Slurm 回填

> 读完你能做到：看懂自己的作业为什么在排队、预计什么时候开始、以及怎么改参数让它提前跑起来。

0. 五分钟速览

如果你只有五分钟，记住这四条，其中第一条最关键：

1. 不写 `-t`（时间限制）的作业，几乎不会被提前调度 —— 这是最常见的原因。
2. 查状态一律用 `sq`：`sq -u` 看自己的作业，`sq -p 分区名` 看分区资源，`sq -wp 分区名` 看每台节点的空闲核。
3. 核数越少、节点越少、时间越短，越容易被回填提前启动。
4. `预计开始` 是估算值不是承诺，每次都重新查。

1. 为什么你的作业在排队

提交作业后，`squeue` 里会看到 `PD`（PENDING，待调度）。很多人第一反应是"集群没资源"。

实际上有三种完全不同的情况：

情况	表现	能不能解决
A. 真的没资源	分区所有节点都满了	等，或者换分区
B. 有资源，但被队头堵住	有节点空着，你的作业就是不启动	能，改参数即可
C. 有资源，你也没被堵，但调度器还没算到你	刚提交，几秒到几十秒的延迟	等一下就好

大多数人是 B —— 而 B 正是这份手册要解决的问题。

Slurm

的主调度按优先级排队：从最高优先级的作业开始，能跑就启动；一旦遇到第一个跑不了的作业就停下来，后

面的全部跟着等。

举个例子， `7702ib` 分区（每台 128 核）：

```
待调度：  [大作业A：需要 4 个整节点 512 核]  ← 队头，现在凑不齐，跑不了
          [你的作业：4 核，10 分钟]
          [别人的作业：2 核]

主调度：A 跑不了 → 停在这里 → 你的 4 核作业明明有地方放，却一直在等
```

这就是队头阻塞：你被前面那个大作业挡住了。回填就是为了解决这个问题。

---

2. 回填是什么

回填（ Backfill）：在不推迟队头作业开始时间的前提下，把后面的小作业提前塞进当前的空闲资源。

还是上面那个例子：

```
队头 A 需要 4 个整节点，现在只能凑齐 3 个 → A 要等到明天早上
但分区里有 5 台节点还有零散空闲核

回填：既然 A 要到明天早上，那今晚这些零散的空闲核空着也是空着
      → 把你的 4 核作业塞进去，现在就跑
      → 只要不影响 A 明天早上准时开始，就没问题
```

回填是填空隙，不是插队 —— 它永远不会让别人的作业晚开始。这是它唯一的原则。

所以有一个关键结论：

你的作业能不能被回填，取决于 "
你现在要的这些资源，是不是正好有一个现成的空隙能放下 "。

资源要得越少、时间越短，空隙就越好找。后面所有技巧都是这一条的推演。

3. 用 sq 看集群状态

集群上有一个速查脚本 sq（在 /usr/bin/sq），比 sinfo / squeue 好用得多。

sq -h # 看完整用法

它有三个视图，本节逐个讲。

3.1 看分区资源： sq -p 分区名

sq -p 7702ib

输出：

================= 分区资源 分区=7702ib =================
分区 状态 核(已用/空闲/其他/总) 节点
7702ib mix 672/352/0/1024 a[001,003-006,008,013-014]

怎么读：

列 含义
分区 分区名（提交时 --partition 的值）
状态 alloc 全满 / mix 部分空闲 / idle 全空闲
核(已用/空闲/其他/总) 672 已在用， 352 空闲， 0 其他（保留 /故障）， 1024 总数
节点 这个分区有哪些节点在线

这个视图告诉你 "这个分区整体还剩多少核 "。

上例：8 台 × 128 核 = 1024 核，已经用了 672，空闲 352 核。

> “只看分区是不够的。352 核空闲不代表你能要到 352 核——
>
> 因为它分散在不同节点上，而一个作业可能只能在一个节点内运行（除非你写多节点）。
>
> 所以还要看节点视图。

