## 前言

本人看网上关于MOOSE的中文教程极少，并且英文教程对于没有接触过有限元的新手极其不友好，于是想做这么一款入门教程。

但是本人深知自己学识浅薄，接触MOOSE时间也并不久，很多底层架构了解的其实不多。有错误或疑问欢迎在https://github.com/MissCat-2025/fuel\_rods.git给我留言。

一些标记的规则申明：

1. 《wsl –update》，《cd ~》，代码复制黏贴可直接运行。书名号加黄色背景的部分，直接复制到指定位置运行（默认为Cursor编辑器对应的linux终端，其他位置会特殊说明）。
2. [下载发行版本](https://aka.ms/wslubuntu2004)，下划线有超链接。
3. （xxxxx黄色背景+红色字是相当重要，）（在集成终端中打开，强调要注意。）
4. （xxx，灰色文字不那么重要，比较赶时间可跳过。）

目录

[前言 1](#_Toc193808357)

[第1章 安装与更新MOOSE相关 3](#_Toc193808358)

[1.1 安装MOOSE 3](#_Toc193808359)

[1.1.1 第一步，安装好linux子系统。 3](#_Toc193808360)

[1.1.2 第二步，在Cursor下打开linux子系统。 3](#_Toc193808361)

[1.1.3 第三步，配置MOOSE运行环境-python环境安装 5](#_Toc193808362)

[1.1.4 第四步，安装MOOSE软件 6](#_Toc193808363)

[1.1.5 第五步，验证MOOSE软件 6](#_Toc193808364)

[1.2 更新Conda和MOOSE 6](#_Toc193808365)

[1.3 Git更新 7](#_Toc193808366)

[第2章 创建好自己的app 7](#_Toc193808367)

[2.1 前言 7](#_Toc193808368)

[2.2 整体把握 8](#_Toc193808369)

[2.3 战前的准备：创建app+ Makefile模块配置+include/src代码开发 8](#_Toc193808370)

[2.4 拟定作战计划-输入文件的配置 10](#_Toc193808371)

[2.5 MOOSE整体逻辑 10](#_Toc193808372)

[第3章 简单介绍问题 12](#_Toc193808373)

[3.1 几何与网格[Mesh] 13](#_Toc193808374)

[3.1.1 常用函数 13](#_Toc193808375)

[3.2 [Kernels]： 13](#_Toc193808376)

[3.2.1 一般参数的定义与引用 14](#_Toc193808377)

[3.2.2 其他常用3.1 [Kernels] 14](#_Toc193808378)

[3.3 [Materials]： 14](#_Toc193808379)

[3.3.1 常用Materials总结 15](#_Toc193808380)

[第1章 Cursor的经验之谈 15](#_Toc193808381)

[第2章 Include与scr自定义函数 15](#_Toc193808382)

[第3章 附录 16](#_Toc193808383)

[3.1 Linux基础命令 16](#_Toc193808384)

[3.2 多线程计算 17](#_Toc193808385)

[3.2.1 更加专业的确定方法： 18](#_Toc193808386)

[3.3 Restart and Recover 19](#_Toc193808387)

[3.3.1 具体设置 19](#_Toc193808388)

[3.4 GPU并行 20](#_Toc193808389)

[3.5 常见报错 20](#_Toc193808390)

[3.5.1 代码格式问题 20](#_Toc193808391)

# 安装与更新MOOSE相关

**第一章安装moose可参考**[**fuel\_rods**](https://github.com/MissCat-2025/fuel_rods.git)**/README.md，那个版本更加简洁，且有配套脚本可以傻瓜式安装，如果你想看更加详细的版本，看这个安装教程也可以。**

如果你是Windows系统，则需要安装linux子系统（WSL），然后安装conda运行环境，最后安装MOOSE。写本教程的时间是2025-3-14，由于版本可能更新，本教程中涉及的代码可能过时而无效或报错。强烈建议去官网按他的流程来一遍（[Windows Subsystem for Linux](https://mooseframework.inl.gov/getting_started/installation/windows.html)）（[Installing MOOSE | MOOSE](https://mooseframework.inl.gov/getting_started/installation/index.html)）。

## 安装MOOSE

### 第一步，安装好linux子系统。

（这一步是按照[windows11 安装WSL2全流程\_wsl2安装-CSDN博客](https://blog.csdn.net/u011119817/article/details/130745551)来的，试过很多回，目前测试是安装最快和最稳的），按照这个来，安装好linux子系统就OK，可以不用安装图形界面。具体步骤如下

* 1. [启用window子系统及虚拟化](https://blog.csdn.net/u011119817/article/details/130745551#1window_14)

power shell 以管理员方式运行，输入这两行代码后重启电脑

《Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux》

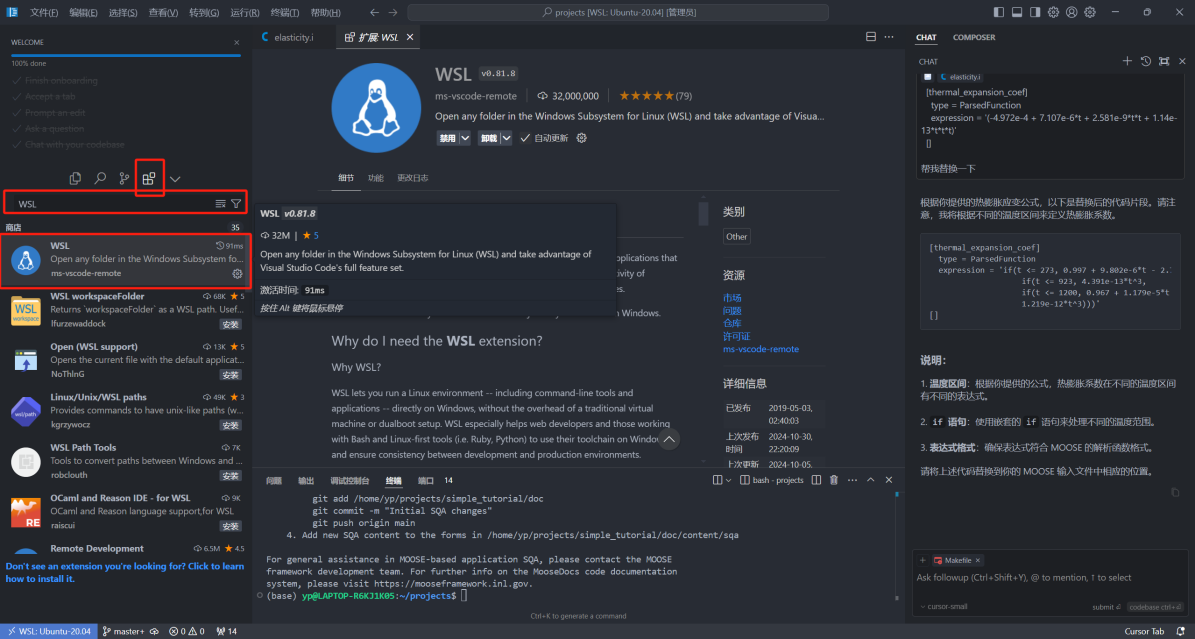
《Enable-WindowsOptionalFeature -Online -FeatureName VirtualMachinePlatform》

* 1. [下载WSL发行版本](https://aka.ms/wslubuntu2004)
  2. power shell 以管理员方式运行后，输入：《wsl --update》
  3. 双击安装b)中下好的WSL，设置linux系统名字（后续提及的“你的linux名字”就是指的是这个）与密码**（注意：输入密码时并不会显示你输入了什么，密码注意别太麻烦，设置一个字符都可以，不然就是在给自己添乱）**
  4. linux子系统(WSL安装完成)

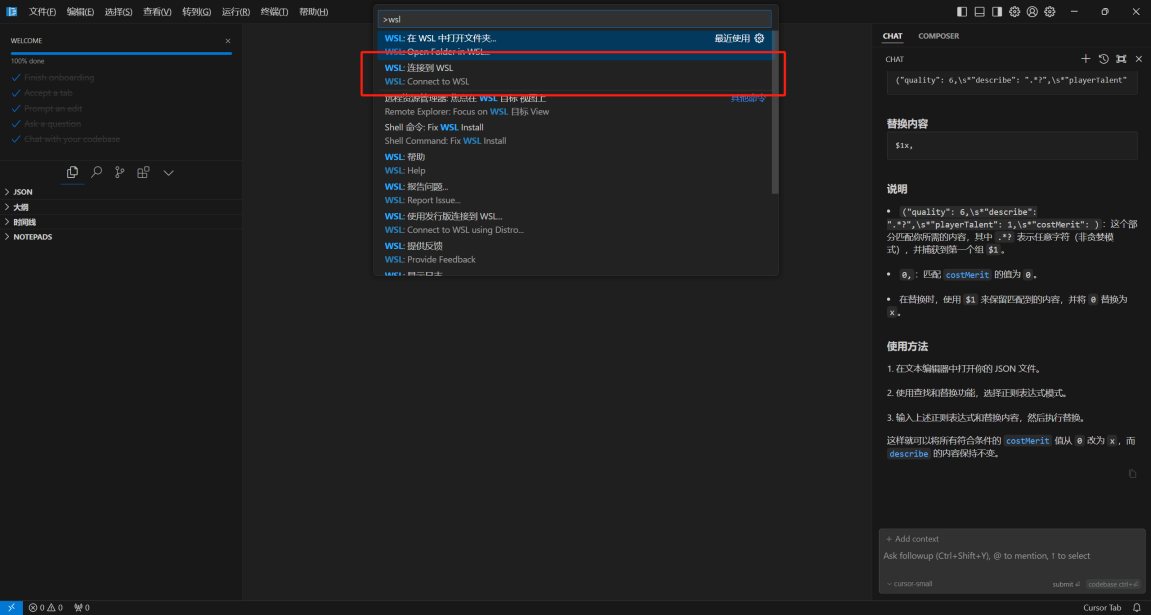
### 第二步，在[Cursor](https://www.cursor.com/)下打开linux子系统。

