第 122 期

PyControl 完全指南

用 Python 实时掌控油藏模拟

——从入门到精通——

CMG STARS · Python · 外部控制技术详解

油藏数模智能化 | 原创技术分享

目 录

右键目录，选择"更新域"刷新页码

一、什么是 PyControl

PyControl 是 CMG（Computer Modelling Group）公司为其油藏模拟软件提供的一套外部控制接口。它允许工程师使用 Python 脚本在模拟运行过程中实时读取数据、做出决策，并动态调整模拟参数。

简单来说，PyControl 就像给模拟器装上了一个"智能大脑"。传统的油藏模拟一旦开始运行，只能按照预设的参数一步步计算下去。而有了 PyControl，你可以在模拟运行的任意时刻插入 Python 代码，实现以下功能：

- 实时读取油藏数据（压力、温度、饱和度、产量等）
- 根据实时数据自动调整生产制度（关井、开井、修改约束条件）
- 动态优化时间步长（在关键节点提高计算精度）
- 接入外部优化算法（如遗传算法、强化学习）自动寻优

1.1 PyControl 的技术背景

油藏数值模拟是石油工程领域的核心技术之一。传统的模拟流程是"离线"的：工程师设定好所有参数后提交计算，等待数小时甚至数天后才能看到结果。如果结果不理想，需要修改参数重新计算。这种"试错式"的工作模式效率极低。

PyControl 的出现改变了这一格局。它基于 CMG 的 OUTBOARD 接口技术，在每次时间步计算完成后进行"握手"（Handshake），将控制权暂时交给外部 Python 脚本。Python 脚本可以读取当前时刻的所有模拟数据，执行任意逻辑，然后将新的控制指令回传给 STARS。

1.2 PyControl 的核心优势

核心优势

PyControl 最大的优势在于将油藏模拟从"离线计算工具"升级为"实时决策平台"。工程师不再只是被动等待结果，而是可以主动干预模拟过程，实现模拟与决策的闭环。

与传统方法相比，PyControl 的核心优势包括：

特性	传统方式	PyControl 方式
数据读取	模拟结束后才能查看	每个时间步实时获取
参数调整	停算 → 改文件 → 重跑	运行时即时下发指令
优化策略	人工试错，耗时巨大	可接入 AI 算法自动寻优
多井联控	逐井手动调整	批量自动控制

1.3 适用人群

PyControl 主要面向以下人群：

- 油藏工程师：需要高效完成历史拟合和生产预测
- 数值模拟专家：需要实现复杂的自动化控制策略
- 数据科学家：希望将机器学习算法应用于油藏优化
- 研究人员：需要灵活的实验平台验证新的开发理论

学习前提

阅读本文需要具备以下基础知识：Python 编程入门、油藏数值模拟基本概念（网格、流体、井筒）、

STARS 软件基本操作。如果你还不熟悉这些内容，建议先补充相关知识。

二、为什么需要 PyControl

在深入了解 PyControl 之前，让我们先来看一个真实的场景。这是无数油藏工程师日常工作的一个缩影。

2.1 一个典型的凌晨加班场景

凌晨两点，某油田研究院的模拟室里，工程师小李盯着屏幕上已经跑了三个小时的 STARS 模拟进程。这是他第 17 次尝试历史拟合。

前 16 次，他每次都要经历这样的流程：调整渗透率参数 → 保存 .dat 文件 → 提交计算 → 等待 3 小时 → 对比结果 → 发现偏差 → 重新调整。三天下来，只完成了不到 10% 的工作量。

"如果能让模拟器在运行过程中自己判断、自己调整就好了"，小李这样想。

这正是 PyControl 要解决的问题。

2.2 传统方法的四大痛点

痛点一：历史拟合效率低下

历史拟合（History Matching）是油藏模拟中最耗时的环节。工程师需要不断调整地质参数（渗透率、孔隙度、相渗曲线等），使模拟结果与实际生产历史尽可能吻合。每一次参数调整都需要重新提交计算，等待数小时甚至数天。

使用 PyControl，工程师可以在 Python 中定义目标函数（如模拟产量与实际产量的误差），在模拟运行的每个时间步计算误差并自动调整参数，实现"边跑边调"的实时优化。