3.2 看每台节点的资源：`sq -wp 分区名`

```
sq -wp 7702ib
```

输出：

```
================= 节点资源 分区=7702ib 仅正常节点 =================
节点 分区     状态 核(已用/空闲/其他/总)  负载    内存(MB)  空闲内存(MB)  原因
a001  7702ib  mix   96/32/0/128          95.20   257000    180000        none
a003  7702ib  alloc 128/0/0/128          128.10  257000    2100          none
a004  7702ib  mix   32/96/0/128          31.80   257000    240000        none
a005  7702ib  alloc 128/0/0/128          126.50  257000    3500          none
a006  7702ib  alloc 128/0/0/128          127.90  257000    2800          none
a008  7702ib  mix   64/64/0/128          63.40   257000    200000        none
a013  7702ib  mix   8/120/0/128          7.90    257000    490000        none
a014  7702ib  mix   88/40/0/128          87.30   514000    420000        none
```

怎么读：

列	含义
`核(已用/空闲/其他/总)`	这台节点还剩多少核
`负载`	这台节点当前的 CPU 负载（核数级别）
`内存(MB)`	物理内存总量
`空闲内存(MB)`	现在还剩多少内存可申请

关键：看 "空闲" 那一列（第 2 个数字）。

上例里：

节点	空闲核	说明
`a003` `a005` `a006`	0	全满，别想了
`a001`	32	还能放 32 核的作业
`a014`	40	还能放 40 核
`a008`	64	还能放 64 核
`a004`	96	还能放 96 核
`a013`	120	还能放 120 核 —— 最大的空隙在这里

> 这就是 "空隙" 的样子。你要跑一个单节点 64 核的作业？`a004`（96 空闲）和 `a013`（120 空闲）都能放下。你要跑 128 核？没有一台完全空闲，回填放不进去，只能排队。

注意 `a014` 的内存是 514000 MB（512 GB），其他是 257000 MB（256 GB）。`7702ib` 里 `a011`～`a014` 是大内存节点，需要用 `-R largemem` 申请。

3.3 看自己的作业：`sq -u` ← 最重要

```
sq -u
```

会输出三段：运行中的、待调度的、以及一段回填提示。

第一段：运行中（RUNNING）

```
================= 运行中作业 用户=user1 =================
JOBID    USER   PARTITION  NAME       核数 已运行      剩余          预计结束             节点
1234001  user1  7702ib     relax-48   48   3小时12分   8小时48分   2026-11-10 02:30   a001
```

列	含义
核数	这个作业占了多少核
已运行	已经跑了多久
剩余	距离时间限制还有多久（说明你的 `-t` 给得够不够）
预计结束	精确的结束时刻
节点	跑在哪台

> 如果 "剩余 "一直是 `1天23小时`这种大数，说明你的 `-t` 给得太宽松了 ——
>
> 这会降低你自己和别人被回填的机会（下一节解释）。

第二段：待调度（ PENDING） ← 核心

```
================= 待调度作业 用户=user1 =================
JOBID    USER   PARTITION  NAME       核数 时限          预计开始              还需等待     原因
1234002  user1  7702ib     scf-64     64   2-00:00:00  2026-11-10 08:00   13小时20分  Resources
1234003  user1  7702ib     test-4c    4    00:10:00    N/A                N/A          Resources
```

这两行是整份手册最重要的一张表。逐个字段讲：

列	含义	怎么看
`核数`	你要多少核	能不能被回填，主要看这个
`时限`	你写的 `-t`	`UNLIMITED` 或空 = 大问题
`预计开始`	调度器估算的开始时刻	是估算，会变
`还需等待`	距离现在还有多久	`N/A` 或 `即将` = 好消息
`原因`	为什么在等	`Resources` 最常见

`还需等待`的三种值，含义完全不同：

显示	含义	你该做什么
`即将` 或 `N/A`	回填已经在起作用，下一个调度周期就会启动（通常几秒到几十秒）	什么都不用做，等着
`30分` / `2小时15分`	有空隙，但要么不够大，要么时间对不上	减核数 / 减时限
`1天3小时` / `2天`	你的作业太大，回填塞不进去，只能等队头作业跑完	必须改参数，见第 6 节