推荐下载[Cursor](https://www.cursor.com/)或者[VsCode](https://code.visualstudio.com/)软件，使用Cursor可直接打开linux子系统的文件夹，方便太多。（这两个软件都集成了ai的代码编辑器功能，极其推荐，并且他们的界面基本一样。下面就介绍Cursor。此外第四章有详细的Cursor使用经验之谈（两个操作即可实现无限使用Cursor），掌握该软件可大幅提升写代码水平）

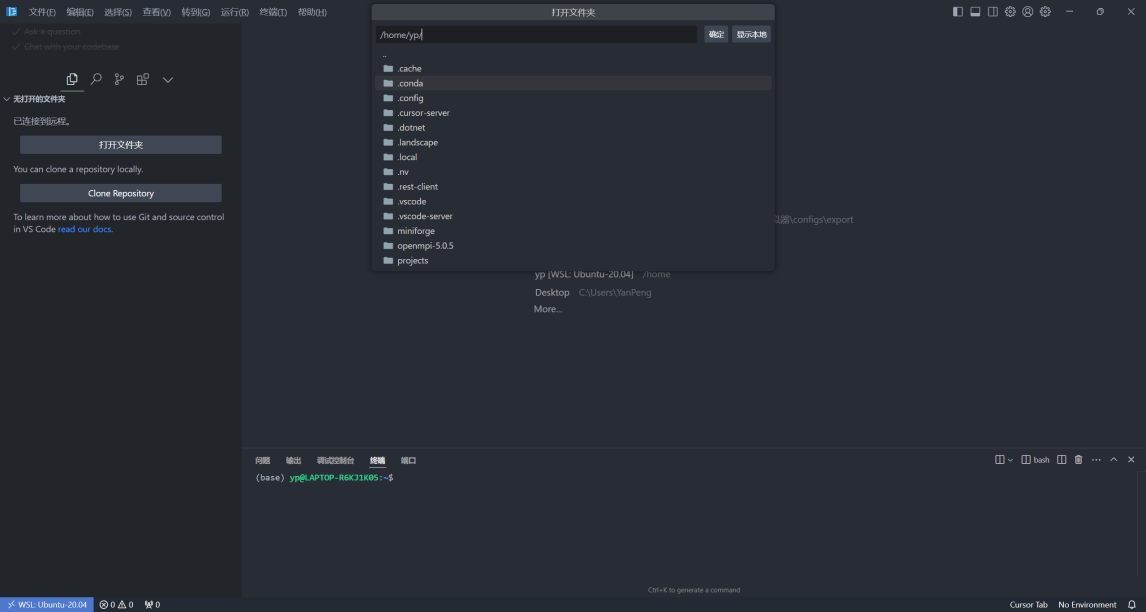
* 1. 下载[Cursor](https://www.cursor.com/)
  2. 登入
  3. 在软件里面安装WSL插件



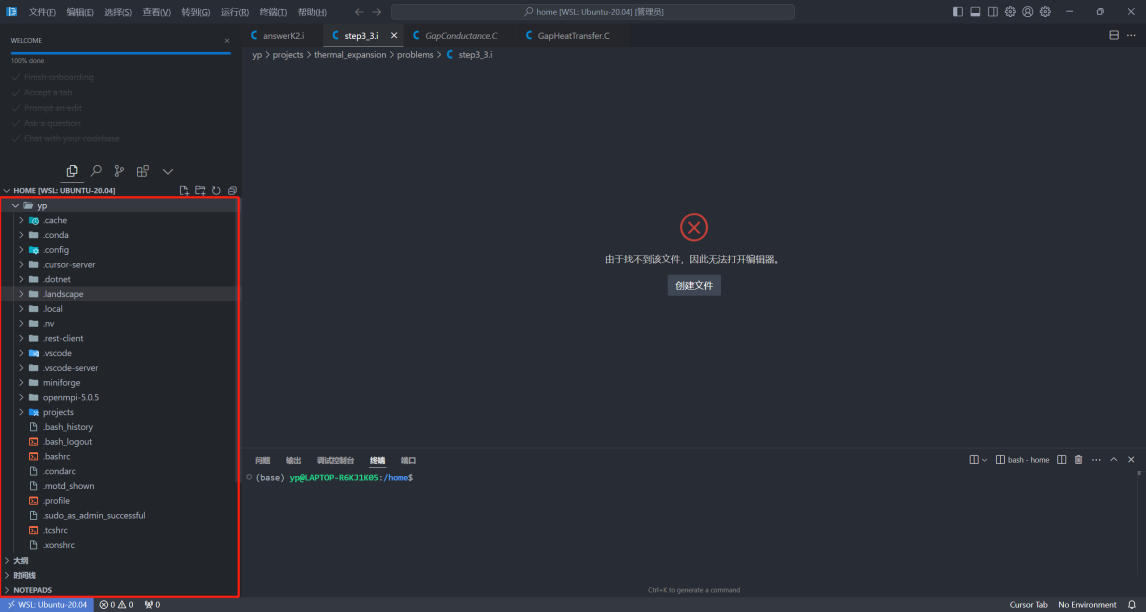
* 1. 输入>WSL，选择连接到WSL



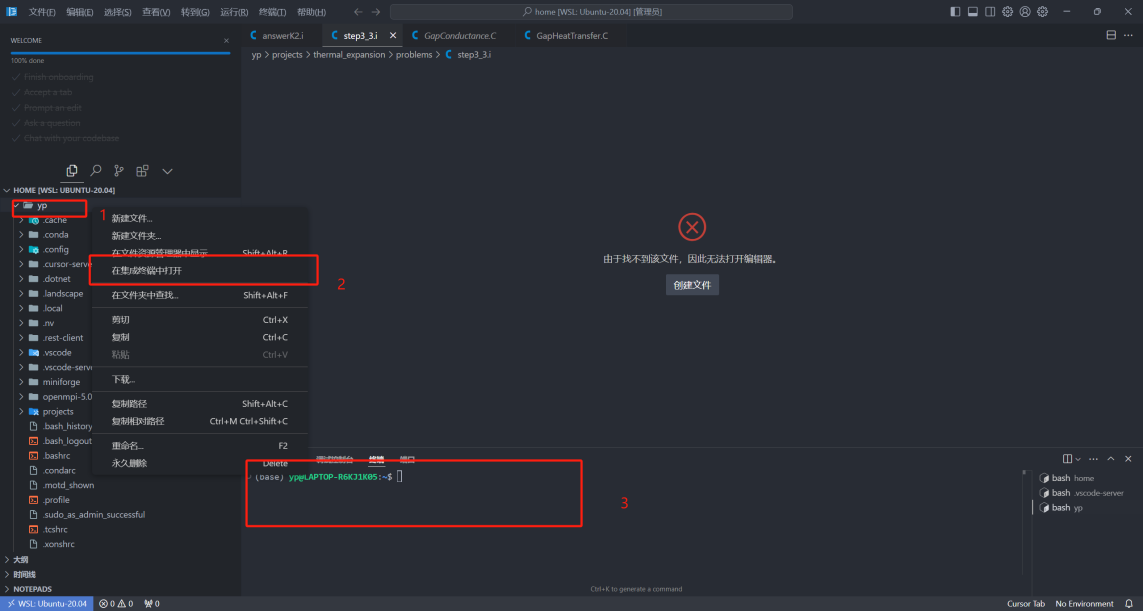
* 1. 然后弹出窗口后，选择打开文件夹/home，点确定



* 1. 出现红色框框。说明你已经配置好了



* 1. 选择《你的linux名字》文件夹，选《在集成终端中打开》，出现3，就说明linux环境配置完成



PS：右键文件选择《在集成终端中打开》这一招用来切换终端位置非常好用！

### 第三步，配置MOOSE运行环境-python环境安装

（[Conda MOOSE Environment | MOOSE](https://mooseframework.inl.gov/getting_started/installation/conda.html)）

* 1. 在linux终端处（上图《3》的位置处）中分别输入

《curl -L -O <https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-x86_64.sh>》以下载MOOSE环境软件Miniforge3