痛点二：生产制度调整滞后

在实际油田开发中，生产井的工作制度需要根据实时监测数据动态调整。例如，当某口井的含水率超过阈值时需要降低产量，当地层压力下降过快时需要补充能量。

传统方法只能基于固定的时间节点调整制度，无法实现真正的"实时响应"。PyControl 可以在每个时间步检查关键指标，一旦触发条件立即执行预设的控制策略。

痛点三：智能化程度不足

随着人工智能技术的发展，越来越多的研究者尝试将机器学习、强化学习等先进算法应用于油藏优化。然而，传统的油藏模拟软件"封闭"的架构使得这些算法难以直接接入。

PyControl 提供了一个开放的接口，使得 Python 生态中的各种 AI 库（TensorFlow、PyTorch、scikit-learn 等）可以直接与 STARS 交互，为油藏智能化的研究和应用铺平了道路。

痛点四：多井协调困难

现代油田往往有数十口甚至上百口生产井，各井之间的生产制度相互影响。人工逐井调整不仅效率低下，而且难以把握全局最优。

通过 PyControl，可以编写统一的控制策略脚本，同时监控和调控所有井的生产状态，实现井群的协调优化。

2.3 PyControl 能带来的改变

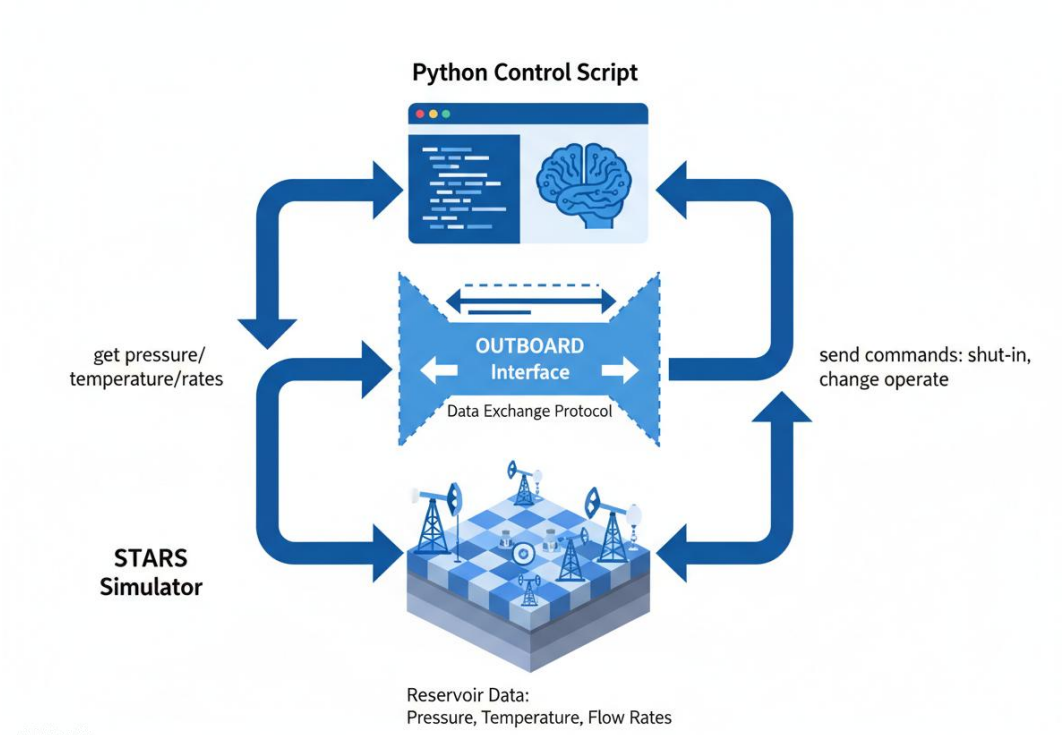
> **效率提升**
>
> 根据实际项目经验，使用 PyControl 进行自动化历史拟合，可以将原本需要数周的工作缩短到数天。某油田项目采用 PyControl + 遗传算法进行注采参数优化，在保持相同精度的情况下，计算时间减少了约 60%。