上例怎么读：

- `1234002`（ 64 核， 2 天时限） → 等 13 小时 20 分。太大，回填放不进去。
- `1234003`（ 4 核， 10 分钟） → `N/A`。回填马上会给它找位置。

第三段：回填提示（每次都会打印）

```
------------------------------------------------------------
```

回填提示：

* "预计开始"是调度器按当前待调度作业算出的时间，可能随作业结束而变化；
* 核数申请越少、`-t` 设得越短，越容易被回填，从而早于"预计开始"启动；
* 作业实际结束早于"预计结束"时，资源会提前释放。

```
------------------------------------------------------------
```

这三句话就是全部要点。

3.4 看所有人的作业：`sq -ap 分区名`

想知道谁在挡着你，用这个：

```
sq -ap 7702ib
```

```
================= 待调度作业 分区=7702ib  用户=全部 ==================
JOBID    USER   PARTITION  NAME      核数 时限         预计开始              还需等待     原因
1233100  user2  7702ib     big-512   512   3-00:00:00  2026-11-10 06:00   11小时20分  Resources
1233150  user3  7702ib     md-128    128   1-00:00:00  2026-11-10 06:00   11小时20分  Resources
1234002  user1  7702ib     scf-64    64    2-00:00:00  2026-11-10 08:00   13小时20分  Resources
1234003  user1  7702ib     test-4c   4     00:10:00    N/A                N/A         Resources
```

这张表能看出队头是谁：

- `1233100`（`user2`，512 核 = 4 个整节点）是队头，要等到明天 06:00
- 后面两个（`user3` 的 128 核、`user1` 的 64 核）都被它挡着，`还需等待` 都是 11 小时以上
- 但 `user1` 的 4 核作业 `N/A` —— 它被回填了，不用等

这就是回填的意义：队头在等 11 小时，但小作业照样能跑。

---

4. 读懂 "预计开始" 和 "还需等待"

4.1 "预计开始" 是估算，不是承诺

`sq` 里的 `%S` 字段来自 `squeue --start`，Slurm 是这么算的：

1. 假设所有正在运行的作业都跑满 -t 的时间（最坏情况）

2. 看队头作业什么时候能凑齐资源

3. 依次往后推

所以它会在这些情况下变化：

发生了什么 预计开始怎么变
有作业提前结束（跑完了而不是超时） 变早
有更高优先级的作业进来 变晚
有作业被取消 变早
你自己改了核数 /时限重新提交 重算

“

不要拿 "预计开始 "去安排日程。它是一个 "当前状态下最乐观的估计 "。

4.2 "还需等待 "是三个数字里最该看的

还需等待 = 预计开始 - 现在

但它有个特殊逻辑：

如果调度器算出来的预计开始就在当下或已经过去 → 显示 即将

如果预计开始是 N/A（算不出来） → 显示 N/A

即将 和 N/A 都是好消息 ——

说明你的作业已经在回填的候选名单里，下一个调度周期（最多几十秒）就会启动。

如果显示的是 "13小时 20分 "这种，那就是真的在等。

5. 四个参数决定你能不能回填

回填能用的信息只有你 `#SBATCH` 里写的那些。这四个参数按影响从大到小排：`-t`

决定生死，核数与节点数决定难度，内存和 `--exclusive` 只是加减分。

5.1 `-t` / `--time`：最重要，没有之一

```
#SBATCH --time=02:00:00       # 2 小时
```

为什么重要：

回填要算 "这个作业会占用资源多久"。如果你写 `UNLIMITED`，回填不知道它会占多久，只能按 "永远" "处理" ——

那就永远找不到空隙。

结果：不写 `-t` 的作业，基本不会被回填。

怎么定 `-t`：

你心里想的	该怎么写
"我不知道要跑多久"	按经验给一个 1.5~2 倍的估计，不要写 `UNLIMITED`
"要跑 3 小时左右"	`--time=04:00:00`
"迭代收敛就跑完"	给一个上限，比如 `--time=12:00:00`，超时了 Slurm 会终止，你可以重提
"真的可能要跑一周"	`--time=7-00:00:00`，但要意识到这个作业基本不会被回填