《bash Miniforge3-Linux-x86\_64.sh -b -p ~/miniforge》以解压MOOSE环境软件miniforge

* 1. 输入《export PATH=$HOME/miniforge/bin:$PATH》以导出环境软件miniforge的路径，目的以后一打开linux终端就能自动启用PATH环境的环境，方便操作
  2. 输入《conda init --all》，你就会发现终端变成了《(base) 你的linux名字@XXXX:~$》
  3. 更新python环境《conda update --all --yes》
  4. 输入《conda config --add channels <https://conda.software.inl.gov/public>》以便下载MOOSE运行环境
  5. 输入《conda create -n moose moose-dev=2025.xx.xx=mpich》安装运行环境，注意红字2025.xx.xx代表的MOOSE软件版本，这可能更新了，[请按住Ctrl键点我去官网看一下](https://mooseframework.inl.gov/getting_started/installation/conda.html)最新版本，大概在下图的位置（漫长等待）。



代码解释：“conda create -n moose”中的moose表示这个环境名字叫做moose，你也可以把它改为其他名字。

* 1. 输入《conda activate moose》以激活环境moose（注意现在这个moose只是MOOSE软件的运行环境，现在还没有安装MOOSE软件，每次你要打开MOOSE软件，先输入conda activate moose激活MOOSE的运行环境环境以避免错误）（如果你在f）步骤中将moose命名为xxx，则这里代码就变成了conda activate xxx）

### 第四步，安装MOOSE软件

* 1. 输入《mkdir -p ~/projects》。在/home/yp路径下创建projects文件夹（自己直接在cursor里右键创建也完全可以）。注意最终是/home/你的linux系统名字/projects
  2. 输入《cd ~/projects》。进入刚才创建的projects文件夹
  3. 输入《git clone https://github.com/idaholab/moose.git》这一步便是抓取MOOSE软件，**需要漫长的等待,如果一直下载不下来，请看1.3 Git更新**
  4. 输入《cd moose》进入moose软件
  5. 输入《git checkout master》将当前工作目录切换到moose的主分支，（输入就好了，可无视这个解释）（通常称为 master 分支）

### 第五步，验证MOOSE软件

* 1. 输入《cd ~/projects/moose/test》，进入MOOSE软件的test目录
  2. 输入《make -j 6》，编译，漫长的等待，注意，这个《make -j 6》这个蓝色6是根据你的cpu内核数来的，合适的配置可以加快编译与程序运行，具体请看3.2多线程计算。编译test文件夹下面的make文件以导出我们最为需要的-opt文件，这个以-opt结尾的文件（例如全称为moose\_test-opt）是我们运行MOOSE输入文件(xxx.i)的必要文件。(可以这样理解，make编译的是我们的整个模型，但模型需要编译浓缩成一个程序才能使用，编译结果是-opt文件，我们用这个模型(-opt)与输入文件(.i)去运行相应MOOSE程序。之后会反复验证与强调这句话)。
  3. 输入《cd ~/projects/moose/test》，进入MOOSE软件的test目录
  4. 输入《./run\_tests -j 6》运行写好的测试函数，以验证MOOSE的各个物理模块是否安装成功。漫长的等待

## 更新Conda和MOOSE

且如果更新conda环境，请始终更新配套的moose版本：（版本不匹配编译的时候会有明显提示）

直接删除moose环境再下一遍，以保证100%更新后无bug，依次运行如下代码即可

《conda activate base》

《conda env remove -n moose》

《conda create -n moose moose-dev=2025.xx.xx=mpich》，[请按住Ctrl键点我去官网看一下](https://mooseframework.inl.gov/getting_started/installation/conda.html)xx.xx为多少，大概在如下位置：



《conda activate moose》

要更新MOOSE存储库，请使用以下命令：

《cd ~/projects/moose》

《git fetch origin》

《git rebase origin/master》

# 《git rebase origin/master》 出问题的时候，可以先运行《git status》看看什么问题，应该是改动过moose某些东西，如果这些改动都不需要，直接丢弃运行如下代码即可：

《git reset --hard HEAD》

## Git更新

Git一直出错：

运行git clone https://github.com/xxx.git的时候一直下不下来，可以将下面三条代码全部运行：

《sudo apt-get update》

《sudo apt-get install --reinstall ca-certificates》

《sudo apt-get upgrade git》

# 创建好自己的app

## 前言

这一小节的目的是检测实现moose的基础条件是否满足。

在进行这一章之前，如果你还无法回答接下来的问题，请先学习相关理论，再来运行你的模型，这将提高你的效率。

1. 你要解的偏微分方程是？它对应的弱形式？对应有几个变量？

例如，我想实现的偏微分方程是上面的热力耦合问题，那必然有如下最少两个偏微分方程，变量分别是热传导对应的温度T，力平衡方程对应的位移u。

热传导方程：

力平衡方程

由于MOOSE的控制方程对应的是弱形式，因此还需列出相应的弱形式，具体请看[官网的推导](https://mooseframework.inl.gov/getting_started/examples_and_tutorials/tutorial01_app_development/step04_weak_form.html)

**强形式变弱形式简单总结就是将所有项移到一边，并根据规则改变正负号，特别注意**

**散度（）需要变号，其余的项一般不需要变号。**

例如变成。因此热传导方程变成=0

后将所有项其移到一边就是MOOSE需要的控制方程对应的是弱形式了。第一步要做的就是如此

1. 偏微分方程的边界条件、初始条件是？（查论文资料）
2. 你要运行的模型的几何与网格划分是？（查论文资料）
3. 偏微分方程中涉及的材料参数或模型的其他参数是？（如热导率、瞬态与准静态的步长与运行总时间等、固体力学中总应变中是否有本征应变、塑性应变、蠕变等）
4. 你想要的输出结果有哪些？（变量对应的分布图？某个过程量？）

以上任何不懂的地方，都可以通过与cursor中的claude3.7-sonnet-thinking、deepseek-r1等推理模型进行快速学习。

## 整体把握

这一节的目的是对moose的整体有个把握，知道【调用MOOSE官方模块库：通过Makefile】、【**功能扩展：**通过自定义的模型文件include与src】、【输入文件与运行程序】的关系

若您曾使用过Abaqus、Fluent、COMSOL等商业有限元软件，会发现MOOSE的核心逻辑与这些软件具有相似性，但其采用纯代码驱动的开发模式，不依赖图形界面。MOOSE通过创建专属应用程序（App）来实现仿真流程构建，这一过程类似于商业软件中在特定物理模块（如固体力学、热流体等）创建工作平台（Workbench）的操作。具体而言：

1. **模块化架构**：商业软件通过选择物理模块激活相关求解器与材料模型，MOOSE则通过在App的Makefile中链接官方模块库（如固体力学模块SOLID\_MECHANICS、传热模块HEAT\_TRANSFER等）来继承对应功能。
2. **功能扩展**：二者都支持通过自定义本构模型、边界条件等扩展功能，但MOOSE采用C++面向对象编程实现，而非图形化参数配置。
3. **工作流构建**：商业软件通过界面操作组合预定义模块，MOOSE则通过编写输入文件定义网格、材料属性、求解器设置等计算要素。

如果没有接触过相关的软件，那拿简化的军事概念做个简单的例子：

**表格 2‑1 MOOSE的理解**

|  |  |  |
| --- | --- | --- |
| 军事概念 | MOOSE对应组件 | 技术解析 |
| **新兵入伍** | 创建应用程序(APP) | 通过./moose/scripts/stork.sh XXX命令创建专属仿真环境，建立代码仓库 |
| **部队编制** | Makefile模块配置 | 选择官方模块如HEAT\_TRANSFER (传热分析)等官方模块 |
| **特种训练** | include/src代码开发 | 继承MOOSE基类开发自定义材料、边界条件等，make编译生成APP对应的可执行文件(\*.opt) |
| **作战计划** | 输入文件(.i) | 配置各个部分，网格/变量/内核/材料 |
| **战役实施** | 执行计算 | 通过./app\_name-opt -i input.i启动仿真，生成exodus等结果文件 |

为了加深理解，后续采用表格 2‑1 MOOSE的理解进行进一步说明。

1. 前三步：创建应用程序(APP)、Makefile模块配置、include/src代码开发都是在准备模型，约等于作战前的准备；
2. 输入文件的配置就是拟定作战计划；
3. 最终使用./app\_name-opt -i input.i来进行计算生成结果，一次小战役就完成了。

## 战前的准备：创建app+ Makefile模块配置+include/src代码开发

这一小节的目的是创建自己的app，该app调用了官方的模块，并生成可执行的-opt文件。

首先我们先进行新兵入伍操作-基于MOOSE创建app，具体操作是运行如下代码：

1. 《cd ~/projects》进入MOOSE的上一级目录（假设文件结构为~/projects/moose）
2. 《./moose/scripts/stork.sh XXX》创建app。注意对应的app文件夹将自动变小写， APP名字就叫做XXXApp。这里请分清楚文件夹的名字与APP名字的区别，app名字可以通过xxx/src/base文件夹确认（图表 2‑1）

|  |  |
| --- | --- |
|  |  |
| 图表 2‑1 xxx/src/base文件夹确认app名字 | 图表 2‑2 xxx/Makefile文件选择官方自带的模块 |

