1 Abaqus介绍

2019年7月14日　　16:02

- 主要分析模块：ABAQUS/Standard, ABAQUS/Explicit, ABAQUS/CFD
- 参考资料：ABAQUS/CAE user's manual

ABAQUS/Standard：静态分析，动态分析，**结构的热响应分析**

典型的操作步骤：

1 创建部件Part

1.1 创建部件

1.2 绘制矩形

1.3 选择操作（拉伸、旋转、平移）

1.4 保存模型

2 创建材料和截面属性Property

2.1 创建材料

2.2 创建截面属性

2.3 给部件赋予截面属性

3 定义装配体Assembly

4 设置分析步Step

- Abaqus/CAE会自动创建一个初始分析步（initial step），用户必须自己创建后续分析步（analysis step），用来施加载荷

5 定义边界条件和载荷Load

5.1 施加载荷

5.2 定义悬臂梁左侧的固支约束

- 施加载荷和定义约束时单击鼠标选择平面，再点击鼠标中键即可弹出编辑对话框

6 划分网格Mesh

Note: 需要选择对部件beam划分网格，而不是为整个装配件划分网格，否则会出错

6.1 设置网格控制参数

- 选择单元形状

6.2 设置单元类型

6.3 设置种子

- 选择单元大小

6.4 划分网格

7 提交分析作业Job

7.1 创建作业分析

7.2 提交分析

8 后处理

- 开发ODB文件

8.1 显示未变形图

8.2 显示变形图

8.3 显示云纹图（彩色）

8.4 显示动画

8.5 显示节点的Mises应力值

- 点击"Query Information"
- 选择"Probe values"

8.6 查询节点位移

- Probe values改为Field output

快捷键：

ctrl+alt+左键：旋转

ctrl+alt+中键：平移

ctrl+alt+右键：缩放

Note: abaqus/CAE中不将材料属性直接赋予单元或几何实体，而是首先在截面属性中定义材料属性，再为每个部件赋予相应的截面属性。

- .cae: 数据库文件

Abaqus的脚本接口与通讯关系

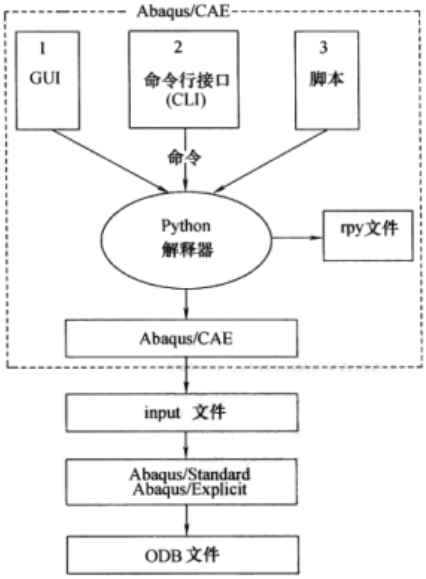


图 2-1 Abaqus 脚本接口与 Abaqus/CAE 的通信关系

Error: 脚本运行过程中可能会出现"Non-ASCII character..."的提示
问题的原因：程序中的编码错误，python默认是acii模式，没有支持utf8
解决办法：在源代码文件第一行添加#coding:utf-8，使可以输入中文字符

- Abaqus脚本接口在Python语言的基础之上进行了扩展，开发了Part, Property, Assembly等内核模块

1.1.2 运行python脚本

1.1.2.1 使用交互式命令行

Abaqus Command->输入'abaqus python'

1.1.2.2 执行脚本程序源文件

Remark: “线性分析”指外部载荷与系统的响应之间为线性关系，ABAQUS是“国际上最先进的大型通用非线性有限元分析软件”

1 非线性问题

- 非线性问题可以分为以下几种类型

（1）几何非线性 Geometric nonlinearity：位移的大小对结构的响应发生影响
（2）材料非线性 Material nonlinearity：材料的应力应变关系为非线性
（3）边界条件非线性 Boundary nonlinearity：边界条件在分析过程中发生变化。例如接触问题就是一种典型的边界条件非线性问题，它的特点是边界条件不能再计算的开始就可以全部给出，而是在计算过程中确定的，接触物体之间的接触面积和压力分布随外载荷而变化。

2 修改Abaqus的默认工作目录

- 右键Abaqus/CAE的快捷方式，修改工作目录

2 前处理模块

2019年7月26日　　9:15

所有部件在Part模块中建立，部件的草图在Sketch模块中创建，各部件在Assembly模块中组装。

2.1 部件模块（Part）和草图模块（Sketch）

- 直接建模&其它软件导入

2.1.1 创建部件

（1）Modeling space
- 3D
- 2D

（2）Options

（3）Type: 共有三种 deformable（任意形状的，在载荷条件下可以变形）, discrete rigid（任意形状的，在载荷条件下不可以变形）, analytical rigid（只可以用直线、圆弧和抛物线创建的形状，在载荷作用下不可以变形）

（4）Shape: solid

（5）Type建模方式：extrusion

2.1.2 导入部件

2.1.3 模型修复与修改

- Tools->Geometry edit->Convert to precise 转换为精确模型

2.2 特性模块（Property）

2.2.1 定义材料属性

- Material->Create
- Material behaviors:
  (1) General;
  (2) Mechanical: elastic弹性，hyperelastic超弹性，**expansion热膨胀系数**
  (3) Thermal热学特性
  (4) Other:

2.2.2 创建和分配截面特性

1 创建截面特性

Create section

（1）Solid实体：用于定义实体的截面特性

2 分配截面特性

2.3 装配模块（Assembly）

- 局部坐标系--整体坐标系进行装配

2.3.1 创建部件实体

- Instance->Create
- Dependent(mesh on part): 用于创建非独立的部件实体
- Independent(mesh on instance): 用于创建非独立的部件实体，是对原始部件的复制，需要对装配件中的每一个实体划分网格、
- Linear pattern 线性阵列模式

2.3.2 部件实体的定位

- 平移和旋转工具
- 约束定位工具

2.4 分析步模块

2.4.1 设置分析步

- Create->step 创建分析步，Step manager 步骤管理器
- 初始步：用户可以在初始步中设置边界条件和相互作用，使之在整个分析中起作用

1 Static general 静力学分析步

1.1 Basic

设置分析步时间与大变形

1.2 Incrementation

设置增量步

1.3 Other

设置求解器、求解技巧、载荷随时间变化

2.4.2 定义输出

2.5 载荷模块 Load

2.5.1 定义载荷

- Load->Create
- Category: mechanical, fluid, **thermal 热学的**
- Types for selected step:
  Thermal: surface heat flux 表面热通量，body heat flux 体热通量，concentrated heat flux 集中热通量

2.5.2 定义边界条件

- **Create boundary condition 创建边界条件**

2.5.3 设置预定义场

- Step: Initial
- Types for selected step: temperature

2.5.4 定义工况

- 载荷
- 边界条件

3 相互作用定义-需在创建分析步之后

2019年7月26日　　10:32

3.1 定义相互作用

- 进入该模块前，需要在Assembly功能模块中创建装配体并完成各部件实体的定位

3.1.1 接触属性的定义

- Interaction->Property->Create 创建接触属性

1 Contact property options 接触属性选项

- mechanical
- thermal

2 Data field 数据区

3.1.2 接触的定义

- Interaction->Create
- Types for selected step 选择相互作用的类型

1 定义Surface-to-surface contact
2 定义self-contact
3 定义Surface-to-surface contact (explicit step)
4 定义self-contact (explicit step)

3.1.3 接触控制的定义

- Interaction->Contact controls->Create

3.1.4 接触实例

例：刚性平压头与平板的接触设置
1 设置接触属性

- Create interaction property
- 创建General, Static分析步后，进入Interaction功能模块

2 创建主面和从面

- 在定义面面接触前，需要创建两个surface表面
- Tools->Surface->Create Create Surface
- 按照提示选择不同的表面

3 定义面-面接触

- Create Interaction 创建相互作用
- 在Types for selected type中选择surface-to-surface contact

3.2 定义约束

- Interaction->Create constraint
- 该模块中的Constraint是约束模型中各部分间的自由度
- 约束类型包括Tie绑定、Rigid Body刚体等

3.2.1 绑定约束

- 将模型中的两个区域（面或节点区域）绑定在一起，使他们之间没有相对运动。允许绑定两个网格不同的区域。
- 创建主面和从面：Tools->Surface->Create，分别选择创建约束的两个表面
- 选择绑定的主面和从面：Create Constraint，分别选择需要约束的两个表面

3.2.2 刚体约束

- 创建一个刚性区域（节点、面或单元），在整个分析过程中，该区域内节点和单元的相对位置保持不变。

3.2.3 耦合约束

3.3 定义连接器

- 用于连接模型装配体中位于两个不同的部件实体上的两个点，或连接模型装配件中的一个点和地面，来建立它们之间的运动约束关系。

例：圆压头和平板的接触实例

1 创建部件

1.1 创建圆片

1.构建1/4圆弧，设置圆片为analytical rigid part

2.指定刚体部件的参考点 Reference Point：用户必须为刚体部件指定一个参考点，刚体部件上的边界条件和载荷都要施加在这个参考点上。

Tools->Reference Point

1.2 创建平板

平板定义为Deformable柔性体，2D Plane二维平面，Base Feature: Shell壳体

2 创建截面和材料属性

- 选择柔性体部件，赋予截面属性

3 定义装配件

- 均选择默认参数，两个部件的位置都是正确的不需要重新定位。

4 划分网格

- 进入Mesh模块，将Approximate global size设置为1.5，单击Assign element type，选择Incompatible modes（非协调模式），即单元类型为CPS4I（4节点四边形双线性非协调轴对称单元）

5 设置分析步

本模型的分析步包含三个部分

1.初始分析步initial：定义边界条件

2.第一个分析步InContant：在圆片上施加一个很小的力10N，使各个接触部件平稳地建立起来

3.第二个分析步Apply Load：将圆片上的作用力改为6000N

Remark: 建议先定义一个很小载荷的分析步，让接触关系平稳地建立起来，再在下一个分析步中施加真实的载荷，这样可以减小收敛的困难，可能会减少计算时间。

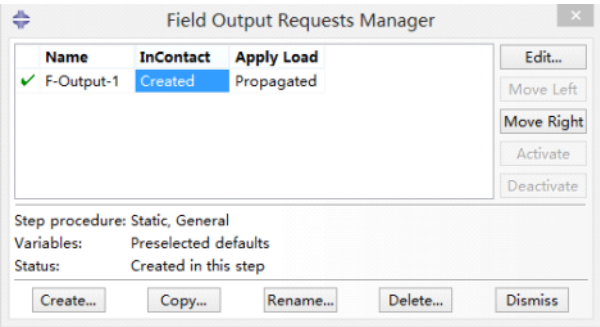
进入step功能模块：

1.创建第一个分析步InContact：类型为默认的Static, General；在Edit Step中，将Nlgeom（几何非线性）设为On

2.创建第二个分析步Apply Load：同上

3.为新的分析步InContact设置场变量输出结果：主菜单Output-Field Output Requests-Manager

4.为新的分析步InContact设置历史变量输出结果：主菜单Output-History Output Requests-Manager



6 定义接触

定义圆片和平板之间的接触

1.定义各个接触面：进入Interaction功能模块，主菜单中选择Tools-Surface-Manager

定义Surf-Plane，选择平板与圆弧接触的平面，类型为Geometry

定义Surf-Cylinder，选择刚体外侧对应的颜色（一般为黄色）

2.定义无摩擦的接触性质：Create Interaction Property，Nature后面输入InrProp-NoFriction。

3.定义接触：Create Interaction，选择主面（master surface）为Surf-Cylinder，选择从面（slave surface）为Surf-Plane。不改变默认的参数Sliding formulation: Finite sliding有限滑移

