01 使用者

秤重校正精靈

📅 最後更新：2026-07-28 | 🛠️ 對應韌體：v5.9.130+ | 📌 負責人：KC

本手冊說明 TDA08B 秤重模組的**引導式校正流程**（4 步驟 + 燈號提示），以及背後的 MQTT 命令、設定建構方式。校正以 MQTT-Chain 工作流引擎 實作，**全部設定、無專屬韌體**。

01

1. 為什麼是「引導精靈」而不是單一命令

TDA08B 的有砝碼校正（依原廠手冊 §5.4.1）是**缺一不可的 4 步驟**：

步驟	暫存器寫入	使用者動作	燈號提示
1 選通道	reg107 = 1	（自動）	—
2 標零點（tare）	reg113 = 1	**放空籃**上秤	第一顆燈亮（進行中）
3 寫砝碼值	reg108 = 砝碼值	**籃內放砝碼**	第一顆燈持續亮
4 確認增益	reg113 = 2	（自動）	第二顆燈亮（完成）

⚠️ **燈號沒有固定顏色。**韌體只控制「第幾顆燈亮/滅」，顏色完全取決於現場把哪一路 DO 接到哪一顆燈泡。舊版本文件在這裡標了藍/黃/綠三種顏色，那是**錯的** —— 實際只有兩顆燈的 亮滅，而且藍色容易被誤認成 Opta 板載的藍色 LED（完全不同的東西，見 §7）。接線與設定方式見 **§7 號誌燈搭配**。

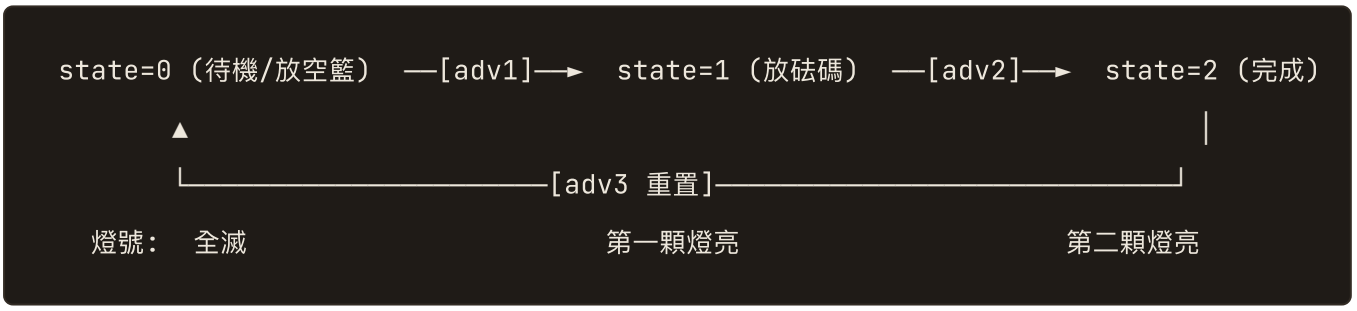
> **籃子（basket-as-tare）流程**：放空籃標零點（扣掉籃重）→ 籃內放砝碼標增益 → 之後秤籃裡的東西就是淨重。**砝碼值換算**：TDA08B A1 dot=2（小數兩位），5kg → `reg108 = 500`（顯示值 ×100）。

每一步需要使用者在環（放空籃/放砝碼）+ 燈號回饋，否則操作員不知道現在該做什麼。這是**有狀態的精靈**，不是 fire-and-forget 命令。

02

2. 狀態機設計

以 MQTT topic `…/cmd/signal/0` 當「目前步驟狀態」（state），由 gateway 發佈；操作員以「推進按鍵」逐步前進。



推進按鍵	MQTT topic	條件（AND）	動作
adv1 標零	`…/cmd/signal/1`	state==0 ∧ adv1≥0.5	reg107=1, reg113=1, state→1, 第一顆燈亮
adv2 標增益	`…/cmd/signal/2`	state==1 ∧ adv2≥0.5	reg107=1, reg108=500, reg113=2, state→2, 第二顆燈亮
adv3 重置	`…/cmd/signal/3`	state==2 ∧ adv3≥0.5	state→0, 全滅