1. xxx/Makefile选官方自带的模块，将对应的no改为yes即可，**并快捷键Ctrl+S保存（改文件请养成保存的好习惯）**。接下来我们看到xxx文件夹下面的Makefile文件，点击并滑动页面至图表 2‑2，可以看到里面有丰富的模块可供选择，具体可以[参考官网了解更多](https://mooseframework.inl.gov/modules/index.html)。对于xxx为了实现芯块的热力耦合研究，选取热传导HEAT\_TRANSFER、固体力学SOLID\_MECHANICS两个物理模块。将no改为yes即可。
2. 特种训练：自定义的include/src代码开发。但由于我们才开始接触MOOSE，一开始就开发自己的代码基本就是给自己徒增困难（MOOSE里面有相当多的函数可用，用官方自带的模块就能解决大多数问题），后续将有详细的教程来自定义include/src代码开发。
3. 在Makefile对应的路径下进行编译。
   1. 按图表 2‑3顺序操作：右键Makefile文件，选择《在集成终端中打开》，将出现一个终端
   2. 在终端输入《conda activate moose》。终端位置对了后（终端应该显示类似(base)yy@LAPTOP-R6KJ1K05:~/projects/xxx$），我们就输入《conda activate moose》激活MOOSE运行环境。
   3. 在终端输入《make -j6》编译xxx-opt文件。最终检测手段就是在xxx文件夹下有没有生成xxx-opt文件（蓝色的6与配置有关，详细见3.2）。漫长等待，其中代码运行的时候应出现：图表 2‑4显示的图像。这个xxx-opt文件就是我们运行模型要用到的app

|  |  |
| --- | --- |
|  |  |
| 图表 2‑3 在集成终端打开将出现的图片 | 图表 2‑4 编译成功显示的图片 |

## 拟定作战计划-输入文件.i的配置

这一小节的目的是详细介绍输入文件的基础部分，并且运行第一个棒状芯块热力耦合程序。

打开路径下的文件：xxx/test/tests/kernels/simple\_diffusion/simple\_diffusion.i，你将看到如下文件，如果你的输入文件的显示格式你看着不舒服，可以试试下载Atom One Dark Theme或类似的UI主题。

|  |  |
| --- | --- |
|  |  |
| 图表 2‑5 典型的输入文件 | 图表 2‑6 UI主题插件 |

可以看到，输入文件分为

[Mesh]：定义计算域的空间离散化策略

[Variables]：定义Sobolev空间中的试探函数空间，大白话即变量

[Kernels]：控制方程的每一个项

[BCs]：边界条件

[Materials]：（材料属性，这个基础文件中没有，但是这个稍微复杂一点的问题都有）

[Executioner]：求解器设置

[Outputs]：数据后处理

等部分，接下来将一个个详细介绍相应的功能

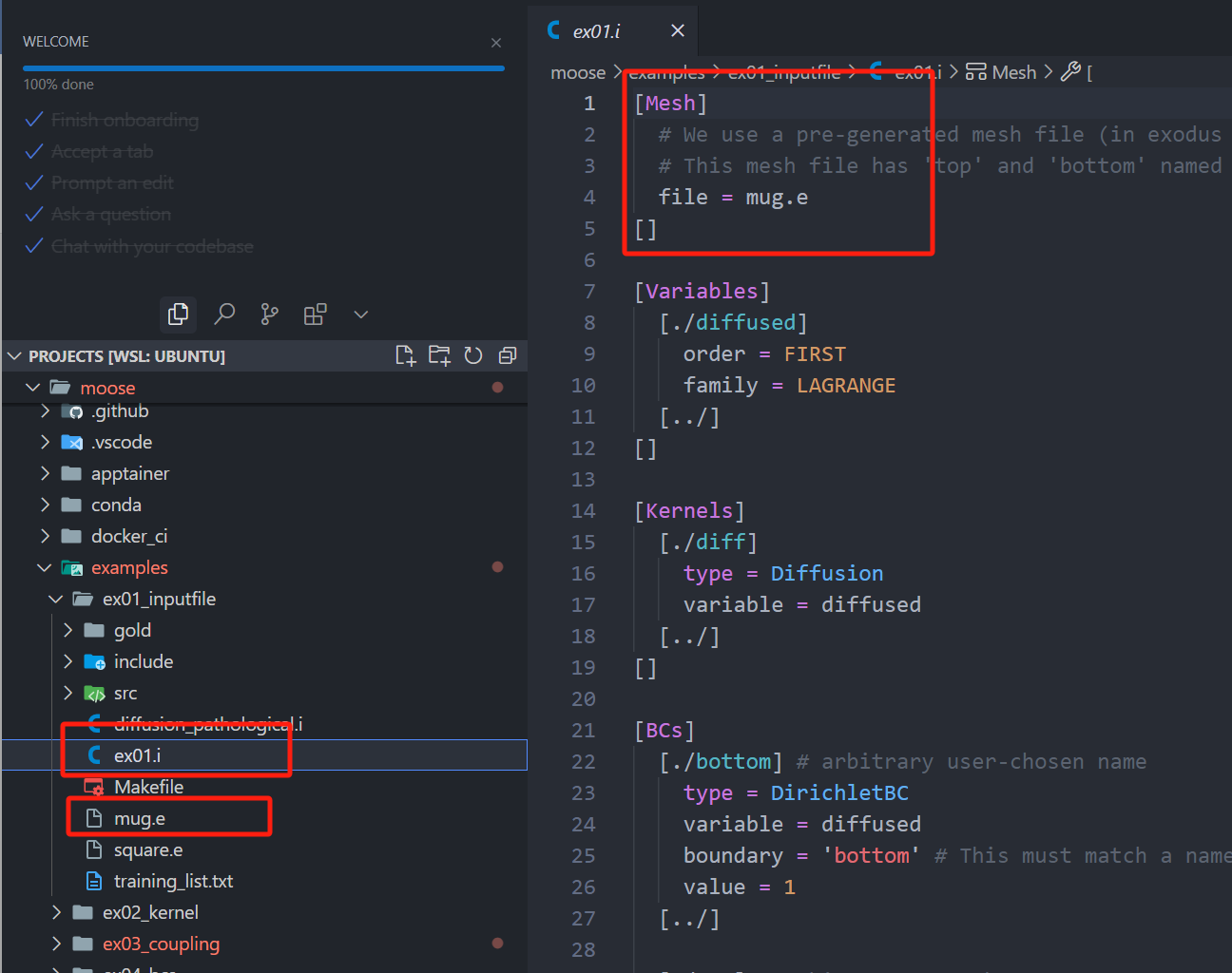
### [Mesh]

MOOSE提供两种主要的网格生成方式：

* **离线生成**：

通过外部工具如CUBIT（由Sandia国家实验室开发，现由Coreform授权，得下破解版）创建复杂几何体。（目前尝试过fluent的几何与网格是不能直接被MOOSE使用的），MOOSE能支持的几何网格格式[请看官网](https://mooseframework.inl.gov/source/mesh/FileMesh.html)。

如果是离线生成，需要在第三方软件中将几何、网格、相关的命名（Blocks、SideSets、NodeSets）等全部完成后输出成相应格式，最后在相应的位置即可，例如MOOSE中的例子moose/examples/ex01\_inputfile/ex01.i，它的几何文件为mug.e



图表 2‑7 离线生成网格与moose的对接

* [**在线生成**](https://mooseframework.inl.gov/syntax/Mesh/)：

调用MOOSE写好的网格生成函数生成几何与网格、命名等。[具体请看官网](https://mooseframework.inl.gov/syntax/Mesh/)

|  |  |
| --- | --- |
|  |  |
| 图表 2‑8 棒状芯块 | 图表 2‑9 环形芯块 |

例如在XXXApp中,生成一个棒状芯块，即可调用ConcentricCircleMeshGenerator函数，如图表 2‑8具体请看fuel\_rods/MOOSE\_Tutorials/[Mesh]/RodFuel/NoClad3D.i，里面还有带包壳的，1/4模型等。

生成一个环形芯块，即可调用ConcentricCircleMeshGenerator函数，如图表 2‑9具体请看fuel\_rods/MOOSE\_Tutorials/[Mesh]/Annular/NoClad3D.i里面还有带包壳的，1/4模型等。

1. MOOSE常用网格生成器

MOOSE提供多种网格生成器，包括：

* GeneratedMeshGenerator：创建简单的规则网格（如矩形、立方体）
* FileMeshGenerator：从文件读取网格（支持多种格式）
* AnnularMeshGenerator：创建环形或扇形网格
* CartesianMeshGenerator：创建非均匀笛卡尔网格
* DistributedRectilinearMeshGenerator：创建可进行并行计算的规则网格
* ConcentricCircleMeshGenerator：创建同心圆网格
* ExamplePatchMeshGenerator：创建2D或3D块状网格

1. **网格操作工具**

MOOSE还提供了多种工具用于修改和处理现有网格：

* BlockDeletionGenerator：删除指定子域的单元
* BoundaryDeletionGenerator：删除边界集
* BreakMeshByBlockGenerator：在不同块的接口处断开网格
* CoarsenBlockGenerator：粗化指定块中的网格
* CombinerGenerator：合并多个网格为一个
* ElementOrderConversionGenerator：转换单元阶数
* ExtruderGenerator：将1D网格挤压为2D，或将2D网格挤压为3D
* CutMeshByPlaneGenerator：沿指定平面切割网格