7 定义边界条件和载荷

Remark: 约束在X方向的位移U1和面内的转动UR3

1.创建集合。进入Load功能模块。主菜单选择Tools-Set-Manager，单击Create分别创建以下集合

1.1 圆片的参考点Set-Cylinder-Ref：单击选择圆片的参考点RP

1.2 集合Set-XY-Fix：选择平板的底面

1.3 集合Set-X-Fix：选择平板的左侧面

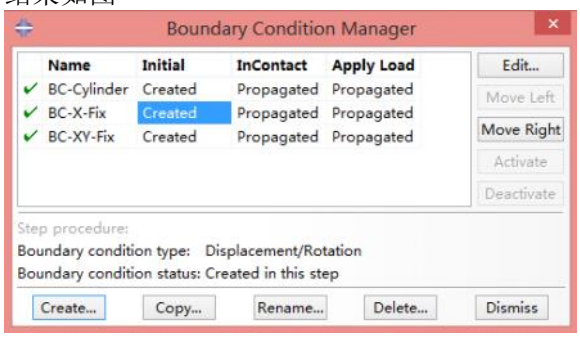
2.定义边界条件

- 主菜单选择BC-Manager，点击Create，Name后面输入BC-Cylinder，将Step设为Initial，将Type for Selected Step设为Displacement/Rotation，选中集合Set-Cylinder-Ref，在Edit Boundary Condition对话框中选中U1和

UR3

- 主菜单选择BC-Manager，再次点击Create，Name后面输入BC-XY-Fix，将Step设为Initial，将Type for Selected Step设为Displacement/Rotation，选中集合Set-XY-Fix，在Edit Boundary Condition中选中U1和U2
- 用同样的方法定义BC-X-Fix

结果如图



3.在第一个分析步中，在圆片上施加一个很小的力10N。

在Load Manager对话框中单击Create，Name后面输入Load-Cylinder，将Step设为InContact，选中集合Set-Cylinder-Ref，在Edit Load对话框中，在CF2后面输入-10。

Remark：CF2指Y轴方向的Concentrated Force，正值为Y轴正方向，负值为Y轴负方向。

4.在第二个分析步中，将圆片上施加的力改为6000N。在Load Manager对话框中，单击Load-Cylinder在第二个分析步Apply Load下面的propagated，单击Edit，将CF2的值改为-6000.

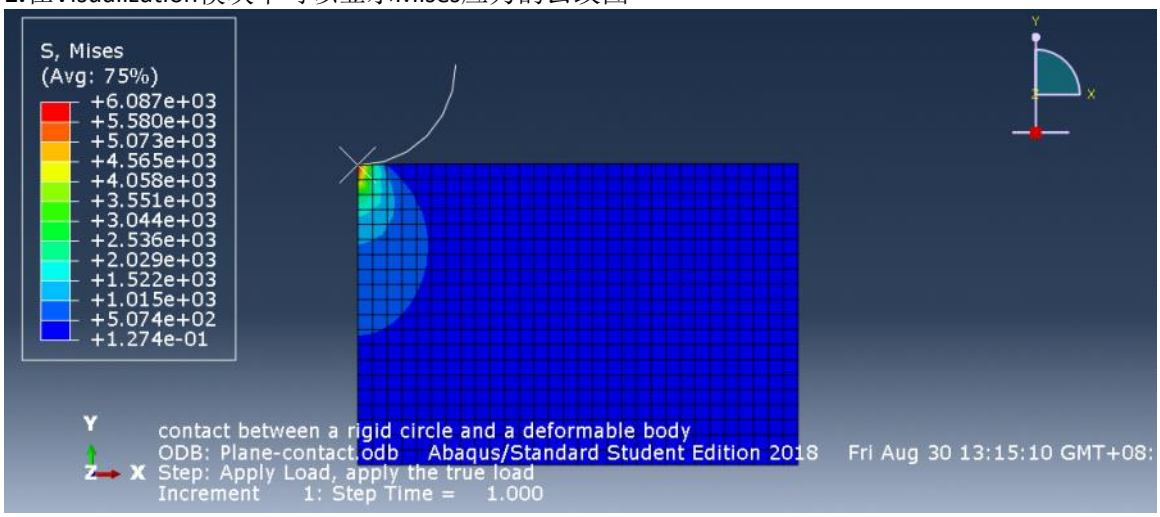
8 提交分析作业

进入Job模块，在主菜单中选择Job-Manager，创建名为Plane-contact的分析作业。单击Job Manager对话框中的Submit来提交分析。分析完成后，单击Results按钮，进入Visualization功能模块。

9 后处理

打开对应的ODB输出数据库文件

1.在Visualization模块中可以显示Mises应力的云纹图



2.显示接触压强CPRESS和接触状态COPEN

4 划分网格

2019年7月26日　　16:35

- 在创建部件（适用于非独立实体）或装配体（适用于独立实体）后，无论是否进行了Property, Interaction, Load等功能模块的设置工作，都可以在Mesh模块中进行网格划分，而无需按照模块的排列顺序一一处理。

4.1 定义网格密度

- 种子是单元的边节点在区域边界上的标记，它决定了网格的密度
- Seed菜单用于撒种子操作
- Seed->Part

例：非独立实体实例

1 设置部件种子

（1）Approximate global size: 输入大致的单元尺寸

（2）Curvature control: 控制曲边的种子设置

2 设置边种子

- 边种子总是优先于部件种子或实体种子

4.2 设置网格控制

- Mesh->Control

4.2.1 选择单元形状

4.2.2 选择网格划分技术设置

- Structured 结构化，Sweep 扫掠，Free 自由划分

4.3 设置单元类型

- Mesh->Element type

4.4 划分网格

4.5 检查网格

- Mesh->Verify

5 分析和后处理

2019年7月26日　　16:56

- Job+Visualization

5.1 分析作业模块 Job

- 在Part模块中创建部件，在Assembly模块中进行部件的装配
- 在进入Interaction模块和Load模块之前的任何时候，在Step模块中定义分析步和变量输出要求
- 在部件创建后（Part模块），Job模块之前的任何时候，都可以进入Property模块进行材料和截面属性的设置
- 如果在Assembly模块中创建的是非独立实体，则可以在创建部件后（Part模块），Job模块之前的任何时候，在Mesh模块中对部件进行网格划分；
- 如果在Assembly模块中创建的是独立实体，则可以在创建装配体后（Assembly模块），Job模块之前的任何时候，在Mesh模块中对装配体进行网格划分；

P128：独立实体&非独立实体功能模块的使用顺序

5.1.1 创建和管理分析作业

5.1.2 创建和管理网格自适应过程

5.2 可视化模块

1 显示无变形图和变形图
2 显示云图
3 显示X-Y图表
4 输出数据表格

6 热应力分析

2019年8月28日　　18:26

1 单个实体的热应力分析

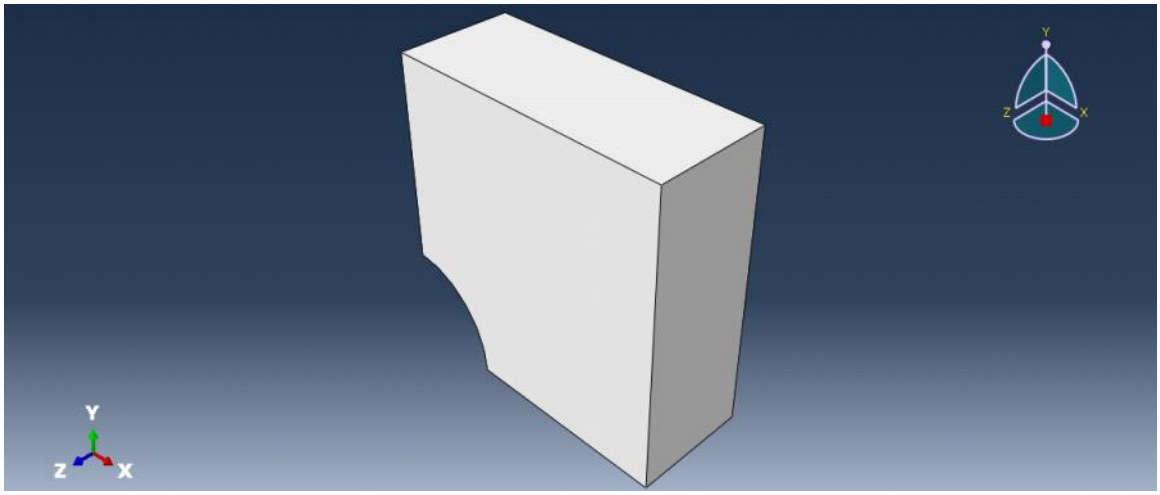
Key points:

1.在Material模块中，定义**线性膨胀系数**

2.在Load模块中，使用**预定义场（predefined field）**来定义温度场

步骤：

1.几何建模



2.定义材料属性-property

命名为Steel

Mechanical-Elastic-输入弹性模量和泊松比

Mechanical-Expansion-输入线性膨胀系数

3.创建和分配截面属性-property

Create Section

Assign Section

4.创建部件实体-assembly

5.设置分析步-step

6.定义边界条件-load

Create boundary condition

Fixed-Top

Fixed-X

Fixed-Y

ENCASTRE: 完全刚性固定

7.定义预设温度场-Create predefined field

Edit predefined field-20 初始温度为20度

(Initial-Created) 20

(Higher Tempereature-Modified) 120

8.划分网格-mesh

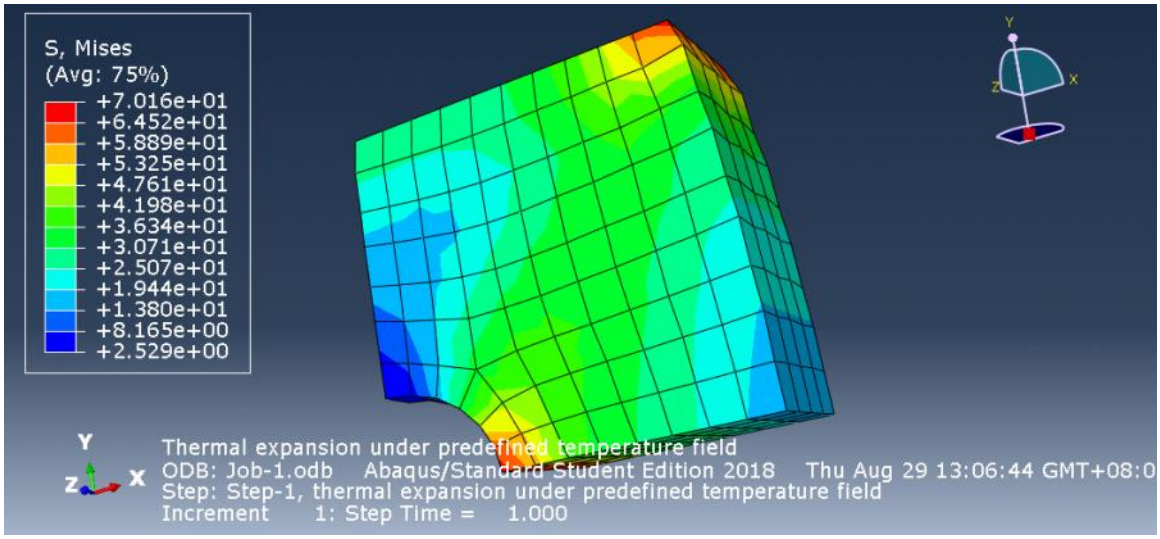
Seed part

Mesh part

9.提交分析作业-job

创建作业分析

提交分析 Job manager-submit



2 多个接触实体的热应力分析

步骤：

1.几何建模

创建两个相同尺寸的二维梁结构

plate-1和plate-2

2.定义材料属性，赋予截面属性

定义两种杨氏模量和泊松比相同，热膨胀系数不同(0.1, 0.3)的材料，分别命名为expand-1和expand-2

分别赋予两个梁结构不同的截面属性section-1和section-2

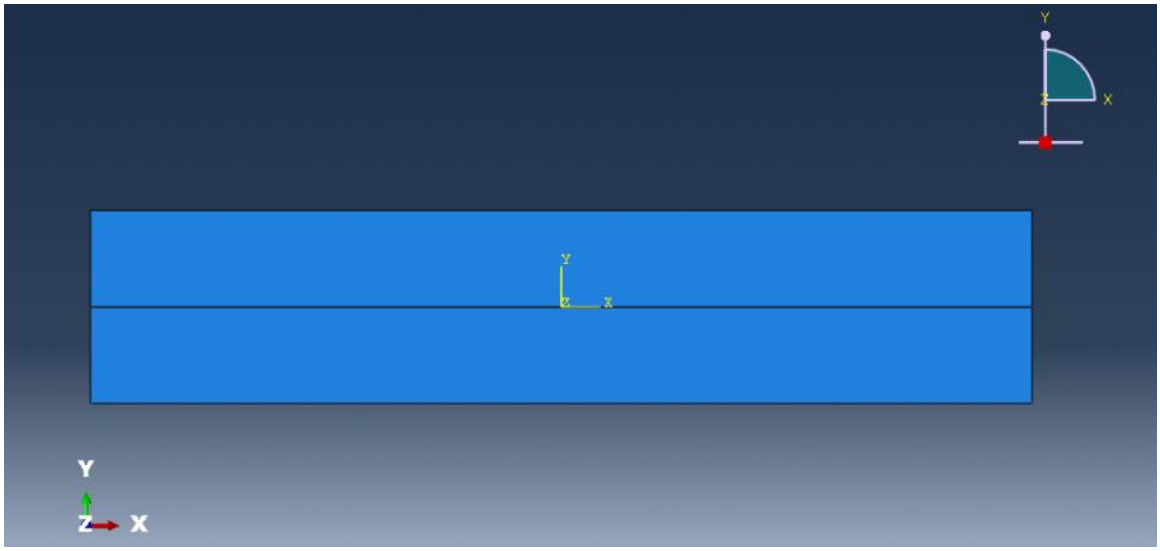
3.创建装配体-assembly

Create->Instance

Note: Linear pattern表示同一类型实体的线性阵列，不适用于不同截面属性的实体创建

Remark: 部件实体的定位和移动可采用平移和旋转

创建plate-1和plate-2的实体时，它们的位置重合，需要使用Assembly-Translate Instance



4.设置分析步-step
创建Step-1
5.定义边界条件-load
Create Boundary Conditions
对于同一装配体内的不同实体，需要分别定义边界条件
ENCASTRE: 完全刚性固定支撑边界条件
Remark: 若边界条件始终保持不变，则Boundary Condition Manager中定义为(Initial-空) (Step1-Created)
若边界条件随着时间推移产生变化，则Boundary Condition Manager中定义为(Initial-Created) (Step1-Propagated)
6.定义预设温度场-Create predefined field
Edit predefined field-20 初始温度为20度
(Initial-Created) 20
(Higher Tempereature-Modified) 120
7.定义相互作用关系
待补充
8.划分网格-mesh
Seed part
Mesh part
9.提交分析作业-job
创建作业分析
提交分析 Job manager-submit

7 多个接触实体的热应力分析

2019年8月30日　　14:09

1 几何实体建模-Part模块

创建两个尺寸相同的二维梁模型，长方形顶点坐标分别为(-10,1)(10,0)和(-10,0)(10,-1)，两个接触实体的空间关系已经定位好。两个实体命名为Part-PlateUpper和Part-PlateLower

2 创建截面和材料属性-Property模块

Young's modulus: 210000
Possion's ratio: 0.3
Expansion: Part-PlateUpper: 0.3 Part-PlateLower: 0.1
创建Material-PlateUpper, Material-PlateLower
创建Section-PlateUpper, Section-PlateLower
选择相应的Part，将对应的Material赋予对应的Section

3 定义装配体

均采用默认参数

4 划分网格

Error: 对装配体划分网格时可能出现"Dependent instances……"的错误信息
解决办法：模型树中，选择Assembly-Instances，右键选择Make independent
Seed part instance-Mesh part instance 注意参数的选择

5 设置分析步

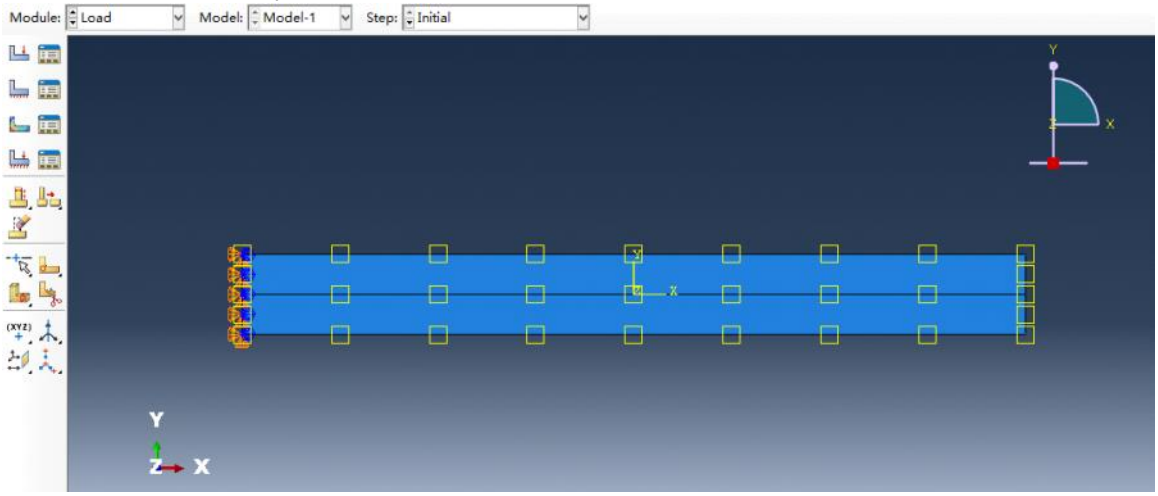
Step-ThermalExpansion

6 定义接触

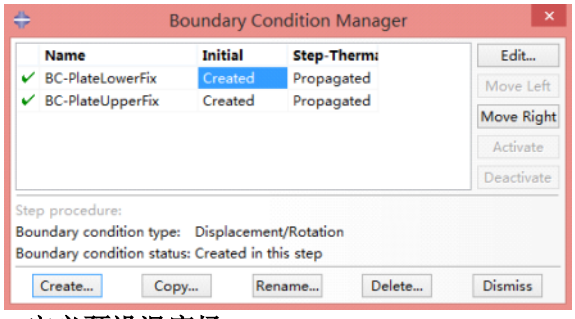
1.定义接触面：进入Interaction功能模块，主菜单中选择Tools-Surface-Manager
分别定义upper plate的下表面和lower plate的上表面
可在Surface Manager中查看Surf-PlateUpper和Surf-PlateLower
2.定义接触性质：
Interaction-Create constraints-Type: tie 两个上下表面贴合，约束模型之间的自由度

7 定义边界条件和载荷

1.创建集合：进入Load功能模块。主菜单选择Tools-Set-Manager，单击Create分别创建以下集合
Set-PlateUpper：upper plate的左端面
Set-PlateLower：lower plate的左端面



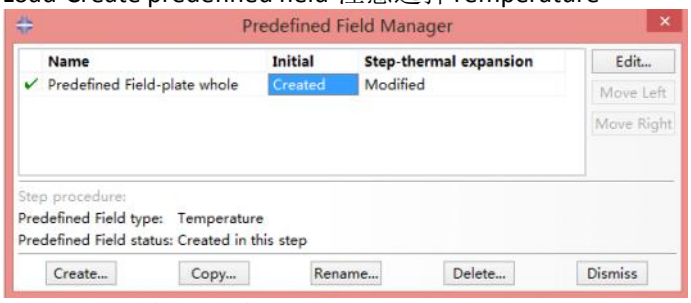
2.定义边界条件
主菜单选择BC-Manager，Step应设置为Initial
Set-PlateUpper和Set-PlateLower均设置为全约束：U1, UR3