更重要的是，PyControl 不仅仅是效率工具，它改变了油藏工程师的工作方式。工程师可以将更多精力放在策略制定和结果分析上，而不是重复性的参数调试。

三、PyControl 工作原理

要真正掌握 PyControl，必须先理解它的工作机制。PyControl 本质上是一个三方耦合系统：STARS 模拟器 ↔ OUTBOARD 接口 ↔ Python 控制脚本。

3.1 三方架构解析



PyControl 三方耦合架构图

整个系统由三个核心组件构成，各自承担不同的职责：

STARS 模拟器 —— 计算引擎

STARS 是 CMG 的热采和高级过程模拟器，负责执行每一步的地质力学和流体流动计算。它的核心任务包括：求解多相流体在多孔介质中的流动方程、计算传热传质过程、更新网格属性（压力、温度、饱和度等）。

在 PyControl 模式下，STARS 每完成一个时间步的计算后，会主动暂停，等待外部指令。

OUTBOARD 接口 —— 通信桥梁

OUTBOARD 是 CMG 提供的外部接口层，它在 STARS 和 Python 之间建立了一条双向通信通道。具体来说，OUTBOARD 负责以下工作：

- 在每个时间步完成后触发"握手"（Handshake），暂停 STARS 的计算
- 将 STARS 的内部数据暴露给 Python（通过共享内存或 IPC 机制）
- 接收 Python 下发的控制指令并回传给 STARS
- 确保数据的一致性和同步性

Python 脚本 —— 智能大脑

Python 脚本是整个系统的"控制器"。它包含一个名为 compute() 的回调函数，每次握手时被 PyControl 自动调用。在 compute() 函数中，工程师可以编写任意的业务逻辑：

- 读取当前的模拟数据（通过 Sim_Data 对象）
- 执行判断和计算（如检查阈值、计算误差、运行优化算法）
- 下发控制指令（通过 Info_to_Sim 对象）

3.2 握手机制详解

"握手"是 PyControl 工作的核心机制。它的时序如下：

```
握手时序

Step 1: STARS 完成当前时间步 [t_n -> t_n+1] 的计算
Step 2: OUTBOARD 触发 Handshake，暂停 STARS
Step 3: PyControl 调用 Python 的 compute(sim_data, info_to_sim)
Step 4: Python 脚本执行逻辑（读取数据、判断、下发指令）
```

```
Step 5: OUTBOARD 将指令传回 STARS
Step 6: STARS 根据指令更新参数，继续下一步计算
```

这个循环在每个时间步重复进行，直到模拟结束。Python 脚本通过检查当前模拟时间（cur_time）来决定在不同的时间点执行不同的操作。

3.3 核心 API 介绍

CMG 提供了两个核心 Python 类用于与 STARS 交互。掌握这两个类的用法是编写 PyControl 脚本的基础。

Sim_Data —— 数据读取接口

```
Sim_Data 常用方法

sim_data.get_current_handshake_time() -> float
    获取当前握手时间（模拟天数）

sim_data.Grid(range_str).get_property(GRIDPROP.PRESSURE, time) -> dict
    查询指定网格区域的压力场，返回 {cell_id: pressure}

sim_data.Well(name).get_property(WELLPROPS.xxx) -> float
    查询单井属性（井底流压、产量、含水率等）

sim_data.Layer(well, layer).get_property(PHASEPROPS.xxx, PHASETYPE.xxx)
    查询指定井指定层的分相流量

sim_data.Sector(name).get_property(SECTORPROPS.xxx, ...)
    查询指定区域的累计量（地质储量、采出量等）

sim_data.Well(name).shut_in()
    关闭指定井（写操作，立即生效）
```

Info_to_Sim —— 指令下发接口

```
Info_to_Sim 常用方法

info_to_sim.update_well_change_first_timestep_size(dt: float)
    修改下一个时间步的步长（提高关键节点的计算精度）

info_to_sim.send(command_str: str)
    向 STARS 发送原始关键字命令
    示例：info_to_sim.send("*OPERATE *MIN *BHP 2300")
    注意：命令语法必须与 .dat 文件中的格式一致
```

安全提醒

send() 方法直接发送原始 STARS 关键字字符串，不做任何语法检查。一个拼写错误（如 *OPREATE 而非 *OPERATE）就会导致模拟崩溃。建议先在独立的 .dat 文件中验证命令的正确性，再写入 Python 脚本。