1. **网格坐标系统**

MOOSE支持多种坐标系统：

* XYZ：标准笛卡尔坐标系（默认）
* RZ：轴对称圆柱坐标系
* RSPHERICAL：球坐标系

可以通过coord\_type参数指定，并可为不同的子域设置不同的坐标系统。

1. **混合维度网格**

MOOSE支持在同一模拟中混合使用不同维度的单元（如同时使用1D、2D和3D单元）。这对于模拟包含细长结构（如梁、壳）的复杂系统非常有用。（详细请看官网）

1. **网格复制与分布式计算**

MOOSE支持两种并行计算模式：

* Replicated Mesh：在每个处理器上复制整个网格
* Distributed Mesh：将网格分割到不同处理器上

通过网格分割可以优化大规模并行计算的性能。

1. **位移网格**

MOOSE支持位移网格功能，允许网格随着解的变化而变形。这对模拟结构变形等问题非常重要。

1. **网格元数据和附加信息**

* 命名实体支持：为块、边界等添加有意义的名称
* 唯一ID：为节点和单元分配唯一标识符
* 周期性节点映射：处理周期性边界条件
* 额外整数ID：为单元存储附加信息
* 网格元数据：存储与网格相关的额外信息

1. **网格系统进阶功能**

* 自适应网格细化：基于解的梯度或误差估计动态调整网格密度
* 高阶几何表示：支持高阶单元以更准确地表示曲面几何
* 网格平滑：改善网格质量的技术
* 并行网格操作：在大规模并行环境中高效地处理网格

### [Variables]

### [Kernels]

### [Materials]

### [BCs]

### [Executioner]

### [Outputs]

## MOOSE整体逻辑

如果以前接触过其他有限元软件，那MOOSE逻辑与它们都差不多。如果你是第一次接触有限元软件，那么MOOSE整体逻辑举个例子更好理解：**完整的运行一个MOOSE程序可以看作是以自己的视角参军打仗，包括战争开始前的各种准备（-opt模型文件），以及真正发生战争时的部队、策略、自己的状态等（.i输入文件）。**

-opt模型文件-战争前的准备。你需要加入某个部队（启用MOOSE的models），用部队提供的一些武器与资源（使用models中的各种函数），以及锻炼一些战斗技能（自己自定义函数）。

.i输入文件-真正的战争的各种状态。真正发生战争时，你得把自己的任务目标明确好（网格文件.e，控制方程对应的变量Variables，边界条件BCs，控制方程变量对应的Kernel,等），得明确自己、队伍与敌人的各种信息（控制方程中的对应的Materials，求解器Executioner，输出控制Outputs，甚至辅助获取信息的AuxVariables、AuxKernels等）。

最终完整的战斗的最简单的代码（完整的运行跑一个MOOSE程序）就是：

X1-opt -i Y1.i ,

X1-opt就是-opt模型文件（战争前的准备），Y1.i就是输入文件（真正的战争的各种状态）。

创建的APP就是你自己，它等于你的全部；Makefile中启用MOOSE的一些大模块（MODULES）约等于自己加入了某个部队，加入的越多，能用的武器与资源越多，但是编译起来也慢。

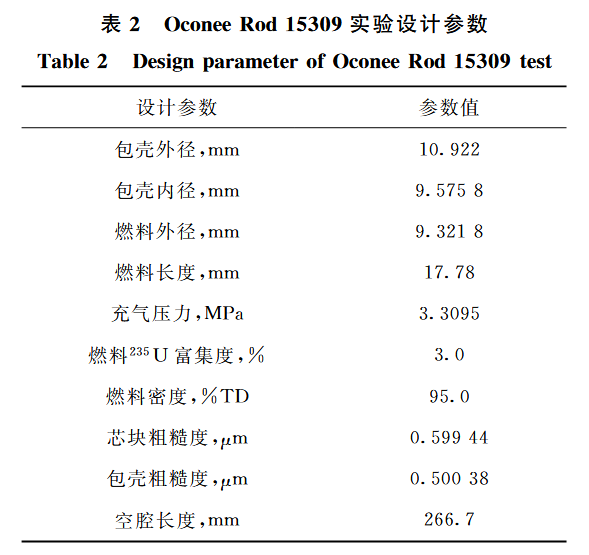
总结一下大概流程就是：

1. 创建基于MOOSE的APP，这个APP里，可以启用相应的MOOSE自带的模块，也可以自己分别在【app路径/include】与【app路径/src】将模型给初步创建好。
2. Make编译好后才是真正将模型创建好了。

输入文件是.i结尾的文件。我在/home/yp/projects/fuel\_rods文件夹下面建立/Oconee\_Rod\_15309文件夹,表示该文件夹下面的输入文件都是为了模拟Oconee Rod 15309反应堆实验的（你也完全可以不建立文件夹，只需要在fuel\_rods文件夹下即可）。

该实验涉及的参数如下：

《[1]邓超群, 向烽瑞, 贺亚男, 等. 基于MOOSE平台的棒状燃料元件性能分析程序开发与验证[J]. 原子能科学技术, 2021, 55(7): 1296-1303.》



为此，我们需要建立一个名叫step1\_to\_generate\_e.i的输入文件，专门为了生成网格几何文件，（详细请看github文件下面的fuel\_rods/pellets/Oconee\_Rod\_15309/step1\_to\_generate\_e.i）,里面进行了非常详尽的注解。

终端输入《mpirun -n 10 ../../fuel\_rods-opt -i step1\_to\_generate\_e.i --mesh-only Oconee\_Rod\_15309.e》

你就可以得到一个名字叫做《Oconee\_Rod\_15309.e》的几何与网格文件，作为后续的网格文件输入。

# 简单介绍问题

以该[文件](file:///D:\\Postgraduate\\Papers\\MOOSE\\examples\\elastoplasticity.i)为例：

这个文件其实要解决的是二氧化铀燃料芯块的热-力耦合方程。

在MOOSE中，必须要明确要解决的控制方程，如，我这里需要解的是如下两个控制方程：

热传导方程： ‑

力平衡方程： ‑

接下来，推导得出它的弱形式。（详细请看[Step 4 Generate a Weak Form | MOOSE (inl.gov)](https://mooseframework.inl.gov/getting_started/examples_and_tutorials/tutorial01_app_development/step04_weak_form.html)的推导）。【或可以先不管，主要看右边的项】

如热传导方程：

可以写成MOOSE使用的弱形式(移项到一边，这里先忽略边界条件)： =0

需要知道的是，MOOSE所有弱形式方程中的每一项（，热平衡方程中出现了3项）其实都对应好[Kernels]中的一项与 [Materials]中的若干项（如中的*ADHeatConductionTimeDerivative方法对应，而*这两个材料属性则是与 [Materials]中的若干项（也可能没有，看每一个项中有没有材料参数），）具体而言，对应的[Kernels]与如图 1.1所示。

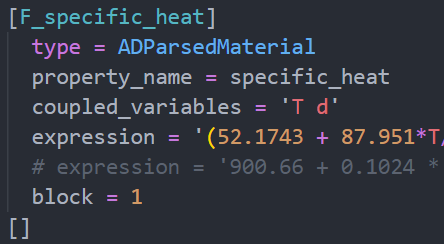
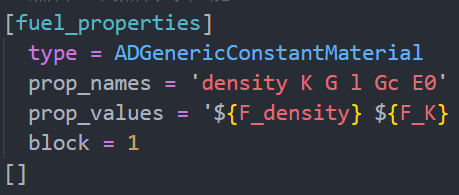
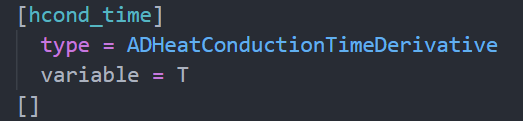


图 1.1 [Kernels]与[Materials]的部分值（左1是[Kernels]的项，右边两个是[Materials]的项）

这里我们再理一下：

1. 所有控制方程
2. 控制方程的弱形式
3. 弱形式的每一项都对应[Kernels]与[Materials]的项（其实还有边界条件的项）

## 几何与网格[Mesh]

### 常用函数

注意这些函数一般只适用于单个简单的情况，复杂的几何需要与其他函数联用。

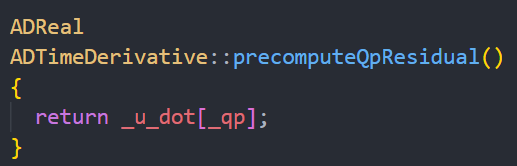
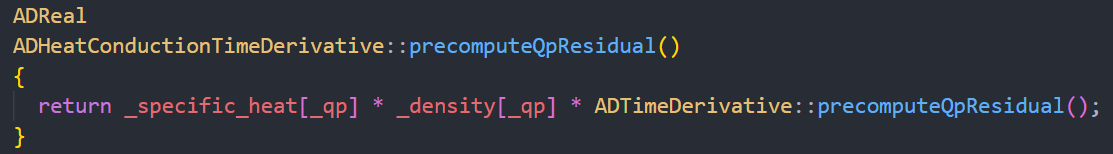
下面是与其他函数联用后的结果展示（都是3维）