看自己的作业时限给得合不合适：

```
sq -u
# 看"剩余"列，如果还剩 1天23小时 而作业其实只要 3 小时，就是浪费
```

5.2 `-n` / `--ntasks` 和 `-N` / `--nodes`：核数越少越好

```
#SBATCH --nodes=1             # 单节点
#SBATCH --ntasks=32           # 32 个任务
#SBATCH --cpus-per-task=1     # 每个任务 1 核
```

回填找空隙，是按 "节点" 为单位找的：

- 单节点 32 核 → 只要有一台节点空闲 32 核就行 → 容易
- 2 节点各 128 核 → 需要两台同时完全空闲 → 难得多
- 8 节点 512 核 → 需要八台同时空闲 → 很难

所以：能用 1 个节点就别要 2 个。核数越少，能放下的空隙越多。

`7702ib` 每台 128 核，常见的空闲档位：

你申请	需要什么条件	难度
≤ 8 核	任何一台有 8 核空闲的节点	□ □□□□
≤ 32 核	一台空闲 32 核	□ □□□
≤ 64 核	一台空闲 64 核	□□ 容易
≤ 96 核	一台空闲 96 核	□□□ 看运气
≤ 120 核	一台空闲 120 核	□□□□ 要看有没有这种节点
= 128 核（整节点）	一台完全空的节点	□□□□□ 难
> 128 核（多节点）	多个节点同时空	□□□□□□ 很难

5.3 `--mem`：写清楚，别让 Slurm 猜

```
#SBATCH --mem=32G              # 单节点 32 GB
#SBATCH --mem-per-cpu=4G       # 每核 4 GB
```

不写 `--mem` 时，Slurm 按分区的 `DefMemPerCPU` 算总内存。`7702ib` 的 `DefMemPerCPU` 是 1900 MB，写 `-n 128` 就默认要 128 × 1900 MB ≈ 243 GB，接近 256 GB 节点的上限。

而 `a011`～`a014` 是 512 GB，默认只用了不到一半 —— 需要大内存就显式写 `--mem`。

所以：

- 单节点作业：写 `--mem`，明确总量
- 大内存需求：`a011`～`a014` 是 512 GB，要用 `-R largemem` 申请

给多了会占用空隙，也会降低回填机会。按实际需要写。

5.4 `--exclusive`：不要随便用

```
#SBATCH --exclusive                # ← 独占整个节点，慎用
```

`--exclusive` 表示"这个节点上除了我不许跑别人"。它会让你以为的 32 核变成 128 核的需求，直接把小作业变成大作业。

除非你确实需要独占（比如测试 GPU 或需要完整内存带宽），否则不要写。

---

6. 从"查到的数字"到"该设多少参数"← 核心章节

前面讲了怎么看、哪些参数重要。这一节讲具体怎么用查到的数据做决定。

6.1 三步判断法

第一步：`sq -wp` 分区名 → 看最大单节点空闲核数（记为 MAXFREE）

第二步：`sq -u` → 看自己作业的"还需等待"

第三步：对照下面的表做决定

6.2 对照表（以 7702ib 的示范数据为例）

前提：`sq -wp 7702ib` 显示的最大单节点空闲核是 `a013` 的 120 核。

你的作业	还需等待显示	结论	怎么做
4 核 / 10 分钟	`N/A` 或即将	已经在回填名单里	等 30 秒
4 核 / 10 分钟	`2小时`	不正常，检查是不是没写 `-t`	补 `-t`
32 核 / 1 小时	`N/A` 或即将	正常，回填生效	等
64 核 / 2 小时	`N/A` 或短时间	正常，`a004`/`a008`/`a013` 都能放	等
64 核 / 2 小时	`13小时`	空隙不够大了（可能被人占了）	降到 32 核，或换分区
96 核 / 2 小时	`N/A` 或短时间	运气好，`a013` 有 120 核	等
96 核 / 2 小时	`几个小时`	只有 `a004`(96) 刚好能放，被占了	降到 64 核
120 核 / 2 小时	`几个小时`	只有 `a013` 能放，别人一占就没戏	降到 96 或 64 核