四、SAGD 模型详解

在深入解析 Python 脚本之前，我们需要先理解"控制对象"——STPYS001.DAT 所描述的 SAGD 油藏模型。只有理解了模型的物理含义，才能理解 Python 脚本中每一步操作的目的。

4.1 SAGD 工艺原理

SAGD（Steam Assisted Gravity Drainage，蒸汽辅助重力泄油）是开采稠油（Bitumen）的核心技术。它的工作原理可以通俗地理解为"地下蒸汽烤箱"：

首先，通过上部的水平注入井向油藏注入高温高压蒸汽（通常 200-250°C），在注入井周围形成一个蒸汽腔（Steam Chamber）。蒸汽冷凝释放的热量加热周围的稠油，使其粘度急剧降低。被加热变稀的原油在重力作用下向下流动，最终被下部的水平生产井采集。

SAGD 成功的关键在于温度。本模型中的沥青（Bitumen）在原始油藏温度（10°C）下粘度高达 5,000,000 cp（厘泊），比蜂蜜还要稠。但在 200°C 时，粘度骤降至约 10 cp，几乎像水一样流动。这种粘度随温度急剧变化的特性，是 SAGD 工艺可行性的物理基础。

4.2 网格系统

数值模拟的第一步是将连续的油藏空间离散化为有限的网格块。本模型采用笛卡尔坐标系，定义如下：

```
网格定义（STPYS001.DAT）

GRID CART 11 10 10          // 11 列 x 10 行 x 10 层 = 1100 个网格块
DI CON 2                     // X 方向间距：常数 2m
DJ JVAR 300. 100. ... 300.  // Y 方向间距：中间密(100m)两边疏(300m)
DK KVAR 1. 1. ... 2. 2.     // Z 方向间距：上 5 层 1m，下 5 层 2m
POR CON 0.34                 // 孔隙度：34%
PERMI CON 2200               // X 方向渗透率：2200 mD
PERMK EQUALSI * 0.8         // Z 方向渗透率 = X 方向 x 0.8
```

网格设计体现了工程师的建模意图：X 方向间距较小（2m）是为了精细刻画井筒附近的流动；Y 方向中间密两边疏是为了在井对附近提高分辨率；Z 方向上密下疏是为了更好地捕捉蒸汽腔的垂向扩展。

4.3 流体模型

本模型采用三组分、三相的流体体系：

组分	化学式	相态	关键属性
水(WATER)	H2O	水相	载热介质
沥青(BITUMEN)	复杂烃类	油相	粘度对温度极敏感
甲烷(CH4)	CH4	气相	溶解气，影响流动性

模型中最关键的输入是粘度-温度查找表（VISCTABLE），它直接定义了沥青在不同温度下的流动能力：

```
粘度-温度表（节选）

温度(°C)     沥青粘度(cp)     描述
    10        5,000,000      极稠，几乎不流动
    50            8,028      稠，流动性差
   100              213      较稀，可流动
   200               10      接近水的粘度
   300                3      极易流动
```

4.4 井筒定义

模型定义了 4 口井，构成两组 SAGD 井对：

井名	类型	层位	操作制度	备注
injtb	注入井（油管）	第 5 层	MAX BHP 5500, STW 400	注汽温度 224°C
injan	生产井（套管）	第 5 层	MAX STL 500, MIN BHP 2480	上层生产
prdtb	注入井（油管）	第 2 层	MAX BHP 5500, STW 400	注汽温度 224°C
prdan	生产井（套管）	第 2 层	MAX STL 500, MIN BHP 2480	下层生产

注意 injtb + injan 位于第 5 层（中层），prdtb + prdan 位于第 2 层（更深层），构成了上下两组 SAGD 井对。Python 脚本后续会在 t=75 天时关闭 injan 和 prdtb，并重新激活 prdan。

五、Python 脚本完全解析

现在进入本文的核心部分——对 stpyc001.py 的完整逐行解析。我们将按照时间线，详细分析脚本在 5 个关键时间节点的操作逻辑。

5.1 脚本骨架

整个脚本的核心是一个 compute() 函数，每次握手时被 PyControl 自动调用。它的基本结构如下：

```
stpyc001.py — 骨架

def compute(sim_data: Sim_Data, info_to_sim: Info_to_Sim) -> None:
    cur_time = sim_data.get_current_handshake_time()
    if cur_time <= 0.0:
        return                      // 跳过初始时刻


    if cur_time == 40.0:        // 网格压力巡检
        ...
    elif cur_time == 60.0:      // 单井压力监控
        ...
    elif cur_time == 75.0:      // 核心控制操作（一键三连）
        ...
    elif cur_time == 100.0:     // 分层流量读取
        ...
    elif cur_time == 110.0:     // 全场地质储量
        ...
```