3.定义预设温度场

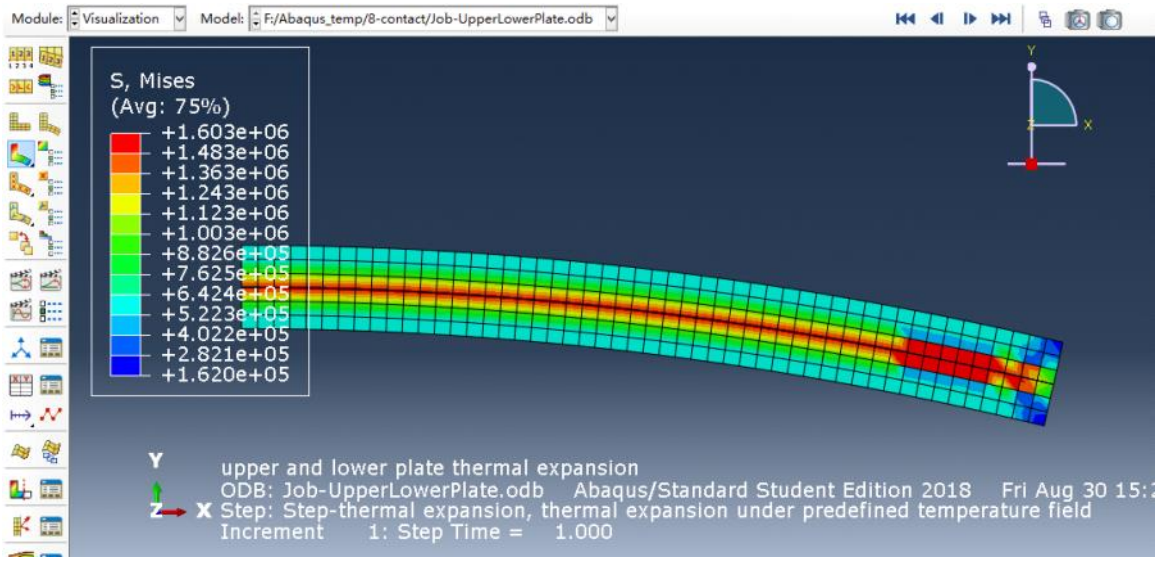
Tools-Set-Manager 创建整个upper plate和lower plate区域的Set，命名为Set-plate whole

Load-Create predefined field-注意选择Temperature



Initial中定义初始温度20，Step-thermal expansion中定义最终温度120

8 提交分析作业和后处理



200 nodes: 1min10s

8 多个接触实体的热应力分析-Python语句汇总

2019年8月31日　　10:33

1 getSequenceFromMask(...)

28.12.1　getSequenceFromMask(...)

This method returns the object or objects in the MeshEdgeArray identified using the specified *mask*. When large number of objects are involved, this method is highly efficient.

Arguments

Required argument

mask

A String specifying the object or objects.

Optional arguments

None.

Return value

A MeshEdge object.

Exceptions

An exception occurs if the resulting sequence is empty.

Error: The mask results in an empty sequence

2 HomogeneousSolidSection(...) 创建Section对象

- Creates a HomogeneousSolidSection object

3 获取某个instance的指定face, edge, cell

在FaceArray, CellArray, EdgeArray中查找

edgeCenter = (-10.0, 0.5, 0.0)
edges1 = e1.findAt((edgeCenter,))
edges1 = (edges1,)
上述三行代码与
edges1 = e1.getSequenceFromMask(mask=('[#2]',),)
作用相同

Note: 将edgeCenter, edges1写为tuple的形式，即(edgeCenter,), (edges1,)

4 Abaqus中edges和faces的编号

以2D rectangle为对象，X轴正方向水平向右，Y轴正方向垂直向上。
上边、左边、下边、右边依次编号为#1、#2、#4、#3
当前面的编号为#1

9-多个接触实体的热变形分析-流程

2019年9月3日　　19:00

1.几何实体建模-Part模块
2.创建截面和材料属性-Property模块
3.定义装配体
4.划分网格
5.设置分析步
6.定义接触
7.定义边界条件和载荷
8.定义节点集 nodeSets（节点的定义基于某一个Instance，但是可在rootAssembly中直接访问）
节点集用于记录变形场
9.定义温度场
10.提交分析作业和后处理
11.读取数据库中的displacement信息

10 显示云图

2019年9月4日　　13:10

- 云图用于在模型上用颜色显示分析变量

1 设置云图显示选项

2 选择云图的场变量

- Result-Field Output

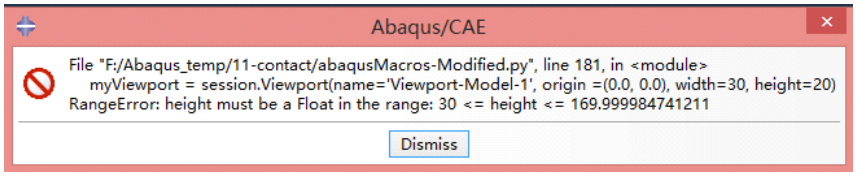
3 云图显示实例

1.取消视图区的标题区、状态区和三维视图方向的标识
Viewport-Viewport Annotation Options-General
取消选择：
Show triad（显示三维视图方向的标识）
Show title block（显示标题区）
Show text and arrows（显示文本和箭头）
Show compass（显示右上角指南针）
Viewport-Viewport Annotation Options-Legend中可定义关于Legend的相关属性

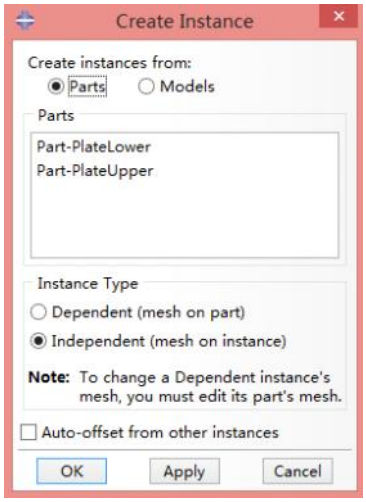
Error与Note合集

2019年9月1日　　15:59

1 关于Viewport的设置

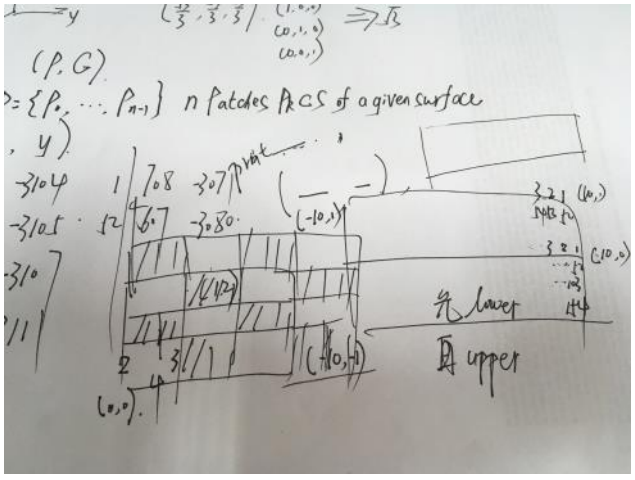


2 创建independent instance



a.Instance(name=instanceName, part=p, **dependent=OFF**)

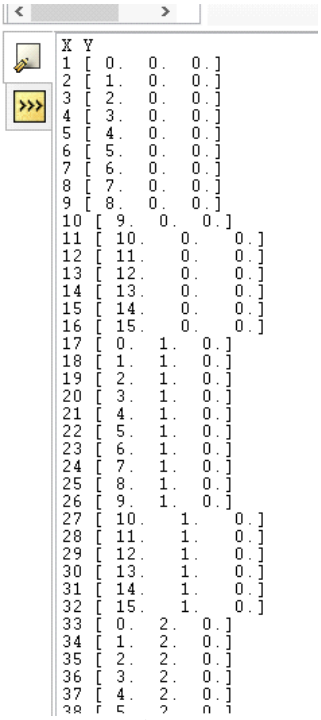
3 关于Section的定义



4 在输出数据库文件中，instances的keys的形式变为

```
The model database "F:\Abaqus_temp\13-MultiPlateContact\Model-1.cae" has been opened.
Warning: The database has been opened with readOnly flag on. It will remain readOnly.
X Y
Warning: The database has been opened with readOnly flag on. It will remain readOnly.
PART-PLATE-0-0-1
PART-PLATE-0-1-1
PART-PLATE-0-2-1
PART-PLATE-0-3-1
PART-PLATE-1-0-1
PART-PLATE-1-1-1
PART-PLATE-1-2-1
PART-PLATE-1-3-1
PART-PLATE-2-0-1
PART-PLATE-2-1-1
PART-PLATE-2-2-1
PART-PLATE-2-3-1
PART-PLATE-3-0-1
PART-PLATE-3-1-1
PART-PLATE-3-2-1
PART-PLATE-3-3-1
```

5 输出节点信息



```
   X Y
1 [ 0.   0.   0.]
2 [ 1.   0.   0.]
3 [ 2.   0.   0.]
4 [ 3.   0.   0.]
5 [ 4.   0.   0.]
6 [ 5.   0.   0.]
7 [ 6.   0.   0.]
8 [ 7.   0.   0.]
9 [ 8.   0.   0.]
10 [ 9.   0.   0.]
11 [ 10.    0.    0.]
12 [ 11.    0.    0.]
13 [ 12.    0.    0.]
14 [ 13.    0.    0.]
15 [ 14.    0.    0.]
16 [ 15.    0.    0.]
17 [ 0.   1.   0.]
18 [ 1.   1.   0.]
19 [ 2.   1.   0.]
20 [ 3.   1.   0.]
21 [ 4.   1.   0.]
22 [ 5.   1.   0.]
23 [ 6.   1.   0.]
24 [ 7.   1.   0.]
25 [ 8.   1.   0.]
26 [ 9.   1.   0.]
27 [ 10.    1.    0.]
28 [ 11.    1.    0.]
29 [ 12.    1.    0.]
30 [ 13.    1.    0.]
31 [ 14.    1.    0.]
32 [ 15.    1.    0.]
33 [ 0.   2.   0.]
34 [ 1.   2.   0.]
35 [ 2.   2.   0.]
36 [ 3.   2.   0.]
37 [ 4.   2.   0.]
```

- 节点输出的顺序和创建instance的顺序相同
- **文件1-OutputNode.py**

6 在step模块中，Initial不算为第一个分析步，Initial后面的分析步规定为第一个分析步

7 nodeSets所属数据结构

Access

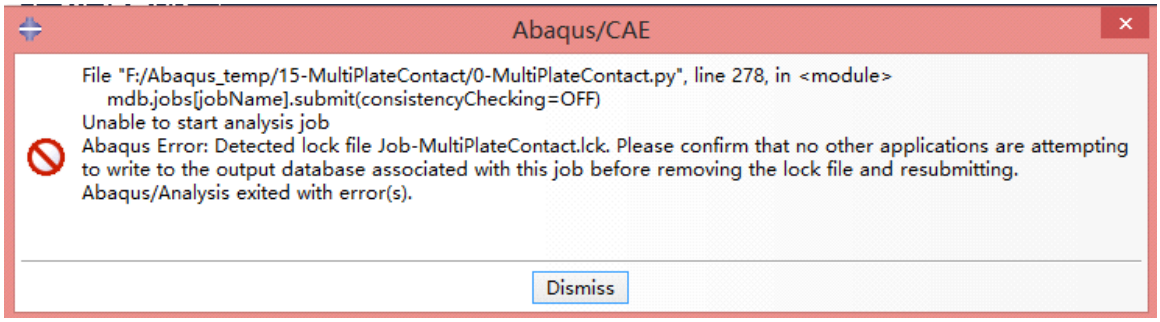
```
import odbAccess
session.odbs[name].parts[name].elements[i]
session.odbs[name].parts[name].elementSets[name].elements[i]
session.odbs[name].parts[name].nodeSets[name].elements[i]
session.odbs[name].parts[name].surfaces[name].elements[i]
session.odbs[name].rootAssembly.elements[i]
session.odbs[name].rootAssembly.elementSets[name].elements[i]
session.odbs[name].rootAssembly.instances[name].elements[i]
session.odbs[name].rootAssembly.instances[name].elementSets[name]\
.elements[i]
session.odbs[name].rootAssembly.instances[name].nodeSets[name]\
.elements[i]
session.odbs[name].rootAssembly.instances[name].surfaces[name]\
.elements[i]
```