详细请见

|  |  |
| --- | --- |
|  |  |
| 环形燃料[AnnularMeshGenerator](https://mooseframework.inl.gov/moose/source/meshgenerators/AnnularMeshGenerator.html)  详细请见fuel\_rods/SomeTests/test0.1\_Mesh/AnnularFuel/ CompleteAnnularPellet3D.i | 棒状燃料[ConcentricCircleMeshGenerator](https://mooseframework.inl.gov/source/meshgenerators/ConcentricCircleMeshGenerator.html)  详细请见fuel\_rods/SomeTests/test0.1\_Mesh/RodFuel/CompleteRodPellet3D.i |

## [Kernels]：

我们找到/moose/modules/heat\_transfer/src/kernels/ADHeatConductionTimeDerivative.C与其调用的ADTimeDerivative.C文件，可以看到计算残差的部分其实与对应的很好。



由于不同的控制方程中有许许多多不同的项，每一个项的形式不一样，为了方便定义与使用，MOOSE里面有非常多[Kernels]以便使用，详细请看[Kernels System | MOOSE](https://mooseframework.inl.gov/moose/syntax/Kernels/)，

本例子中， ADHeatConduction、ADHeatConductionTimeDerivative、HeatSource分别对应

=0

### 一般参数的定义与引用

如同python或者C++，需要命名全局变量或者局部变量一样。在moose输入文件中，我们先将一些材料参数（如热导率，密度等）进行全局变量的定义（此时moose在计算时是不会调用的，需要用特殊的引用格式才能被moose使用）。

首先在此使用全局变量与表达式的进行命名，写在MOOSE的输入文件(.i文件的开头)，例如

density=3980.0*#*kg⋅m-3

thermal\_conductivity=31*#*W⋅m-1⋅K-1

*#* thermal\_conductivity\_dem=0.026#孔隙的导热系数W⋅m-1⋅K-1

elastic\_constants=3.7e11*#*Pa

nu = 0.3

K = '${fparse elastic\_constants/3/(1-2\*nu)}'

G = '${fparse elastic\_constants/2/(1+nu)}'

注意引用的格式一般用 ？ **= '${ elastic\_constants } ${ K}'**

### 其他常用3.1 [Kernels]

## [Materials]：

其实就是将\_specific\_heat[\_qp] 、\_density[\_qp]等[Kernels]需要调用的参数提前先定义好，可以是常数(ADGenericConstantMaterial方法),可以是与变量耦合的公式（如热导率与温度耦合的经验公式ADParsedMaterial方法），甚至与[Materials]本身模块耦合的公式（ADDerivativeParsedMaterial方法）

下面是[Materials]里面常用的模块与例子：

[fuel\_properties]

type = ADGenericConstantMaterial

prop\_names = 'density K'

prop\_values = '${F\_density} ${F\_K}'

block = 1

[]

在 ADGenericConstantMaterial中，使用

prop\_names = 'x1 x2 x3 x4'与

prop\_values = '${y1} ${y2} y3 y4'

以确定材料参数 x1,x2,x3等值。其中${y1}表示y1是定义的全局变量（即在输入文件开头的density=3980.0*#*kg⋅m-3、nu = 0.3、K = '${fparse elastic\_constants/3/(1-2\*nu)}'、G = '${fparse elastic\_constants/2/(1+nu)}'等

）。

[base\_thermal\_conductivity]

type = ADParsedMaterial

property\_name = base\_conductivity

coupled\_variables = 'T'  *#* T in Kelvin

*#* 使用Harding-Martin表达式

block = 1

expression = '100/(7.5408 + 17.692\*T/1000 + 3.6142\*(T/1000)^2) + 6400/((T/1000)^2.5)\*exp(-16.35/(T/1000))'

[]

在ADParsedMaterial中，使用property\_name确定材料参数的名字。耦合变量为coupled\_variables，公式使用expression。（expression的乘法为\*，次方为^）

*#*热导率

[thermal\_conductivity]

type = ADDerivativeParsedMaterial

property\_name = thermal\_conductivity

coupled\_variables = 'd'

material\_property\_names = 'base\_conductivity'

expression = '(1-d)\*base\_conductivity + d\*base\_conductivity\*0.1'

block = 1

[]

在ADDerivativeParsedMaterial中，也和ADParsedMaterial类似，只是它多了与已经定义好的其他材料参数耦合的量material\_property\_names。

### 常用Materials总结

1. addRequiredParam - 用于定义**必需参数**，使用此方法时不能提供默认值，因为它的目的就是强制用户提供这个参数值
2. addParam - 用于定义**可选参数**，可以(也应该)提供默认值

## 输出调试

有时我们需要在终端输出一些值以确保程序正确运行，目前统计有以下几个常用方案：

方法一：仅在特定时间步和进程上输出

（在.C文件里直接加入类似的语句：）

### 仅在特定时间步和进程上输出

在实际应用中，推荐使用方法一，通过仅在主进程和特定时间步输出，大大减少输出量

*// 仅在主进程和每10个时间步输出一次*

  if (\_fe\_problem.processor\_id() == 0 && \_fe\_problem.timeStep() % 10 == 0)

  {

    \_console << "\n============ 蠕变率计算 (时间步: " << \_fe\_problem.timeStep() << ") ============" << std::endl;

    \_console << "  有效应力: " << effective\_stress << std::endl;

    \_console << "  温度: " << \_temperature[\_qp] << std::endl;

    \_console << "  氧超量: " << \_x[\_qp] << std::endl;

    \_console << "  热蠕变1 (低应力扩散): " << creep\_th1 << std::endl;

    \_console << "  热蠕变2 (高应力位错): " << creep\_th2 << std::endl;

    \_console << "  辐照蠕变: " << creep\_ir << std::endl;

    \_console << "  总蠕变率: " << creep\_th1 + creep\_th2 + creep\_ir << std::endl;

    \_console << "===================================" << std::endl;

  }

  \_plastic\_strain[\_qp] = \_plastic\_strain\_old[\_qp] + delta\_ep \* \_Np[\_qp];

*//打印出塑性应变，只打印出一部分，并且只需要值*

*// 提取 (0,0) 分量的实数值*

  Real eps\_pp\_xx = raw\_value(\_plastic\_strain[\_qp](0,0));

*// 计算 L2 范数：sqrt(plastic\_strain : plastic\_strain)*

  Real norm\_pp = std::sqrt(raw\_value(\_plastic\_strain[\_qp]

                                       .doubleContraction(\_plastic\_strain[\_qp])));

  \_console << "plastic\_strain\_xx = " << eps\_pp\_xx

           << ", ||plastic\_strain|| = " << norm\_pp << std::endl;

### 使用PostprocessorInterface输出至CSV文件

如果需要更详细分析，推荐使用方法二，将数据输出到CSV文件

.h文件中写上：

*// 添加后处理器变量声明*

  PostprocessorValue & \_creep\_rate1\_pp;

  PostprocessorValue & \_creep\_rate2\_pp;

  PostprocessorValue & \_creep\_rate3\_pp;

  PostprocessorValue & \_total\_creep\_rate\_pp;

  PostprocessorValue & \_residual\_pp;

.C文件写上：

*// 在源文件中添加后处理器相关代码*

InputParameters

SmallDeformationUO2CreepModel::validParams()

{

  InputParameters params = SmallDeformationJ2Plasticity::validParams();

*// ... 现有代码 ...*

*// 添加后处理器声明*

  params.addParam<PostprocessorName>("creep\_rate1\_pp", "热蠕变1后处理器名称");

  params.addParam<PostprocessorName>("creep\_rate2\_pp", "热蠕变2后处理器名称");

  params.addParam<PostprocessorName>("creep\_rate3\_pp", "辐照蠕变后处理器名称");

  params.addParam<PostprocessorName>("total\_creep\_rate\_pp", "总蠕变率后处理器名称");

  params.addParam<PostprocessorName>("residual\_pp", "径向返回残差后处理器名称");

  return params;

}

SmallDeformationUO2CreepModel::SmallDeformationUO2CreepModel(const InputParameters & parameters)

  : SmallDeformationJ2Plasticity(parameters),

    PostprocessorInterface(this),

*// ... 现有成员初始化 ...*

    \_creep\_rate1\_pp(isParamValid("creep\_rate1\_pp") ?

                   getPostprocessorValue("creep\_rate1\_pp") :

                   getPostprocessorValueByName("dummy")),

    \_creep\_rate2\_pp(isParamValid("creep\_rate2\_pp") ?

                   getPostprocessorValue("creep\_rate2\_pp") :

                   getPostprocessorValueByName("dummy")),

    \_creep\_rate3\_pp(isParamValid("creep\_rate3\_pp") ?

                   getPostprocessorValue("creep\_rate3\_pp") :

                   getPostprocessorValueByName("dummy")),

    \_total\_creep\_rate\_pp(isParamValid("total\_creep\_rate\_pp") ?