脚本的逻辑非常清晰：通过 if/elif 判断当前模拟时间 cur_time，在不同的时间点执行不同的操作。这种"时间触发"的设计模式是 PyControl 脚本的标准写法。

5.2 t = 40 天：网格区域压力巡检

模拟运行到第 40 天时，脚本决定"抽查"一下指定区域的压力状况。

```
t = 40.0 — 网格压力巡检

range_prop, range_index = sim_data.Grid("1 4 2:5")
    .get_property(GRIDPROP.PRESSURE, cur_time)
for cell, cell_pres in range_prop.items():
    print(f"Pressure for grid cell '{cell}': {cell_pres:10.2f}")
```

这里 Grid("1 4 2:5") 选取了 I=1~4、J=2~5 的区域，共 4x4x10 = 160 个网格块。get_property() 返回一个字典，键为网格块标识（如 "3 5 7" 表示第 3 列第 5 行第 7 层），值为对应的压力值。

代码解读

这一步相当于"无人机巡检"——派出数据探测器检查指定区域的地层压力分布是否均匀，是否存在异常高压或低压区。巡检结果通过 print() 输出，可以在 STARS 的日志文件（.log）中查看。

5.3 t = 60 天：单井压力监控

到了第 60 天，脚本把焦点缩小到单井 prdtb，直接查询该井的井底流压。

```
t = 60.0 — 单井压力监控

wbl_pres = sim_data.Well("prdtb").get_property(
    WELLPROPS.WELL_REF_LAYER_BLOCK_PRESSURE)
print(f"Well Block Pressure for well 'prdtb': {wbl_pres:10.2f} "
      f"@ time {cur_time:7.2f}")
```

WELL_REF_LAYER_BLOCK_PRESSURE 返回的是井筒所在参考网格块的压力值，对于生产井来说就是井底流压（BHP）。监控该值可以判断井是否在合理的工作制度下运行——如果流压低于设定的 MIN BHP，就需要减产或关井保护。

5.4 t = 75 天：核心控制操作（一键三连）

这是整个脚本最精彩的部分！模拟进行到第 75 天时，Python 脚本完成了三个连锁操作：

```
t = 75.0 — 关键控制操作

print(f"Updating recurrent data @ time {cur_time:7.2f}")


// 操作 1：缩小下一个时间步长
info_to_sim.update_well_change_first_timestep_size(0.1)


// 操作 2：紧急关井
sim_data.Well("injan").shut_in()
sim_data.Well("prdtb").shut_in()


// 操作 3：重新定义 prdan 的生产制度
info_to_sim.send("*PRODUCER 'prdan'")
info_to_sim.send("*OPERATE *MIN *BHP 2300.0")
info_to_sim.send("*OPERATE *MAX *STL 500.0")
```

三个操作的详细解读：

操作 1：缩小时间步长

在 STARS 中，时间步长是自适应调整的。但在井况突变（关井+开井）时，默认步长可能过大而错过关键动态。update_well_change_first_timestep_size(0.1) 将下一个步长强制设为 0.1 天（默认最大 15 天），确保剧烈变化被精确捕捉。

操作 2：紧急关井

shut_in() 是写操作，调用后立即生效。被关闭的井在当前时间步后停止生产或注入。这里关闭了第一对 SAGD 井对（injan 和 prdtb），可能原因是该井对周围的油藏已被充分采出，需要调整开发策略。

操作 3：重新定义生产制度

send() 方法允许发送任意 STARS 关键字命令。这里将 prdan 重新定义为生产者，并设定了两个约束条件：MIN BHP 2300（流压不能低于 2300 kPa，比原来的 2480 更宽松）和 MAX STL 500（日产液量上限 500 m³）。这意味着该井可以在更低的压力下继续生产。