8 规定原点为Fixed boundary condition

- **在abaqus中，对节点集nodeSets的定义是基于instances的**

```
# Obtain rootAssembly of the model
a = mdb.models[modelName].rootAssembly
instanceName = 'Part-Plate-0-0-1'
# Obtain the vertex set of the instance indicated by 'instanceName'
v1 = a.instances[instanceName].vertices
vertexCenter = (0.0, 0.0, 0.0)
# Find the target vertex by coodinates
v = v1.findAt((vertexCenter, ))
v = (v, )
setName = 'Set-Plate-Origin-Fix'
# Create a set representing the vertex
a.Set(vertices=v, name=setName)
region = a.sets[setName]
bcName = 'BC-Origin-Fix'
# Origin fix in x-direction, y-direction, xy-rotation
# Set boundary conditions for the vertex
mdb.models[modelName].DisplacementBC(name=bcName,
    createStepName='Initial', region=region, u1=SET, u2=SET, ur3=SET,
    amplitude=UNSET, distributionType=UNIFORM, fieldName='', localCsys=None)
```

9 数据库文件多次读写的问题



- 猜测：相同name的job不能提交分析多次

0 Python语言在Abaqus中的应用

2019年7月27日　　9:01

- **Abaqus的二次开发接口：**用户子程序（User subroutine），Abaqus脚本接口（Abaqus Scripting Interface，使用Python编写）
- 本书采用Python2.6

Abaqus6.10帮助文档

1. Abaqus Scripting User's Manual: 该手册详细介绍了Abaqus脚本接口的基础知识，Python脚本的开发环境、访问输出数据库的各种命令等。
2. Abaqus Scripting Reference Manual: 该手册详细介绍了Abaqus对象模型中所有的Python命令
3. Getting started with Abaqus: Interactive Edition: 附录中的Example files给出了部分实例脚本的源代码

1 Python语言基础

2019年7月29日 10:22

1.3 基础知识

1.3.1.4 模块导入

1 import sys
2 from … import …

1.3.3.1 变量

全局变量

声明全局变量：salary1 = 2000
引用全局变量：global salary1; salary1 = 2500
Note: 为了避免混淆全局变量和局部变量，可以将全局变量放到单独的文件中
例如，创建只保存全局变量的文件_global.py
salary1 = 2000
salary2 = 5000
使用定义的全局变量
import _global
_global.salary1
_global.salary2

局部变量

局部变量的生命周期仅限于函数或代码块范围

1.3.5 文件类型

- Python语言主要包括三种文件类型：源代码文件.py .pyw，字节代码文件.pyc，优化代码文件.pyo。这些代码文件无需编译或连接，可直接通过python.exe pythonw.exe解释执行

1.3.5.1 源代码文件.py .pyw

- .py由python.exe解释执行；.pyw用于开发图形用户界面，由pythonw.exe解释执行

1.3.5.2 字节代码文件

- 源代码文件经过编译后生成扩展名为.pyc的文件
- 字节代码文件与平台无关，可以在Windows, UNIX, Linux等操作系统上运行
- 运行源代码文件后就可以得到字节代码文件

1.4 内置数据结构

1.4.1 元组 tuple

tuple_name = (element1, element2, …)

1.4.1.2 元组的访问

- 索引[]和切边[1:5]

1.4.2 列表 list

1.4.3 字典 dictionary

1.5 结构化程序设计

1.5.1 条件语句 if…elif…else

1.5.2 循环语句 while, for … in

1.6.1 函数

- 递归函数，lambda函数

1.6.2 模块

- import module_name
- from module_name import function_name
- from module_name import *
- from module_name import function_name as short_alias_name

1.6.2.2 模块的属性

- 模块的属性__name__用来判断当前模块是否是程序的入口
- 如果正在运行当前的程序，则__name__的值为"__main__"
- 函数的三引号字符串存放在类的__doc__属性中

1.7 面向对象编程

- 根据函数或语句块来设计程序，称为面向过程编程
- 如果将数据和功能结合起来进行编程，则称为面向对象编程

1.7.2.1 类的属性

- 类的属性和实例属性

2 Abaqus脚本接口

2019年7月27日　　10:46

- Abaqus脚本接口直接与内核进行通信

编写脚本可以实现的功能

1 自动执行重复任务：如创建材料库

2 进行参数分析：编写脚本来实现逐步修改部件的几何尺寸或某个参数，然后提交分析作业，通过脚本来控制某个量的变化情况，如果达到某个指定要求，则停止分析、

3 创建和修改模型

4 访问输出数据库（ODB文件）

2.1 Abaqus脚本接口

2.1.1 Abaqus脚本接口与Abaqus/CAE的通信

- GUI， 命令行接口，脚本

2.1.2 命名空间

- 命名空间可以理解为程序的执行环境

1 脚本命名空间 script namespace

脚本接口命令的主要执行命名空间

（1）脚本；

（2）命令行接口；

（3）在Abaqus/CAE的File菜单下，选择Run Script

2 日志命名空间 journal namespace

- 从图形用户界面中发出的Abaqus脚本接口命令将在日志命名空间中执行
- 使用语句from abaqus import *将向脚本命名空间中导入mdb变量，此时可用它来访问对象
- 如果在使用对象的完整路径，并给出库的关键字，脚本命名空间仍然可以引用在日志命名空间中创建的对象

2.1.4 运行脚本的方式

1 启动Abaqus/CAE的同时运行脚本（Abaqus Command）

- abaqus cae script=simple_beam.py

Error: 可能提示ABAQUS error: "=" is not a valid argument

解决方式：右键以管理员身份运行Abaqus Command，再运行abaqus cae script=simple_beam.py

2 不启动Abaqus/CAE而直接运行脚本

- abaqus cae noGUI=simple_beam.py

3 从屏幕运行脚本

- 启动一个新任务时

4 从File菜单运行脚本

- 在Abaqus/CAE中建立模型时，工作目录下将自动生成abaqus.rpy文件，该文件记录了所有与Abaqus/CAE操作对应的命令

2.1.5 创建脚本的方法

1 在宏管理器中录制宏

2 借助abaqus.rpy文件

3 借助PythonReader.exe软件

【实例2-1】编写脚本实现旋转显示视口的功能

Note: 只有退出Abaqus/CAE之后，才可以打开abaqus.rpy文件

- 模型数据库.cae，输出数据库.odb

2.2 Abaqus脚本接口基础知识

2.2.1.2 访问对象

- 应手动导入**符号常数**模块

```
from abaqusConstants import *  #一般出现在脚本的第一行
```

- 访问Material对象

```
import material    #导入material模块，此时，material对象、方法和成员都变得可用
mdb.models[model_name].materials[material_name]    #必须使用mdb限定材料对象、命令或成员
```

- **Python支持为访问对象语句创建新变量的方法，避免代码过长**

```
sideLoadStep = session.odbs['Forming loads'].steps['Side load']
```

2.2.1.3 路径

- 运用构造函数创建对象
- 一般来说，构造函数的首字母为大写，其余字母小写；其他方法以小写字母开头
- 有些构造函数的路径可能多余一个，例如，编写脚本时既可以为Part对象创建基准，也可以为RootAssembly对象创建基准

2.2.1.4 参数

- 必选参数+可选参数
- 编写脚本时应尽可能选用关键字参数，这样参数的顺序可以任意

2.2.1.5 返回值

2.2.2 Abaqus脚本接口中的数据类型

2.2.2.1 符号常数

- QUAD, SAX2T表示单元类型，DEFORMABLE表示变形体，3D,2D表示维数
- e.g. a.setMeshControls(elemShape = QUAD) 设置单元形状为四边形

2.2.2.2 库

- 存储某一特定类型对象的容器
- e.g. mdb.models['Model-1'].parts 包含模型Model-1中的所有部件

2.2.2.3 数组

2.2.2.4 布尔值

- Abaqus Python中布尔值为ON和OFF

2.2.2.5 序列

2.2.3 面向对象编程与Abaqus脚本接口

- 创建对象的方法称为构造函数，有些对象不包含构造函数，此时创建的第一个对象将称为另一个对象的成员。
- e.g. session.viewports['Viewport-1'].setValues(width = 50) setValues(…)是Viewport对象的一种方法
- 使用构造函数Viewport创建了Viewport对象，将其值赋给myViewport
  myViewport = session.Viewport(name = 'newViewport', width = 100, height =100)

2.2.3.2 Abaqus脚本接口中的成员

- 使用界定符访问对象的成员
  myWidth = session.viewports['myViewport'].width
- 调用__members__方法可以列出对象的所有成员
  session.viewports['myViewport'].__member__
- **Abaqus对象的成员具有只读属性**，因此，不允许使用赋值语句指定成员的值，但是可以调用setValues(…)方法改变成员值
  shellSection.setValues(thickness = 2.0)
- type()函数可用于显示对象的类型
  print 'Section type = ', type(mySection)
- 列出对象的所有成员
  mySection.__members__
- 列出对象的所有方法
  mySection.__method__
- 创建对象后，还可以调用对象的某些方法来输入或修改数据。例如，可以调用addNodes和addElements方法为部件分别添加节点和单元

2.2.3.3 总结

- 构造函数：Abaqus脚本接口的惯例：构造函数名首字母大写，其它字母小写；一般方法首字母小写

2.2.4 异常和异常处理

1 InvalidNameError
2 RangeError
3 AbaqusError: 建模过程中的操作与前后设置的相关性
4 AbaqusException: 建模过程中的操作与前后设置的相关性

2.3 在Abaqus/CAE中使用脚本接口

2.3.1 Abaqus对象模型

例：通过Cell对象访问cells库中的4号（第5个）元素

cell4=mdb.models['block'].parts['crankcase'].cells[4]
将模型数据库block中部件名为crankcase的第5个元素存储在变量cell4中

例：导入Part模块和Section模块

import part
import section

例：导入Abaqus/CAE的所有模块

from caeModules import *

2.3.1.3 抽象基本类型

例：pressure压力和concentrated force集中力都是Load载荷，共享"Load"公共属性

2.3.1.4 查询对象模型

1 type()函数查询对象类型
2 object.__members__查询对象成员
3 object.__methods__查询对象方法
4 object.__doc__查询相关信息

Note: 只有在命令行接口CLI和Abaqus命令提示符下，才可以使用Tab键自动完成命令功能

2.3.2 复制和删除对象

2.3.2.1 复制对象

- 拷贝构造函数

例：复制函数格式

ObjectName(name = 'name', objectToCopy = objectToBeCopied) 将返回名为name，且与objectToBeCopied类型相同的对象