你的作业	还需等待显示	结论	怎么做
128 核（整节点）	几个小时	没有完全空闲的节点	拆成 2 个 64 核作业，或换分区
多节点（如 2×128）	基本都很长	需要多个节点同时空	拆成多个单节点作业

6.3 时限 `-t` 怎么定

核心原则：写"够用就行"的值，不要写"保险"的值。

```
# □ □□□ —— 降低自己被回填的机会，也占着资源不放
#SBATCH --time=7-00:00:00       # 实际上 3 小时就跑完了

# □ □□ —— 留 1.5~2 倍余量
#SBATCH --time=04:00:00         # 实际 2~3 小时
```

为什么"给太多"会害自己：

回填算空隙时，要保证你的作业不会把空隙占太久。你写 7 天，回填就认为"这个作业要占 7 天"，任何小于 7 天的空隙都不能用。

写完怎么验证：

```
sq -u
# 运行中的作业看"剩余"列
# 如果"剩余"和"已运行"比是 5:1 以上，说明 -t 给得太宽了
```

迭代类作业（SCF 收敛、分子动力学）的建议做法：

```
#SBATCH --time=24:00:00      # 给一天，跑不完说明有问题，重提

# 或者用 --signal 让作业超时前自己保存
#SBATCH --signal=B:USR1@300     # 超时前 300 秒发信号
```

在脚本里捕获信号、保存中间结果、然后主动退出：

```
trap 'echo "time up, saving..."; save_my_state.sh; exit 0' USR1
```

这样即使用满时限也不会白跑。

6.4 一个完整的决策流程

```
# ① 先看你最关心的分区还剩多少
sq -p 7702ib

# ② 看每台节点的空隙
sq -wp 7702ib
#    找一个"空闲核数 ≥ 你的作业核数"的节点，记下那个数字

# ③ 提交作业（假设你要 64 核）
sbatch -p 7702ib -N 1 -n 64 -t 04:00:00 --mem=64G myjob.sh

# ④ 立刻查
sq -u

# ⑤ 看"还需等待"
#    是 N/A / 即将  → 等 30 秒，然后 sq -u 再看
#    是几小时       → 执行⑥
#    （先确认你没写 UNLIMITED）

# ⑥ 改参数重来
scancel <你的jobid>
sbatch -p 7702ib -N 1 -n 32 -t 04:00:00 --mem=32G myjob.sh
#    核数减半 → 能放下的空隙翻倍
```

---

7. 动手验证：两个作业的对照实验

光看手册没用，自己跑一次就知道了。用两个 60 秒的小作业做对比：

实验 A：不规范写法

```
sbatch -p 7702ib -N 1 -n 8 --wrap="sleep 60"
```

注意：没写 `-t`。

实验 B：规范写法

```
sbatch -p 7702ib -N 1 -n 8 -t 00:05:00 --wrap="sleep 60"
```

立刻查两者的状态

```
sq -u
```

预期结果：

作业	时限显示	还需等待	说明
A	`UNLIMITED`	可能是 `N/A`，也可能排很久	取决于当时有没有大作业挡着
B	`00:05:00`	`N/A` 或 即将	几乎必然立即启动

如果当天正好没有大作业挡着，A

也会很快启动。想看到差别，最好在有排队的时候做这个实验（比如工作日白天）。

实验 C：大作业的对照

```
# 申请整节点 —— 观察它要等多久
sbatch -p 7702ib -N 1 -n 128 -t 00:10:00 --wrap="sleep 60"

# 申请 4 核 —— 观察它是不是立刻启动
sbatch -p 7702ib -N 1 -n 4 -t 00:10:00 --wrap="sleep 60"
```

```
sq -u
```