> **重要提醒**
>
> send() 发送的字符串直接传给 STARS 解析器，不做语法检查。务必确保命令拼写正确、格式规范。建议在独立的 .dat 文件中先手动验证命令，确认无误后再写入 Python 脚本。shut_in() 是不可逆操作——一旦调用，该井在当前模拟中将被永久关闭（除非后续用 send() 重新激活）。

5.5 t = 100 天：分层流量精细化读取

第 100 天时，脚本查询注入井 injtb 在第 5 层的注水速率。

```
t = 100.0 —— 分层流量读取

layer_wat = sim_data.Layer("injtb", 5).get_property(
    PHASEPROPS.STANDARD_CONDITION_RATE, PHASETYPE.WATER)
print(f"Layer STW rate for well 'injtb' layer #5: {layer_wat} "
      f"@ time {cur_time:7.2f}")
```

Layer 对象实现了分层数据查询，是比 Well 更精细的读取粒度。SAGD 过程中，蒸汽主要沿上部层位推进，通过分层监控可以判断蒸汽腔的垂向扩展范围。

STANDARD_CONDITION_RATE 返回标准条件（地面条件）下的体积流量。

5.6 t = 110 天：全场地质储量统计

第 110 天时，脚本统计整个油田的剩余原油地质储量。

```
t = 110.0 —— 全场地质储量

Oil_IP = sim_data.Sector("Entire  Field").get_property(
    SECTORPROPS.PHASE_IN_PLACE, PHASETYPE.OIL,
    CONDITIONS.STANDARD_CONDITION)
print(f"Oil In-Place @ SC for sector 'Entire Field': {Oil_IP} "
      f"@ time {cur_time:7.2f}")
```

Sector("Entire Field") 查询整个模型的统计区域。PHASE_IN_PLACE 返回指定相（原油）的地下体积，结合 STANDARD_CONDITION 条件换算到地面标准条件。这个数据是评估采收率的核心指标。

采收率计算

采收率 =（原始地质储量 - 剩余地质储量）/ 原始地质储量。Python 脚本可以在每个时间点自动计算并记录该值，生成完整的采收率变化曲线，为开发方案评价提供量化依据。

六、工程应用场景

掌握了 PyControl 的基本用法后，让我们来看看它在实际工程中的典型应用场景。这些场景展示了 PyControl 从简单的自动化到复杂智能化的完整能力谱系。

6.1 场景一：动态生产制度优化

这是最基础也是最常用的应用场景。传统的生产制度一旦设定，在整个模拟过程中保持不变。而实际油田开发中，生产制度需要根据油藏动态实时调整。

使用 PyControl，可以实现以下自动化策略：

- 压力保护：当井底流压低于安全阈值时自动降低产量

- 含水控制：当含水率超过设定值时自动关井或减产
- 注采平衡：实时监控全区注采比，动态调整注水量
- 汽窜预防：SAGD 过程中监测蒸汽腔发育，防止汽窜

6.2 场景二：自动化历史拟合

历史拟合是油藏模拟中最耗时、最需要经验的环节。PyControl 可以大幅提高这一过程的效率。

基本思路是：在 Python 中定义目标函数（如模拟产量与实际产量的均方误差），在每次握手时计算误差，然后使用优化算法（如梯度下降、遗传算法、贝叶斯优化）自动调整模型参数。

自动化历史拟合伪代码

```
from scipy.optimize import minimize

def compute(sim_data, info_to_sim):
    cur_time = sim_data.get_current_handshake_time()

    // 读取模拟产量和实际产量
    sim_rate = sim_data.Well("prdan").get_property(
        WELLPROPS.STANDARD_CONDITION_RATE, PHASETYPE.OIL)
    actual_rate = load_actual_data(cur_time)  // 从文件/数据库加载

    // 计算误差并更新优化器
    error = (sim_rate - actual_rate) ** 2
    new_perm = optimizer.step(error)  // 优化器自动调整

    // 将新渗透率下发给 STARS
    info_to_sim.send(f"*PERMI CON {new_perm}")
```

