改模块名(value="##"), 名称不能与已有的ribbon重名

```
<entry name="name" value="Extension Demo" />
```

2.修改版本号(value="##")

```
<entry name="minProductVersion" value="2022.2" />
```

3.根据需求调整hypermesh, hyperview, hypergraph中的相关文件

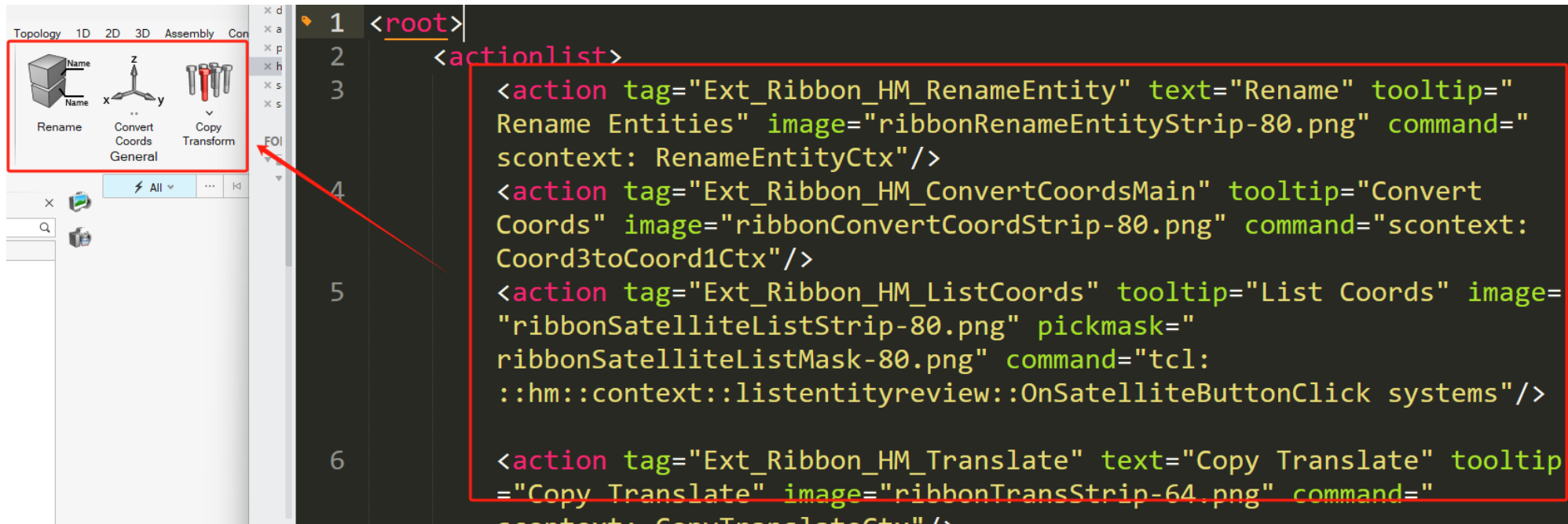
hm-ribbon.xml

对应图标和按钮设置

action tag与action actiontag中的名称对应;

tooltip为鼠标移动到图片显示的提示;

image为功能对应的图片, 只需写图片名称即可, 图片存放在images文件夹中;



hm-ribbon.xml

绑定python命令

注意command="py: \$EXTENSION_DIR"

1. \$EXTENSION_DIR指代的是extension目录;
2. 调用py中的命令, \$EXTENSION_DIR.hm.test.function(), 表明调用的是extension/hm/test.py 中的function函数;
3. 在test.py中, function的参数定义必须为(*args, **kwargs)

```
<root>
  <actionlist>
    <action tag="Ext_Ribbon_HM_RenameEntity" image="ribbonRenameEntityStrip-80.png" command="tcl: ::ExtensionDemo_Byyq::Test"/>
    <action tag="Test2_xyq" image="ribbonRenameEntityStrip-80.png" command="py: $EXTENSION_DIR.hm.test.function()"/>
    <action tag="Test3_xyq" image="ribbonRenameEntityStrip-80.png" command="py: $EXTENSION_DIR.hm.contexts.test.function()"/>
  </actionlist>
  <page tag="Ext_Ribbon_HM_Page1" text="Tools By yq">
    <group tag="Ext_Ribbon_HM_Group1" text="General">
      <action actiontag="Ext_Ribbon_HM_RenameEntity"/>
    </group>
    <group tag="Ext_Ribbon_HM_Group2" text="General">
      <action actiontag="Test2_xyq"/>
    </group>
    <group tag="Ext_Ribbon_HM_Group3" text="General">
      <action actiontag="Test3_xyq"/>
    </group>
  </page>
</root>
```

```
from hw import *
from hw.hv import *
from hw.xmlui import gui
from hw import gui as gui2
import os
```

```
def function(*args, **kwargs):
```

```
    def onClose(event):
        dialog.Hide()
```

```
    def onRun(event):
        dialog.Hide()
        gui2.tellUser("Done!")
```