Remark: 可以调用拷贝构造函数为不同模型创建新对象

```
firstBolt = mdb.models['Metric'].Part(name = 'boltPattern', dimensionality = THREE_D, type =
DEFORMABLE_BODY)
secondBolt = mdb.models['SAE'].Part(name = 'boltPattern', objectToCopy = firstBolt)
```

不同model中可以含有相同名称的对象，上例中分别属于SAE和Metric模型

2.3.2.2 删除对象

例：删除对象

```
# 调用Material构造函数创建Material对象aluminum
myMaterial = mdb.models['Model-1'].Material(name = 'aluminum')
# 删除aluminum对象
del mdb.models['Model-1'].materials['aluminum']
# 删除指向aluminum对象的变量myMaterial
del myMaterial
```

Remark: 如果某些Abaqus/CAE对象既不是Mdb对象的成员，也不是Session对象的成员，则del方法不再适用。例如，XYData和Leaf等临时变量，只要指向该对象的变量存在，该对象就存在；如果删除指向该对象的变量，该对象也被删除

2.3.3 指定区域

- 建议采用**findAt**方法查找顶点、边、面或体，findAt方法的参数可以是边、面或体上的任意点，也可以是顶点的XYZ坐标。默认容差为1E-6，将返回指定点或与指定点距离小于1E-6的对象。

2.3.4 指定视口中的显示对象

- 指定视口中的显示对象

session.viewports[name].setValues(displayedObject = object)
displayedObject参数可以是Part对象、Assembly对象、Sketch对象、Odb对象、XY-Plot对象或None对象

2.4 A型部件实例

1.建立模型（模型数据库）

- Mdb() 构造函数创建了模型数据库对象

2.创建二维草图

- mySketch = myModel.ConstrainedSketch(name = 'Sketch A', sheetSize = 200.0)
  调用ConstrainedSketch构造函数创建对象Model A
- 创建内部控制点和外部控制点，依次连线绘图

3.创建三维变形体部件

- 调用BaseSolidExtrude构造函数对草图mySketch增加拉伸特征

4.拉伸草图创建部件的第一个特征

5.创建部件实例

- Instance构造函数创建部件实例

6.布置网格种子，划分单元网格

7.创建新视口

8.在新视口中显示划分单元后的阴影图

P188

2.5 访问输出数据库并对分析结果进行运算

- 读取输出数据库（ODB）文件中的结果数据

例2-3

导入相应模块-创建新视口对象-打开输出数据库-获取不同Step结束时刻的位移增量和应力增量-在新视口中显示deltaDisplacement-在视口中显示deltaStress

Note: Abaqus脚本接口直接与内核进行通信，与Abaqus/CAE的图形用户界面GUI无关

Abaqus对象模型包含了三个根对象：Session对象、Mdb对象、Odb对象

3 EditPlus编辑器的Python开发环境配置

2019年7月27日　　16:07

1 添加Python群组

2 设置Python高亮显示和自动完成功能

- 下载实现语法加亮和自动缩进等功能的特征文件

3 设置Python中文件的模板

4 缩写脚本快速建模

2019年8月19日　　21:29

1 交互式输入

- getInput(…)
- getInputs(…)

3 创建材料库

3.2.1 录制宏

- 可通过录制宏来创建材料库

3.2.1.1 录制宏的操作步骤

1."File"-"Macro Manager"
2."Create"-"Continue": 可以选择abaqusMacro.py文件保存的目录
3."Stop Recording"

3.2.2 通过录制宏来创建材料库

例3-5：通过录制宏创建包含三种材料的材料库
1.在Abaqus/CAE中录制宏文件
2.在宏文件abaqusMacros.py的基础上修改部分源代码，生成创建材料库脚本material_library.py
3.运行脚本material_library.py文件：启动Abaqus/CAE，创建新模型Model-1，选择"Run Script"，将弹出getInput对话框，输入模型名Model-1并单击OK按钮。

5 编写脚本访问输出数据库

2019年8月26日　　20:01

5.1 简介

- 如果希望通过编写脚本来访问和处理输出数据库中的计算结果，需要包含以下语句
  from odbAccess import *
- 若脚本中用到Abaqus脚本接口中的符号常量，则还应导入模块abaqusConstants，包含以下语句
  from abaqusConstants import *
- 如果根据其他的软件分析结果创建Abaqus输出数据库，一般需要创建材料对象material objects、截面对象section objects或梁截面对象beam profile objects，应包含一下语句
  from odbMaterial import *
  from odbSection import *

5.1.1 三组概念

5.1.1.1 模型、模型数据库和输出数据库

1.模型model

- 包含任意多个部件及其相关属性（材料、接触、荷载、边界条件、网格分析步等）。每个模型只能包含一个装配件，同一模型数据库中的不同模型相互独立。

2.模型数据库model database

- 模型数据库由多个模型组成。扩展名为.cae。**建议每个模型数据库只包含一个模型。**

3.输出数据库output database

- 包含Visualization模块后处理需要的所有结果，扩展名为.odb。
- Abaqus/Standard和Abaqus/Explicit求解器的输出数据库中包含3类信息：场输出、历史输出和诊断信息。
- Abaqus/Standard求解器的部分诊断信息也将输出到.msg文件；
- Abaqus/Explicit求解器的部分诊断信息也将输出到.sta文件；
- Abaqus/CFD求解器的输出数据库包含两类信息：**节点场输出nodal field output，单元历史输出element history output**

5.1.2 使用对象模型编写脚本

- 先创建Odb对象，OdbStep对象是Odb对象的一个成员，Frame对象是OdbStep对象的一个成员，FieldOutput对象是Frame对象的一个成员。

例：引用FieldOutput对象中场变量数据序列的某个元素

```
odb.steps['10 hz vibration'].frames[3].fieldOutputs['U'].values[47]
```

5.2 输出数据库对象模型

1.模型数据

2.结果数据

5.2.1 模型数据

- 用于定义分析模型

所有的模型数据如下：

1.部件parts

- 部件是装配体的基本组成部分，在Assembly模块中，同一个部件可以创建多个部件实例。部件不能够直接参与有限元分析。
- 输出数据库中的部件包含所有的节点、单元、表面和集合的信息。

2.根装配root assembly

- 根装配包含定位部件实例的相关信息。有限元模型中需要对根装配定义边界条件等。

3.部件实例 part instances

4.材料materials

5.截面sections

- 截面建立材料属性和部件实例之间的关系。
- 将材料属性赋予截面，再将截面赋予对应的部件实例

6.截面分配section assignments

11.相互作用 interactions

- 定义分析中表面与表面的接触关系，只有接触对contact pairs定义的相互作用才会写出到输出数据库

12.相互作用特性 interaction properties

5.2.2 结果数据

1.分析步 steps

例：访问分析步对象Crush

```
crushStep = odb.steps['Crush']
```

2.帧 frames

- 每个增量步的分析结果

3.场输出 field output

- 适合输出模型中大部分区域的结果数据，且输出频率较低

4.历史输出 history output

5.3 从输出数据库读取数据

5.3.1 打开输出数据库

- from odbAccess import *
  odb = openOdb(path = 'viewer_tutorial.odb')
- 使用openOdb()方法

5.3.2 读取模型数据

1.根装配

- myAssembly = odb.rootAssembly

2.部件实例

- for instanceName in odb.rootAssembly.instances.keys()
  print instanceName
- 输出所有实例的名称

3.区域

- 节点集、单元集、表面

4.材料

- allMaterials = odb.materials
  for materialName in allMaterials.keys():
  print 'Material Name:', materialName

5.截面

6.截面分配

5.3.3 读取结果数据

1.分析步

- for stepName in odb.steps.keys():
  print stepName
  输出每一个分析步的关键字名称
- step1 = odb.steps.values()[0]
  print step1.name
  指定库中的某一项
  输出结果为Step-1

2.帧

- lastFrame = odb.steps['Step-1'].frames[-1]

5.3.4 读取场输出数据

- for fieldName in lastFrame.fieldOutputs.keys():
  print fieldName
  调出第一个分析步最后一帧的所有变量
- 读取场输出变量中各个数据值
  displacement = lastFrame.fieldOutputs['U']
  fieldValues = displacement.values
  # 对于每一个位移值，输出节点编号和节点坐标值
  for v in fieldValues:
  print 'Node=%d U[x]=%6.4f, U[y]=%6.4f' % (v.nodeLabel, v.data[0], v.data[1])

5.3.5 读取历史输出数据

5.5 实例

5.5.1 读取节点信息和单元信息

```
#coding:utf-8
# 导入模块odbAccess后才可以访问输出数据库中的对象
from odbAccess import *
# 打开输出数据库
odb = openOdb(path = 'viewer_tutorial.odb')
# 访问根装配
assembly = odb.rootAssembly
# 对于根装配中的每个部件实例，执行如下命令
numNodes = numElements = 0
for name, instance in assembly.instances.items():
        n = len(instances.nodes)
        print '部件实例的节点数为%s: %d' % (name, n)
        numNodes = numNodes + n
        print '节点坐标'
        # 对部件实例中的每个节点，输出节点编号和节点坐标
        # 对于三维部件，节点坐标分别为X, Y, Z；二维部件，节点坐标为X, Y
        if instance.embeddedSpace = THREE_D:
                print 'X Y Z'
                for node in instance.nodes:
                        print node.coordinates
        else
                print 'X Y'
                for node in instance.nodes:
                        print node.label, node.coordinates
        # 对于每个部件实例中的单元，输出单元编号、单元类型、节点数和单元连接
        n = len(instance.elements)
        print '部件实例的单元数', name, ':', n
        numElements = numElements + n
        print '单元连接'
        print '数目 类型 连接'
        for element in instance.elements:
                print '%5d %8s' % (element.label, element.type),
                for nodeNum in element.connectivity:
                        print '%4d ' % nodeNum,
                print
print '部件实例的个数：', len(assembly.instances)
print '总单元数：', numElements
print '总节点数：', numNodes
odb.close()
```

- 建议将所有的输出数据库文件和实例脚本都存放在工作目录下，否则需要给出文件的绝对路径
- **默认情况下，在Abaqus/CAE中以只读属性打开输出数据库文件，编写脚本访问输出数据库时必须将只读属性取消，否则会出现警告信息**

5.5.2 读取场输出数据 P251.pdf

例：打开输出数据库文件，输出零件的位移信息

```
#coding:utf-8
from odbAccess import *
odb = openOdb(path = 'viewer_tutorial.odb')
# 创建变量表示第一个分析步
step1 = odb.steps['Step-1']
# 创建变量表示第一个分析步的最后一帧
lastFrame = step1.frames[-1]
# 创建变量表示零件施加载荷的节点集'PUNCH'，它属于部件实例PART-1-1
center = odb.rootAssembly.instances['PART-1-1'].nodeSets['PUNCH']
.....
```

