流程：系統每5分鐘觸發一次 → 執行緒開始執行 → 檢查執行結果 → 5 分鐘後，系統重新取出新的執行緒

【要解決的問題】

1. 避免尖峰值造成的異常 → 平衡 CPU 負載，避免一次處理過多執行緒。
2. 避免執行緒重複執行 → 確保未完成的執行緒能夠繼續執行，而不是重新取出。
3. 確保每個執行緒最短時間內執行 → 透過最佳化排程機制，縮短等待時間。

【分流機制】

1. 基於Little's Law的執行緒數量計算
   * 系統透過監控任務到達率和服務時間，使用Little's Law公式（L = λW）計算理論上最佳的執行緒數量
   * 計算公式：最佳執行緒數 = 到達率（平均每秒要接收任務數）× 服務時間（20秒）
2. CPU負載與佇列狀態的動態調整
   * 系統會根據當前CPU使用率和待處理任務數量動態調整執行緒池大小
   * 當CPU負載高於85%或待處理任務少於3個時，執行緒數會減少，此時減少執行緒數到理論值的60%（但不少於3個）
   * 當CPU負載低於40%且待處理任務多於10個時，執行緒數會增加，此時增加執行緒數到理論值的140%（但不超過20個）
3. 任務重排優先機制
   * 系統會識別等待時間過長（超過5分鐘）的任務，提高其優先級並重新排入佇列前端
   * 這確保長時間等待的任務不會被無限期延遲
   * （若執行的時間超過20秒，會發出警告，但不會重新安排執行或是中斷，會繼續執行任務直到完成）
4. 負載平衡的任務產生策略
   * 任務產生不是固定的，而是根據當前系統負載動態調整
   * 在高負載情況下減少產生的任務數量（1-5個）
   * 在低負載情況下增加產生的任務數量（15-20個）
   * 同時考慮每小時總量目標（200個任務），確保均勻分配
5. 平滑化處理
   * 避免執行緒數量的劇烈變化，使用加權平均法計算最終執行緒數
   * current\_optimal\_threads=0.7\*current\_optimal\_threads+0.3\*optimal\_threads

考慮了理論模型（Little's Law），還考慮了實際系統狀況（CPU使用率、佇列長度），並通過任務重排和平滑化處理來避免系統波動，達到資源有效利用和任務均衡處理的目標。

|  |  |  |
| --- | --- | --- |
|  | 優化前 | 優化後 |
| 執行緒池 | 固定大小（8個執行緒） | 使用Little's Law動態計算最佳執行緒數量，並根據CPU負載進行調整 |
| 負載感知 | 不考慮系統當前負載，只考慮隨機數和小時限制 | 高負載時減少，低負載時增加 |
| 重排機制 | 將未完成任務重新放入隊列尾部 | 優先級重排，長時間等待的任務會被放到隊列前端 |
| 異常監控 | 無法識別執行時間過長的任務 | 可識別並報告執行時間異常的任務 |