实例三： Extension Demo (Day2)

基于Extension_Demo自定义Ribbon功能。

1. 定位模板文件

进入HW安装目录下的Extension_Demo文件夹，将其作为自定义功能的开发模板。

2. 功能配置

修改Extension.xml文件中的内容。

3. 命令绑定

新建py脚本，修改ribbon.xml文件。

4. 部署验证

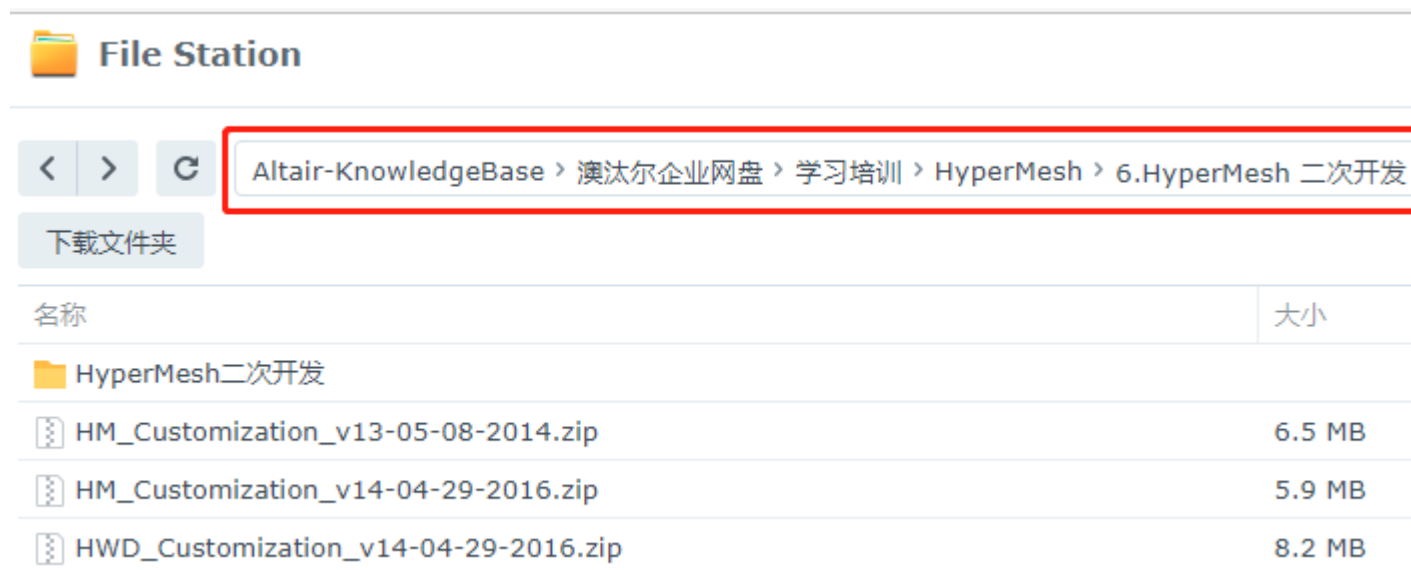
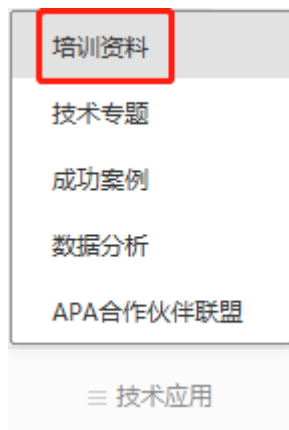
重启软件后检查Ribbon栏是否正常加载新功能。

学习方法

1. 关注微信公众号AltairChina并注册
2. 进入：技术应用 > 培训资料
3. 依次进入： 澳汰尔企业网盘 > 学习培训 > HyperMesh > 6.HyperMesh二次开发

问题咨询：

support@altair.com.cn



THANK YOU

更多资讯，欢迎关注Altair微信公众号



官网 : www.altair.com.cn

技术博客 : blog.altair.com.cn

邮箱 : info@altair.com.cn

技术服务热线: 400-619-6186