5.5.3 创建输出数据库并添加数据

5.5.4 查找Mises应力的最大值

5.5.5 计算位移增量

5.5.6 计算平均应力

6 编写脚本进行后处理

2019年8月27日　　15:02

6.1 自动后处理

- 编写脚本访问输出数据库ODB文件和Visualization模块
- 导入odbAccess模块可以访问输出数据库，导入Visualization模块可以实现后处理

1.可视化命令

例：在视口中显示viewer_tutorial.odb

```
odb = session.openOdb('viewer_tutorial.odb')
vp = session.viewports['Viewport: 1']
vp.setValues(displayedObject = odb)
```

例：绘制Mises应力S的变形云图

```
od = vp.odbDisplay
od.setPrimaryVariable(variable = 'S', outputPosition = INTEGRATION_POINT)
od.display.setValues(plotState = (CONTOURS_ON_DEF,))
```

6.1.2 实例

在变形图模式下，获取输出数据库最后一帧的Mises应力图，并将其保存为PNG格式的文件

6.2 外部数据的后处理

- ANSYS的分析结果属于外部数据

7 基于Abaqus的4D打印变形分析

2019年8月27日　　15:13

1 实现简单实体的受热变形分析
2 实现简单实体的组合受热变形分析

可以直接在创建单个实体时就按照装配体的空间关系进行定位，这样在创建装配体时就不需要移动部件实体了

8 ODB输出数据库

2019年8月30日　　20:20

1 节点位置信息

```
for name, instance in assembly.instances.items():
        for node in instance.nodes:
                print node.label, node.coordinates
```

输出节点坐标的顺序按照instance名称的字母排序
从右上角的点开始，从右向左，从上到下依次输出二维或者三维坐标

Note笔记

2019年7月27日　　9:18

1 如何查看windows系统下的python版本

- 对于我自己的电脑
- cmd进入DOS
- F: 转到F盘
- cd Anaconda3: 转到Anaconda3目录
- python --version: 查看python版本
- 本电脑中的python版本为：python 3.6.5::Anaconda

2 Windows命令行指令

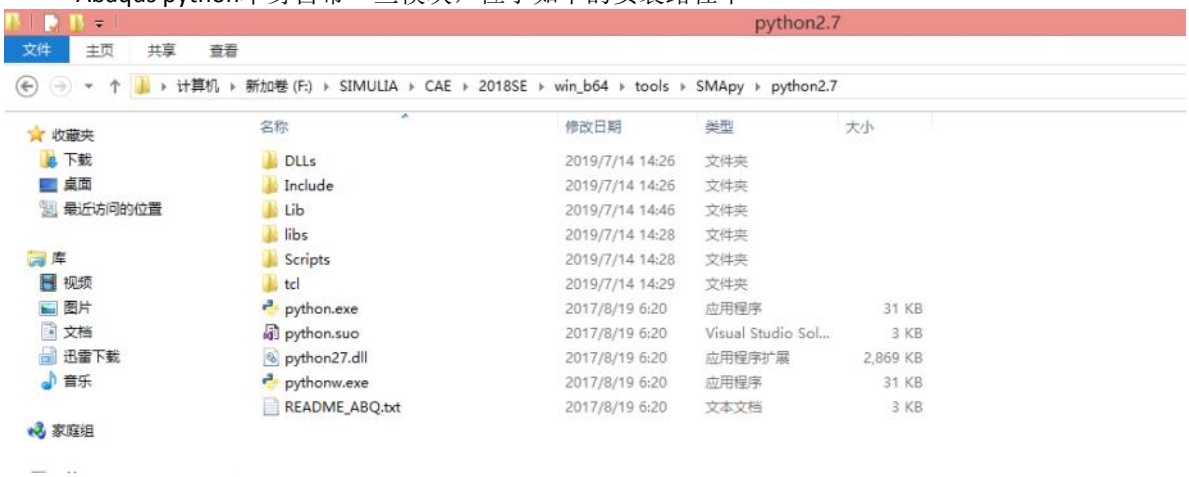
- cd \: 返回根目录
- pip list：查看所有安装的python包的目录
- pip install XXXX: 安装XXXX包

3 Python默认路径修改

- 如果电脑里装了多个版本的python时，要想在cmd python时进入需要的python版本，可以通过修改系统环境变量，将原有路径修改为想要的路径即可
- 右键“计算机”-“属性”-“高级系统设置”-“环境变量”-“用户变量”-“PATH”-将路径修改为带Scripts的路径F:\Anaconda3\envs\abaqus_python2.7\Scripts

4 扩展abaqus python模块

- Abaqus python本身自带一些模块，位于如下的安装路径中



- 可对现有模块进行扩展，安装方法同python setup.py install和pip install
- 首先要解决两个问题

（1）能访问到abaqus python中的python.exe

在系统变量PATH中添加abaqus python的安装路径

F:\SIMULIA\CAE\2018SE\win_b64\tools\SMApy\python2.7

如果系统中安装了其他版本的python，需要对当前的python.exe重命名，例如python_abaqus.exe

（2）安装setuptools和pip模块

下载安装包后解压，cd到该目录下

采用python_abaqus setup.py install的方式安装

也可采用在线安装的方式pip install（需要cd到Scripts路径）

Error: 'pip'不是内部或外部命令，也不是可运行的程序

解决办法：cd到Scripts目录下，再运行pip list等带pip的命令行

Error: 运行pip时提示'Failed to create process'

原因：改动了python的目录名称或位置

5 Abaqus python的版本控制

- 教材和帮助文档中采用的版本均为version 6.10和python 2.6，需要保证版本的一致性

6 Abaqus的object对象类别

- Abaqus的根对象主要有三个：session, mdb和odb

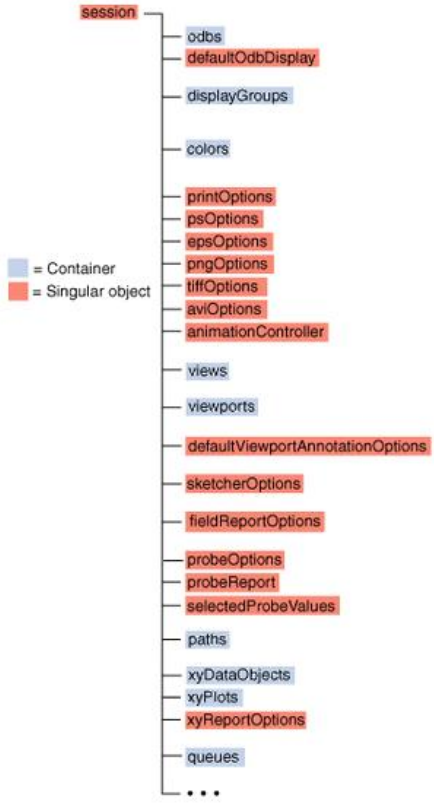
例：

session.viewports['Viewport-1'].bringToFront()
mdb.models['wheel'].rootAssembly.regenerate()
stress=odb.steps['Step-1'].frames[3].fieldOutputs['S']

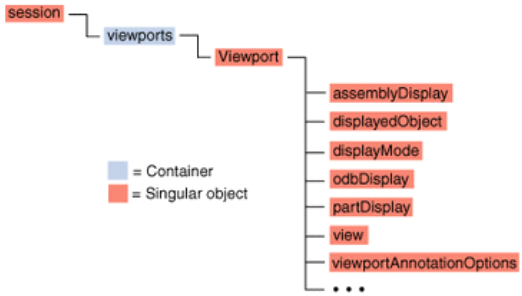
- Abaqus对象模型中的对象分为**容器（由相同类型的对象组成）**和**单个对象（如session和mdb对象）**

1 session对象

- 导入session对象
  from abaqus import *
  from abaqus import session
- 包含定义视口viewport对象，远程队列对象queues，和视图view对象等

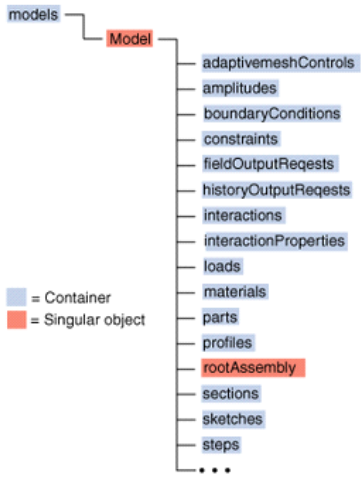


- 其中viewport有如下对象

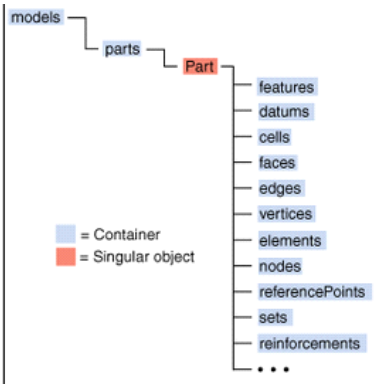


2 mdb对象

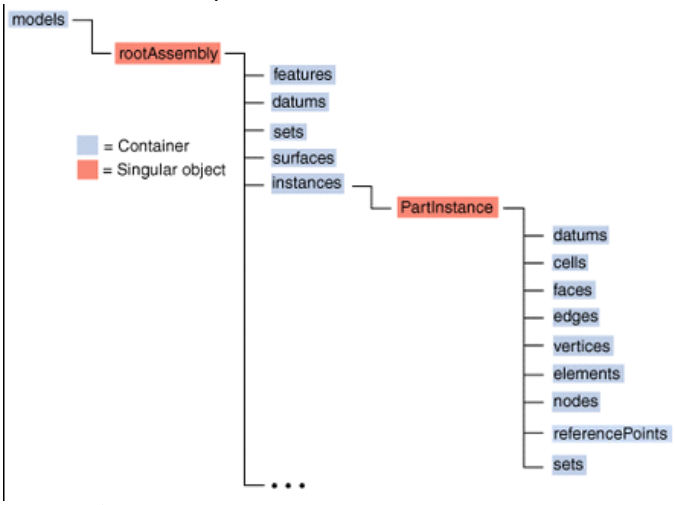
- mdb对象（model database）指保存在模型数据库中的对象，使用下面任意一条语句都可导入mdb对象
  from abaqus import *
  from abaqus import mdb
- mdb对象由model对象和job对象组成，Job对象引用了Model对象，但是Model对象不拥有Job对象
- 其中model对象包括