预期：4 核的立刻 RUNNING，128 核的要等（因为没有完全空闲的节点）。

这个对比做完，你就彻底理解回填了。

---

8. 完整流程示例：一个真实的参数调整过程

场景：`user1` 要跑一个 ABACUS 的 SCF 计算，24 核，估计 3 小时。

第一次提交（不规范）

```
sbatch -p 7702ib -N 1 -n 24 -J scf-24 abacus.sh
```

```
sq -u
```

```
================= 待调度作业 用户=user1 =================
JOBID    USER    PARTITION  NAME      核数 时限         预计开始              还需等待     原因
1234010  user1   7702ib     scf-24    24    UNLIMITED   2026-11-10 09:30    16小时45分   Resources
```

问题诊断:

- 时限 `UNLIMITED` → 回填用不上
- 等 16 小时 45 分 → 明显不合理（分区有空闲核）

第二次提交（补了 `-t`）

```
scancel 1234010
sbatch -p 7702ib -N 1 -n 24 -t 06:00:00 -J scf-24 abacus.sh
```

```
sq -u
```

```
================= 待调度作业 用户=user1 =================
JOBID    USER    PARTITION  NAME      核数 时限         预计开始              还需等待     原因
1234011  user1   7702ib     scf-24    24    06:00:00    N/A                   N/A          Resources
```

`N/A` = 回填已经接手，下一个周期就启动。

```
# 30 秒后再看
sq -u
```

```
================= 运行中作业 用户=user1 =================
JOBID    USER    PARTITION  NAME      核数 已运行 剩余       预计结束              节点
1234011  user1   7702ib     scf-24    24    0分      6小时0分   2026-11-09 23:45    a013
```

从 16 小时 45 分变成 30 秒，只因为加了一个 `-t`。

如果补了 `-t` 还是要等

```
sq -wp 7702ib
# 看最大空闲是多少
```

假设最大空闲只有 16 核：

```
# 把 24 核降到 16 核（多开几个小作业并行）
scancel 1234011
sbatch -p 7702ib -N 1 -n 16 -t 06:00:00 --wrap="...前 16 核的部分..."
sbatch -p 7702ib -N 1 -n 8  -t 06:00:00 --wrap="...剩下的 8 核..."
```

两个小作业比一个大作业更容易被回填（虽然 MPI 效率会略降，但总比干等 16 小时强）。

---

9. 常见问题

Q1：我写了 `-t`，为什么还要等

三个原因，逐个排查：

```
# ① 队头作业太大了，你需要的核数连空隙都塞不下
sq -ap 7702ib | head -5

# ② 你的核数超过了任何单节点的空闲核数
sq -wp 7702ib

# ③ 你的作业是多节点（-N 2 以上）
scontrol show job <jobid> | grep NumNodes
```

Q2：`预计开始`一直往后推，是不是集群坏了

不是。说明有更高优先级的作业在不断进来，或者你前面那个队头作业也在等更前面的人。

```
sq -ap 分区名 | head -10
# 看排在你前面的都是谁、要多少核
```

Q3：我的作业明明只需要 5 分钟，为什么要等几小时

九成是没写 `-t`，或者写了 `-t=7-00:00:00`。

```
squeue -j <jobid> -o "%.10i %.9P %.20j %.10l %.10T"
# %l 是 TimeLimit
```

如果显示 `UNLIMITED` 或很大的值，改成实际需要的。

Q4: `-t` 给少了，作业被杀了怎么办

```
# 查看是否超时
sacct -j <jobid> --format=JobID,State,ExitCode,Elapsed,Timelimit
# State=TIMEOUT 就是超时
```

然后:

1. 加大 `-t` 重提
2. 或者改成断点续算（把中间结果存盘，下次接着跑）

建议做法：在脚本里定期保存状态，这样超时也不会白跑。

```
# 每隔 N 步保存一次
if (( step % 10 == 0 )); then
    cp -r OUT.ABACUS OUT.ABACUS.save
fi
```