                        getPostprocessorValue("total\_creep\_rate\_pp") :

                        getPostprocessorValueByName("dummy")),

    \_residual\_pp(isParamValid("residual\_pp") ?

                getPostprocessorValue("residual\_pp") :

                getPostprocessorValueByName("dummy"))

{

}

### 简化输出，使用标记点输出

如果只是想快速调试，可以使用方法三，简化输出内容并使用特殊符号标记

// 只输出一个简单的总结信息

\_console << "\n★ 蠕变率: " << total\_rate

<< " (T=" << \_temperature[\_qp]

<< ", x=" << \_x[\_qp]

<< ", σ=" << effective\_stress << ")" << std::endl;

# Cursor的经验之谈

Cursor是ai辅助写代码的神器，但是现在咱们几乎还没用上它的ai功能，本章将介绍它的一些基础功能，目标是在它的辅助下，实现任一一个自动化脚本的编写

# Include与scr自定义函数

MOOSE中确实有非常多非常好用的模块与对应的函数，但是为了解决特殊问题，我们总是会用到MOOSE里面没有的函数，这一章重点介绍自定义函数。

这里面

# 附录

## Linux基础命令

1. 明确【终端位置】与【文件位置】的区别与联系

注意打开Linux终端或者Windows终端后，其实都是有一个对应位置的，如Windows终端显示【(base) PS C:\Users\xxxxx>】，那么可以确定该终端目前在C盘下的Users\xxxxx文件夹下。如果启用相关命令，一定是基于该路径进行的操作（环境变量除外）。这是必要是基础知识。如linux终端显示【(base) yp@LAPTOP-R6KJ1K05:~/projects/moose/test$】那么同理，该终端现在位于~/projects/moose/test文件夹下。

1. 切换终端位置，cd相关命令（其实一般的切换目录命令直接使用Cursor软件自带的功能点击就很方便了，但是还是得明确文件位置与切换这个概念，因为运行程序时用的上）

假设有这么一个文件路径/home/yp/projects/a1/b2/c3,终端位置在/home/yp/projects

* 其他任意目录

《cd /home》终端位置切换到 /home 目录:

* 子目录

《cd a1/b2/c3》终端位置切换到子文件夹(即/home/yp/projects/a1/b2/c3)

* 本目录

《cd ./》终端位置切换到当前目录（啥事都没做）

* 上级目录

《cd ../》终端位置切换到上级目录(即/home/yp)

《cd ../../》终端位置切换到上上级目录（即/home）

1. MOOSE中最常用到的地方
2. 例子1，简单例子（如下图）：

当前linux终端位置为：~/projects/moose/examples/ex01\_inputfile$

要想跑MOOSE程序，那么得分别知道模型文件(即-opt文件)的与输入文件(即.i文件)的相对于目前终端的相对位置，

-opt文件（ex01-opt）在/home/yp/projects/moose/examples/ex01\_inputfile/，即当前文件夹(./ex01-opt)

.i文件（ex01.i）在/home/yp/projects/moose/examples/ex01\_inputfile/,当前文件夹(./ex01.i)【，其实./也可以省略，因为./ex01-opt -i ./ex01.i命令下，./ex01.i部分默认为路径，所以当前文件夹下，./也可不加。但是由于第一项作为命令的开始有许多限制，而不能省略./】

所以命令为《./ex01-opt -i ./ex01.i》或《./ex01-opt -i ex01.i》

1. 例子2，困难例子如下图：

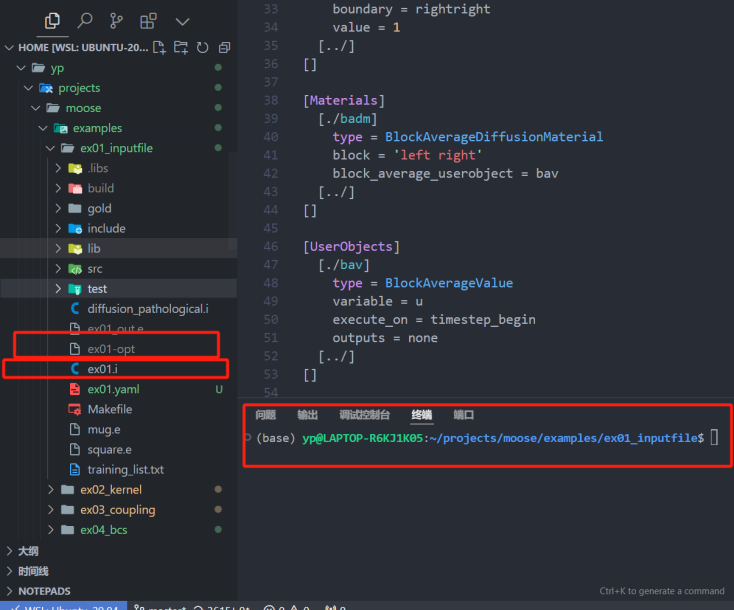
当前linux终端位置为：~/projects/raccoon/verification\_granite/model1/elastoplasticity\_out\_cp$

要想跑MOOSE程序，那么得分别知道模型文件(即-opt文件)的与输入文件(即.i文件)的相对于目前终端的相对位置，

-opt文件（raccoon-opt）在/yp/projects/raccoon/raccoon-opt，即相对位置为上上上级目录（../../../raccoon-opt）

.i文件（elastoplasticity.i）在yp/projects/raccoon/verification\_granite/model1/elastoplasticity.i即相对位置上级目录（../elastoplasticity.i）