- Part对象包括

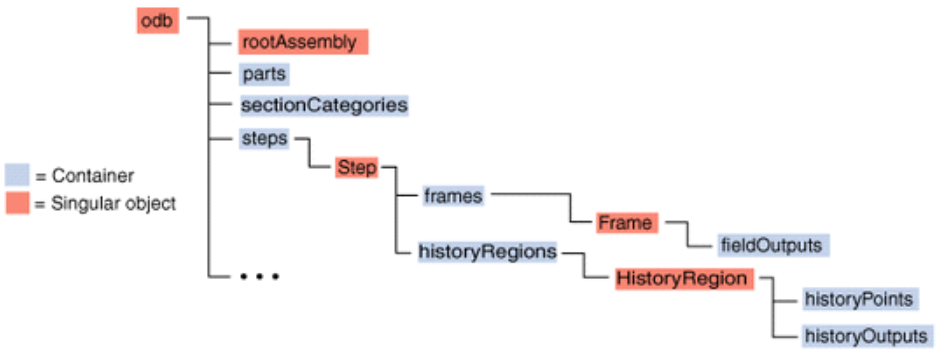


- rootAssembly对象包括



3 odb对象

- 使用下面语句可导入odb对象
  from odbAccess import*
  from odbAccess import openOdb,Odb
- odb对象保存在输出数据库中，由模型数据（model data）和结果数据（result data）组成



7 单个部件和多个部件（装配体）的网格划分有什么区别，接触类型

定义与之的关系是什么

- 分别为部件划分网格，而不是为整个装配体划分网格

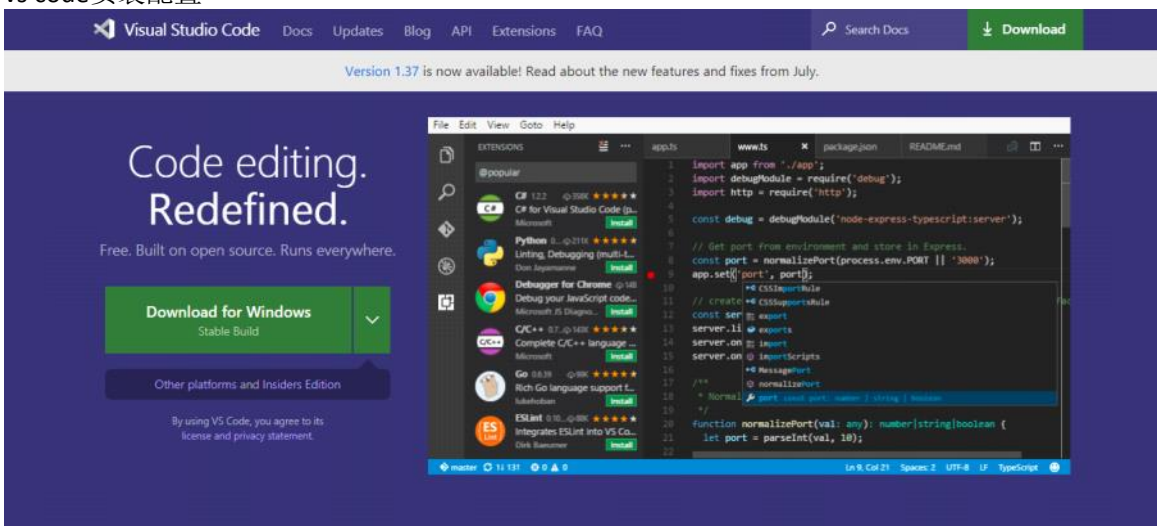
8 部件实例

- 在创建装配体时，针对不同材料属性的部件定义不同的部件实例

vs code+python安装配置

2019年8月9日　　9:24

vs code安装配置



Python补充资料

2019年8月9日　　14:31

1 zip函数

- 将两个iterable对象bind在一起
- for child1, child2 in zip(offspring[::2], offspring[1::2])
  print child1, child2

2 map函数

- 根据函数对指定序列做映射
- def square(x):
  return x**2
  map(square, [1,2,3,4,5])

3 if __name__ == '__main__' 的理解

- 一个Python的文件有两种使用方式
  （1）直接作为脚本执行；
  （2）import到其它python脚本中被调用（模块重用）执行
- if __name__ == '__main__' 表示只有第一种情况下脚本才会被执行

4 pip相关命令

- pip install <包名>：安装
- pip uninstall <包名>：卸载
- pip install <包名> --upgrade：升级包
- pip list: 查看已安装的包

5 用Anaconda管理python环境

- cd Anaconda3
- conda info --env: 查看所有已经安装的python环境
- conda create --name python35 python=3.5: 创建一个python3.5的环境，命名为python35
- conda activate python35: 激活刚刚创建的环境，之后可以调用pip install等命令来管理python包
- condat deactivate: 退出当前所在的python环境
- conda remove -n python35 --all：删除python包

创新点

2019年8月26日　　20:08

1. 从二维扩展至三维
2. 扩展适应性函数fitness function的形式，添加离散密度平滑与应力集中减小的terms
3. Abaqus+Python+MATLAB联合仿真
4. 用粒子群算法代替遗传算法

Ideas

2019年7月29日　　11:35

1 如何评价变形结果与目标形状之间的误差

- 由于输入有限元模型节点的几何离散性，因此可根据目标形状的数学表达形式来拟合后的节点位置。例如目标形状为二次曲线，则用二次曲线来拟合节点位置。

2 设计目标形状的函数

- 调用不同的函数即可得到不同的目标结果

3

DEAP代码测试

2019年9月3日　　19:15

1 规定指定数目的循环

Remark: Use deap.creator, we can create individuals in a simple way.

2 DEAP中map()函数的运用

- map()函数支持输入多个iterable对象，并行地从这些参数中提取元素

DEAP学习

2019年8月9日　　10:16

creator module: create your own type
Initialization: fill created types with random values

1 Basic tutorials

Maximizing fitness with positive weights, minimizing fitness with negative weights

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
```

- "FitenessMin": name for the newly created class
- base.Fitness: base class
- weight attribute must be a tuple, so that multi-objective and single-objective fitness functions can be treated in the same way.

```
creator.create("FitnessMulti", base.Fitness, weights=(-1.0, 1.0))
```

- above command: minimizing the first fitness and maximizing the second fitness

1.1 Individual

- inherit from standard list type and has a fitness attribute

```
import random

from deap import base
from deap import creator
from deap import tools

creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

IND_SIZE=10

toolbox = base.Toolbox()
toolbox.register("attr_float", random.random)
toolbox.register("individual", tools.initRepeat, creator.Individual,
                 toolbox.attr_float, n=IND_SIZE)
```

1.2 Permutation

```
import random

from deap import base
from deap import creator
from deap import tools

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

IND_SIZE=10

toolbox = base.Toolbox()
toolbox.register("indices", random.sample, range(IND_SIZE), IND_SIZE)
toolbox.register("individual", tools.initIterate, creator.Individual,
                 toolbox.indices)
```

1.3 Arithmetic expression

1.4 Evolution strategy

1.5 Particle

2 Operators and algorithms

2.1 A first individual

```
ind1 = toolbox.individual()
```

2.2 Evaluation

- Most personal part of evolutionary algorithm
- A typical evaluation function takes one individual as argument and returns its fitness as a tuple.

```
def evaluate(individual):
    # Do some hard computing on the individual
    a = sum(individual)
    b = len(individual)
    return a, 1. / b

ind1.fitness.values = evaluate(ind1)
print ind1.fitness.valid    # True
print ind1.fitness          # (2.73, 0.2)
```

- **Evaluation function must return a tuple.**

2.3 Mutation

- Various mutation operators are presented in **deap.tools** module
- An independent copy must be made prior to mutation.

In order to apply a mutation (here a gaussian mutation) on the individual `ind1`, simply apply the desired function.

```
mutant = toolbox.clone(ind1)
ind2, = tools.mutGaussian(mutant, mu=0.0, sigma=0.2, indpb=0.2)
del mutant.fitness.values
```

The fitness' values are deleted because they're not related to the individual anymore. As stated above, the mutation does mutate and only mutate an individual it is neither responsible of invalidating the fitness nor anything else. The following shows that `ind2` and `mutant` are in fact the same individual.

```
print ind2 is mutant    # True
print mutant is ind1    # False
```

2.4 Crossover

Lets apply a crossover operation to produce the two children that are cloned beforehand.

```
child1, child2 = [toolbox.clone(ind) for ind in (ind1, ind2)]
tools.cxBlend(child1, child2, 0.5)
del child1.fitness.values
del child2.fitness.values
```

2.5 Selection

- Made among a population.

```
selected = tools.selBest([child1, child2], 2)
print child1 in selected        # True
```

2.6 Using the toolbox

- Contains all revolutionary tools.
- **register()** and **unregister()** are used to add or remove tools from the toolbox

```
from deap import base
from deap import tools

toolbox = base.Toolbox()

def evaluateInd(individual):
    # Do some computation
    return result,

toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evaluateInd)
```

- mate(), mutate(), evaluate(), select()

Initialization

deap.tools. **initRepeat**(*container*, *func*, *n*)

Call the function *container* with a generator function corresponding to the calling *n* times the function *func*.

Parameters:	• **container** – The type to put in the data from func. • **func** – The function that will be called n times to fill the container. • **n** – The number of times to repeat func.
Returns:	An instance of the container filled with data from func.

This helper function can be used in conjunction with a Toolbox to register a generator of filled containers, as individuals or population.

```
>>> import random
>>> random.seed(42)
>>> initRepeat(list, random.random, 2) # doctest: +ELLIPSIS,
...                                    # doctest: +NORMALIZE_WHITESPACE
[0.6394..., 0.0250...]
```

See the List of Floats and Population tutorials for more examples.

deap.tools. **cxTwoPoint**(*ind1*, *ind2*)

Executes a two-point crossover on the input sequence individuals. The two individuals are modified in place and both keep their original length.

Parameters:	• **ind1** – The first individual participating in the crossover. • **ind2** – The second individual participating in the crossover.
Returns:	A tuple of two individuals.

This function uses the `randint()` function from the Python base `random` module.

deap.tools. **mutFlipBit**(*individual*, *indpb*)

Flip the value of the attributes of the input individual and return the mutant. The *individual* is expected to be a sequence and the values of the attributes shall stay valid after the `not` operator is called on them. The *indpb* argument is the probability of each attribute to be flipped. This mutation is usually applied on boolean individuals.

Parameters:	• **individual** – Individual to be mutated. • **indpb** – Independent probability for each attribute to be flipped.
Returns:	A tuple of one individual.

This function uses the `random()` function from the python base `random` module.

deap.tools. **selTournament**(*individuals*, *k*, *tournsize*, *fit_attr='fitness'*)

Select the best individual among *tournsize* randomly chosen individuals, *k* times. The list returned contains references to the input *individuals*.

Parameters:	• **individuals** – A list of individuals to select from. • **k** – The number of individuals to select. • **tournsize** – The number of individuals participating in each tournament. • **fit_attr** – The attribute of individuals to use as selection criterion
Returns:	A list of selected individuals.

This function uses the `choice()` function from the python base `random` module.

- toolbox.register("select", tools.selTournament, tournsize=3)：此句规定了生成offspring的方式。在上一generation中，经历了mutation, crossover操作产生了新的generation，tournsize=3表示从新的generation中每次随机选取3个indivduals，选择fitnessmax的那一个进入下一generation，“随机选择3个操作”的次数由offspring = toolbox.select(pop, len(pop))规定。