Q5: 为什么别人 4 核的作业插在我前面跑了

因为那是回填，不是插队。

回填只占用 "本来会空着 "的资源，不会让你的作业晚一秒开始。这是 Slurm 的设计保证。

如果有人用 `--qos=high`（高优先级 QoS）抢先，那才是插队 —— 但那种情况下你的作业会显示 `PREEMPTED`:

```
sacct -j <jobid> --format=JobID,State,DerivedExitCode
# State=PREEMPTED 说明被抢占
```

Q6: 怎么知道我的作业跑在哪台机器上

```
sq -u
# 运行中作业的"节点"列

# 或者
```

```
squeue -j <jobid> -o "%.10i %R"
```

Q7: `sq` 和 `squeue` 有什么区别

`sq` 是把 `sinfo` / `squeue` 的常用查询包装过的脚本，输出更适合人看：

需求	用 `sq`	用原生
看分区空闲	`sq -p 分区`	`sinfo -p 分区 -o "%P %t %C %N"`
看节点空闲	`sq -wp 分区`	`sinfo -N -p 分区 -o "%N %t %C %O"`
看自己的作业	`sq -u`	`squeue -u $USER -o "%.10i %.9P %.20j %.10T %.20S"`
看所有人	`sq -ap 分区`	`squeue -p 分区 -o "..."`

`sq` 帮你算了 "还需等待" 和 "已运行/剩余"，原生命令要自己算。

Q8: 能不能预先知道作业会被回填

不能 "预先"，但可以 "实时看"。

标准做法是 "提交 → `sq -u` → 30 秒后再看"。如果转成 RUNNING 就是被回填了。

Q9: 大内存作业怎么申请

`7702ib` 里只有 `a011`～`a014` 是 512 GB（其他 256 GB）：

```
#SBATCH --partition=7702ib
#SBATCH -R largemem          # 指定大内存节点
```

先用 `sq -wp 7702ib` 确认 `a011`～`a014` 有空闲，再提交。

Q10: 作业一直 `N/A` 但就是不启动

先看节点状态：

```
sq -w          # 默认只显示正常节点
sq -w a013     # 点名某台时照实显示（可能 drain 了）
```

如果节点 `drain`（故障/维护），`sq -wp` 的默认视图不显示它，但作业也调不上去。

```
sinfo -R      # 看所有 drain 的节点和原因
```

10. 速查卡

命令

```
sq                      # 分区总览
sq -p 7702ib            # 指定分区
sq -wp 7702ib           # 该分区每台节点的空闲情况
sq -u                   # 我的作业（运行中 + 待调度）
sq -ap 7702ib           # 该分区所有人的待调度作业（看谁在挡着）
sq -w a013              # 点名单台节点（含异常状态）
sq -h                   # 完整用法与说明
```

作业头（推荐模板）

```
#!/bin/bash
#SBATCH --partition=7702ib
#SBATCH --job-name=myjob
#SBATCH --nodes=1                   # 能用 1 个节点就别要 2 个
#SBATCH --ntasks=32                 # 核数按实际需要
#SBATCH --cpus-per-task=1
#SBATCH --mem=64G                   # 明确写内存
#SBATCH --time=04:00:00             # ★ 必须写，这是能否被回填的关键
#SBATCH --output=out.%j
#SBATCH --error=err.%j
```

三个数字

看到	意思	动作
`还需等待 = N/A` 或 `即将`	回填已接手	等 30 秒
`还需等待 = 几小时`	空隙不够	减核数 / 减时限 / 换分区
`~` 时限 `= UNLIMITED`	回填失效	立刻补 `-t` 重提

> “ 核数少、节点少、时间短、写了 `-t` —— 满足这四条，作业几乎必然立即启动。

附录

本手册的示范数据说明

文中所有 7702ib 的输出示例（节点名、核数、作业 ID、 user1 / user2 / user3）都是按 sq 的真实输出格式构造的教学示例，不是某一时刻的实际数据。

你自己的集群状态请以实时运行 sq 的结果为准。

分区硬件参数（ 128 核/台、 256 GB 或 512 GB 内存）来自 计算资源 。

Revision #3
Created 2026-11-09 22:49:02 CST by Jiang Meng
Updated 2026-11-09 23:15:46 CST by Jiang Meng