于是命令就变成了../../../raccoon-opt -i ../elastoplasticity.i



左图为例子1 右图为例子2

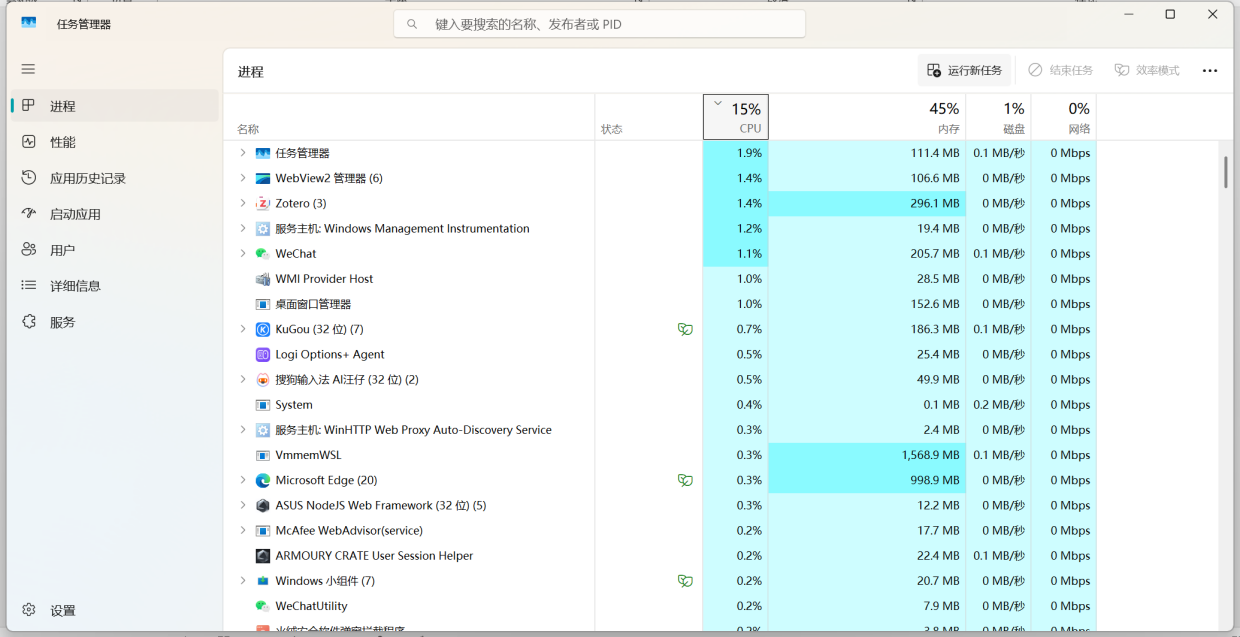
## 多线程计算

目前用到多线程计算的地方有最少3个：

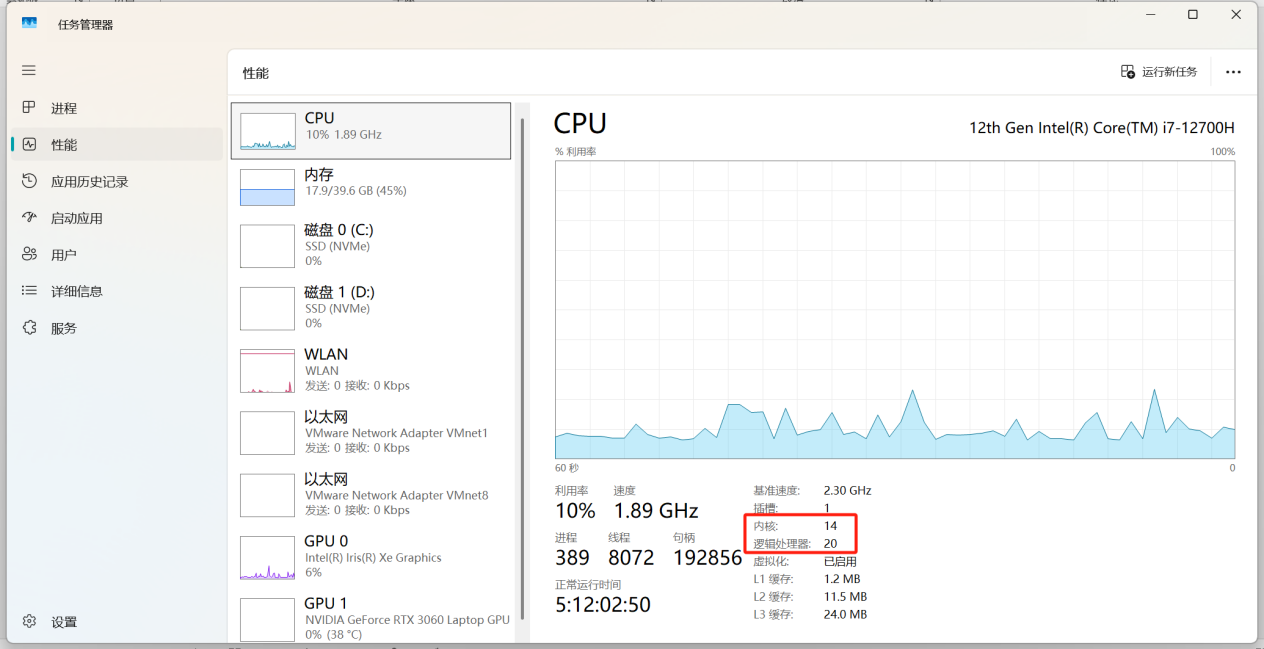
1. 《make -j x》
2. 《./run\_tests -j x》
3. 《mpirun -n x ./ex01-opt -i ./ex01.i --n-threads=y》其中标蓝的地方，即x(处理器),y（线程）的选取，需要进一步考虑！数字代表的是计算使用的核数与线程，合适的x,y可大大提高运算速度。[具体请参考原文](https://mooseframework.inl.gov/getting_started/examples_and_tutorials/tutorial01_app_development/step07_parallel.html" \l "step-7-execute-in-parallel)。

具体分为如下几步：

1. 键盘同时按住左下角的CTRL键+SHIFT键+左上角的ESC键（没有弹出窗口可以先按住前两个，后按ESC键）



1. 点击性能，



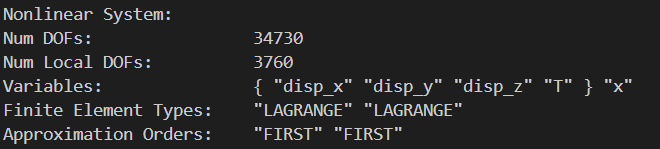
1. 内核14，这是个关键参数，这是我们能取的最大值。我的经验是y取1，x得小于内核数，如果你跑程序时还需要用到它干很多事情，那么可以取10，甚至取8。

《mpirun -n 10 ./ex01-opt -i ./ex01.i --n-threads=1》这将大幅提高运行速度。

### 更加专业的确定方法：

1. 结论：
   * + [**MOOSE开发人员倾向于认为，20000是单个流程可能负责的理想的DOF数量（Local DOFs）。**](https://mooseframework.inl.gov/getting_started/examples_and_tutorials/tutorial01_app_development/step07_parallel.html#step-7-execute-in-parallel)但当出现以下情况时，可使用更低DOFs/process：
       - 强非线性问题（需要更多牛顿迭代）
       - 多物理场耦合（如热-力-相场耦合）
       - 内存限制（每个进程DOFs超过内存容量）
     + 越多核心数越好，但核心数多到一定程度后会达到一个阈值。
     + 线程数的加速效果相当有限。在只有一个核心时加速才明显。
2. 具体实操：

每当运行一个程序时，终端的将会出现类似下面的信息：



其中的Num Local DOFs是用总自由度Num DOFs除以使用的内核数，官网提示这个值小于20000为佳。但是遇到多物理场、强非线性时，可继续降低。

1. 根据测试，我使用的20核28逻辑处理器的台式电脑：

热力化学耦合（**强非线性问题+多物理场耦合**）的情况下的测试：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 核心数 | 线程数 | Local DOFs | 运行时间(s) | 加速比 | 效率(%) |
| 1 | 1 | 34730 | 170.60 | 1.00 | 100% |
| 1 | 2 | 34730 | 146.50 | 1.16 | 58% |
| 1 | 3 | 34730 | 142.70 | 1.20 | 40% |
| 2 | 1 | 17365 | 71.80 | 2.38 | 119% |
| 2 | 2 | 17365 | 63.80 | 2.67 | 67% |
| 2 | 3 | 17365 | 62.50 | 2.73 | 45% |
| 3 | 1 | 11577 | 43.10 | 3.96 | 132% |
| 3 | 2 | 11577 | 39.40 | 4.33 | 72% |
| 3 | 3 | 11577 | 41.40 | 4.12 | 46% |
| 4 | 1 | 8683 | 32.90 | 5.19 | 130% |
| 4 | 2 | 8683 | 34.20 | 4.99 | 62% |
| 4 | 3 | 8683 | 32.60 | 5.23 | 44% |
| 5 | 1 | 6946 | 24.80 | 6.88 | 138% |
| 5 | 2 | 6946 | 26.20 | 6.51 | 65% |
| 6 | 1 | 5788 | 24.00 | 7.11 | 118% |
| 6 | 2 | 5788 | 23.70 | 7.20 | 60% |
| 7 | 1 | 4961 | 22.90 | 7.45 | 106% |
| 7 | 2 | 4961 | 22.60 | 7.55 | 54% |
| 8 | 1 | 4341 | 21.90 | 7.79 | 97% |
| 8 | 2 | 4341 | 21.00 | 8.12 | 51% |
| 9 | 1 | 3859 | 21.30 | 8.01 | 89% |
| 9 | 2 | 3859 | 19.40 | 8.79 | 49% |
| 10 | 1 | 3473 | 20.80 | 8.20 | 82% |
| 10 | 2 | 3473 | 19.40 | 8.79 | 44% |

因此，推荐尽可能用多一些内核数，而线程数给个1即可。

## Restart and Recover

当我们运行的程序迫不得已需要中止时，我们就需要使用Recover命令了（得提前设置部分参数）。[具体见官网](https://mooseframework.inl.gov/releases/moose/2024-03-08/application_usage/restart_recover.html)。

### 具体设置

[Outputs]

  exodus = true #表示输出exodus格式文件

  [my\_checkpoint]

    type = Checkpoint

    time\_step\_interval = 5    # 每5个时间步保存

    num\_files = 4             # 保留最近4个检查点

    wall\_time\_interval = 3600 # 每小时保存一次（秒）

  []

  file\_base = 'outputs/test1\_RestartAndRecover/step2\_onlyPellet3D'

[]

绿色字体是与程序中断后的恢复有关的代码。其中标注较为清楚。file\_base后面的是一串路径，输出.e文件与恢复文件xxxxxx\_cp都是在这个路径之下。

在[Outputs]设置好后，正常运行程序时，程序就可以保存检查点文件。这些文件格式如下

# 典型检查点目录结构

xxx\_cp/

├── 0005-mesh.cpa.gz # 网格数据（所有进程共享）

├── 0005-restart-0.rd# 进程0的数据

├── 0005-restart-1.rd# 进程1的数据

├── 0005-restart-2.rd# 进程2的数据

├── ...

├── 0010-mesh.cpa.gz

├── 0010-restart-1.rd

└── ...

其中0005-restart-x.rd的x的最大值直接与你的所指定的核数相关。即

《mpirun -n x ./ex01-opt -i ./ex01.i --n-threads=1》

下一次需要恢复时，直接在《mpirun -n x ./ex01-opt -i ./ex01.i --n-threads=1》后加—recover即可，

运行命令为

《mpirun -n x ./ex01-opt -i ./ex01.i --n-threads=1 --recover》

这将从最新的检查点出发继续运行。

## GPU并行

难度看起来有些大，基本没有看到完整的教程

## 常见报错

### 代码格式问题

(moose) yp@LAPTOP-R6KJ1K05:~/projects/fuel\_rods/pellets/Wu2021/input\_files$ mpirun -n 4 ../../../fuel\_rods-opt -i step3\_ThermalCouple2.i --n-threads=2

===================================================================================

=   BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES

=   PID 235217 RUNNING AT LAPTOP-R6KJ1K05

=   EXIT CODE: 9

=   CLEANING UP REMAINING PROCESSES

=   YOU CAN IGNORE THE BELOW CLEANUP MESSAGES

===================================================================================

YOUR APPLICATION TERMINATED WITH THE EXIT STRING: Segmentation fault (signal 11)

This typically refers to a problem with your application.

Please see the FAQ page for debugging suggestions

原因：这里没有打注释，一遇到这个报错，你应该庆幸了，因为问题很快就能被发现且容易解决。