6.3 场景三：智能井群联控

当油田有数十口甚至上百口井时，人工监控每口井的状态是不现实的。PyControl 可以作为"中央控制器"，实现：

- 异常检测：自动扫描所有井的井底流压，发现低于安全阈值的井立即告警或关井
- 注采平衡：实时监控全区注采比，当注水量不足时自动提高注入井的配注
- 汽腔调控：通过分层温度监控判断蒸汽腔扩展状态，动态调整注汽速率防止汽窜
- 多模型协同：同时控制多个 STARS 实例，实现区块级联合模拟

6.4 场景四：数字孪生基础

数字孪生（Digital Twin）是当前石油行业的热点方向。它的核心思想是在数字空间中构建一个与物理油田实时同步的虚拟副本。PyControl 为数字孪生提供了关键的技术基础：

- 实时数据接入：通过 Python 脚本读取实时生产数据，驱动模拟器更新
- 在线更新：根据新钻的井或新采集的数据，动态更新模型参数
- 预测分析：基于当前状态进行短期预测，辅助生产决策

七、最佳实践与常见问题

最后，让我们总结一些使用 PyControl 的最佳实践，以及新手常遇到的问题和解决方案。

7.1 环境准备清单

组件	要求	验证方法
CMG 版本	2020.10 或更高	Launcher → 版本信息
Python	3.8 ~ 3.11	python --version
CMGPy 库	随 CMG 安装	import cmgpy
STARS 许可	含 OUTBOARD/PyControl	运行 STPYS001 测试

7.2 调试技巧

Tip 1：从"只读"开始

先只使用 sim_data.get_xxx() 方法读取数据，不要急于使用 shut_in() 或 send()。确保你能正确读取数据后，再逐步添加写操作。这是一个非常重要的安全原则。

Tip 2：善用 print() 调试

PyControl 会将 Python 的 print() 输出重定向到 STARS 的日志文件（.log）。在关键位置打印变量值，是排查问题的最有效手段。建议在 compute() 函数开头就打印 cur_time，确认脚本被正确调用。

Tip 3：小步快跑

先在简单的测试模型上验证逻辑（粗网格、短时间），确认无误后再上正式模型。这可以大幅降低试错成本。

7.3 常见避坑指南

避坑 1：send() 命令不做语法检查

info_to_sim.send() 发送的字符串直接传给 STARS 解析器。任何拼写错误、格式错误或引号不匹配都会导致模拟崩溃。建议先在独立的 .dat 文件中手动验证命令。

避坑 2：时间判断用精确值

cur_time == 75.0 这种判断在 STARS 中通常是安全的（因为握手时间由模拟器精确控制）。但如果涉及计算（如 cur_time + 0.1 == 75.1），建议使用 abs(cur_time - 75.0) < 1e-6 的判断方式，避免浮点数精度问题。

避坑 3：shut_in() 不可逆

一旦调用 shut_in()，该井在当前模拟中将被永久关闭（除非后续用 send() 重新激活）。务必在调用前添加确认逻辑，避免误操作。建议先用 print() 打印确认信息，观察几个时间步后再实际调用。

7.4 推荐的开发流程

基于以上经验，我们推荐以下六步开发流程：

0. 完整阅读 .dat 文件，明确控制目标和约束条件
1. 搭建只读监控脚本，确认能正确读取压力、产量等数据
2. 添加时间判断框架，在每个时间点打印确认信息

3. 逐步添加写操作：先调步长，再 send()，最后 shut_in()

4. 用粗网格、短时间的测试模型验证全部逻辑

5. 将控制策略抽象为函数，接入实际工程判断条件

7.5 展望未来

PyControl 的出现标志着油藏模拟从"离线计算工具"向"实时决策平台"的演进。结合当前 AI 技术的发展，我们可以预见以下几个方向：

方向	技术路径	预期效果
数字孪生	PyControl + 实时数据	模拟与实际状态实时同步
强化学习	PyControl + RL Agent	AI 自动学习最优注采策略
多模型集成	同时控制多个 STARS	区块级联合优化
云端部署	PyControl + 云计算	随时随地远程调控
不确定性量化	PyControl + 贝叶斯推断	评估地质不确定性

> *PyControl 不仅仅是一个接口，它是油藏工程与人工智能之间的桥梁。掌握它，你就掌握了将油藏模拟从"事后分析"升级为"实时决策"的钥匙。未来已来，让我们共同探索油藏数模的智能化之路。*

感谢阅读

PyControl 完全指南

油藏数模智能化 | 原创技术分享