順序強制：每步用 2-source AND（`state==N` ∧ `advN`），只有在前一步 state 正確時才會觸發 → 缺一不可。

03

3. 操作流程（現場操作員）

> 前提：磅秤已接 TDA08B（slave 3, bus0, 9600/8N1），MQTT 已連線。推進按鍵以**瞬時**方式送（發 `1` 後立即發 `0`，模擬實體按鈕）。

用 MQTTX / mosquitto_pub 操作

```bash
PRE="mes/gateway/<UID>"   # UID 見 /api/poll 或設備標籤

# 步驟 1：放空籃在秤上 → 推進標零
mosquitto_pub -h <broker> -p 1883 -u smms -P <pw> -t "$PRE/cmd/signal/1" -m 1
mosquitto_pub -h <broker> -p 1883 -u smms -P <pw> -t "$PRE/cmd/signal/1" -m 0
#  → 第一顆燈亮，state=1（觀察 $PRE/cmd/signal/0 = 1）

# 步驟 2：籃內放 5kg 砝碼 → 推進標增益
mosquitto_pub ... -t "$PRE/cmd/signal/2" -m 1
mosquitto_pub ... -t "$PRE/cmd/signal/2" -m 0
#  → 第二顆燈亮，state=2，weight 應校到 5.00

# 完成。要重新校正：
mosquitto_pub ... -t "$PRE/cmd/signal/3" -m 1
mosquitto_pub ... -t "$PRE/cmd/signal/3" -m 0
#  → 全滅，state=0，回到待機
```

觀察校正結果

```bash
# 訂閱即時重量（若有設 Converter 遙測通道）
mosquitto_sub -h <broker> -p 1883 -u smms -P <pw> -t "$PRE/scale/weight" -v
# {"weight":5.00,"stable":1,...}  ← 校正成功後放 5kg 砝碼應顯示 5.00
```

> 七色燈可用三色燈或任意兩路 DO 模擬：實際只有「整顆亮/滅」，用第一顆＝進行中、第二顆＝完成。**哪一路 DO 對應哪顆燈由現場接線決定**，索引寫法與實測步驟見 §7.3。

04

4. 燈號對照

state	意義	第一顆燈	第二顆燈
0	待機 / 放空籃	滅	滅
1	放砝碼	**亮**	滅
2	完成	滅	**亮**

05

5. 設定建構（工程/部署）

整套 wizard 以 API 建構（compact JSON，序列化送）。完整 schema 見 spec-mqtt-chain-workflow.md。建構順序：

1. **寫暫存器**（`POST /api/config` modbusRegisters，連同既有讀暫存器一起送）：

- reg107（cal_ch）、reg108（cal_wt）、reg113（cal_trig），全部 `mode=1`（Write）、`functionCode=6`。

2. **TCP IO 狀態通道**（`POST /api/tcpio`，CONVERTER/OUTPUT）：
   - `wz-st1` payload `1`、`wz-st2` payload `2`、`wz-st0` payload `0`，topic 都指 `…/cmd/signal/0`。
3. **Signals**（`POST /api/signals`，2-source AND，trigger=CHANGE）：校正標零 / 標增益 / 重置。
4. **Actions**（`POST /api/actions`）：標零 / 標增益 / 重置，含 CH_MODBUS 寫 + CH_TCP 發 state + CH_DO 燈號。
5. **Rules**（`POST /api/rules`）：3 條 signal→action。

> ⚠️ 每個 POST 後 GET 驗證（特別是 signal 的 `sourceCount` 要等於 2）。JSON 必須 compact、Content-Length 用 byte 數。詳見規格文件「已知雷區」。

06

6. 驗證紀錄（2026-05-26, 72.77）

機制層已端到端驗證 PASS（磅秤暫拆，TDA08B 控制器在線）：

```
adv1 → rules/2/trigger 校正標零 → cmd/signal/0=1 → 第一顆燈 ON
adv2 → rules/3/trigger 校正標增益 → cmd/signal/0=2 → 第二顆燈 ON
adv3 → rules/4/trigger 校正重置 → cmd/signal/0=0 → 全滅
```

底層每環節確認：MQTT 輸入、2-source AND 順序強制、CHANGE 邊緣、rule fire、CH_MODBUS 寫命令、CH_DO 燈號切換、CH_TCP state loopback。

待驗：接回磅秤後跑完整校正，確認 reg108 增益生效、weight 校到 5.00（需實體 load cell）。

07

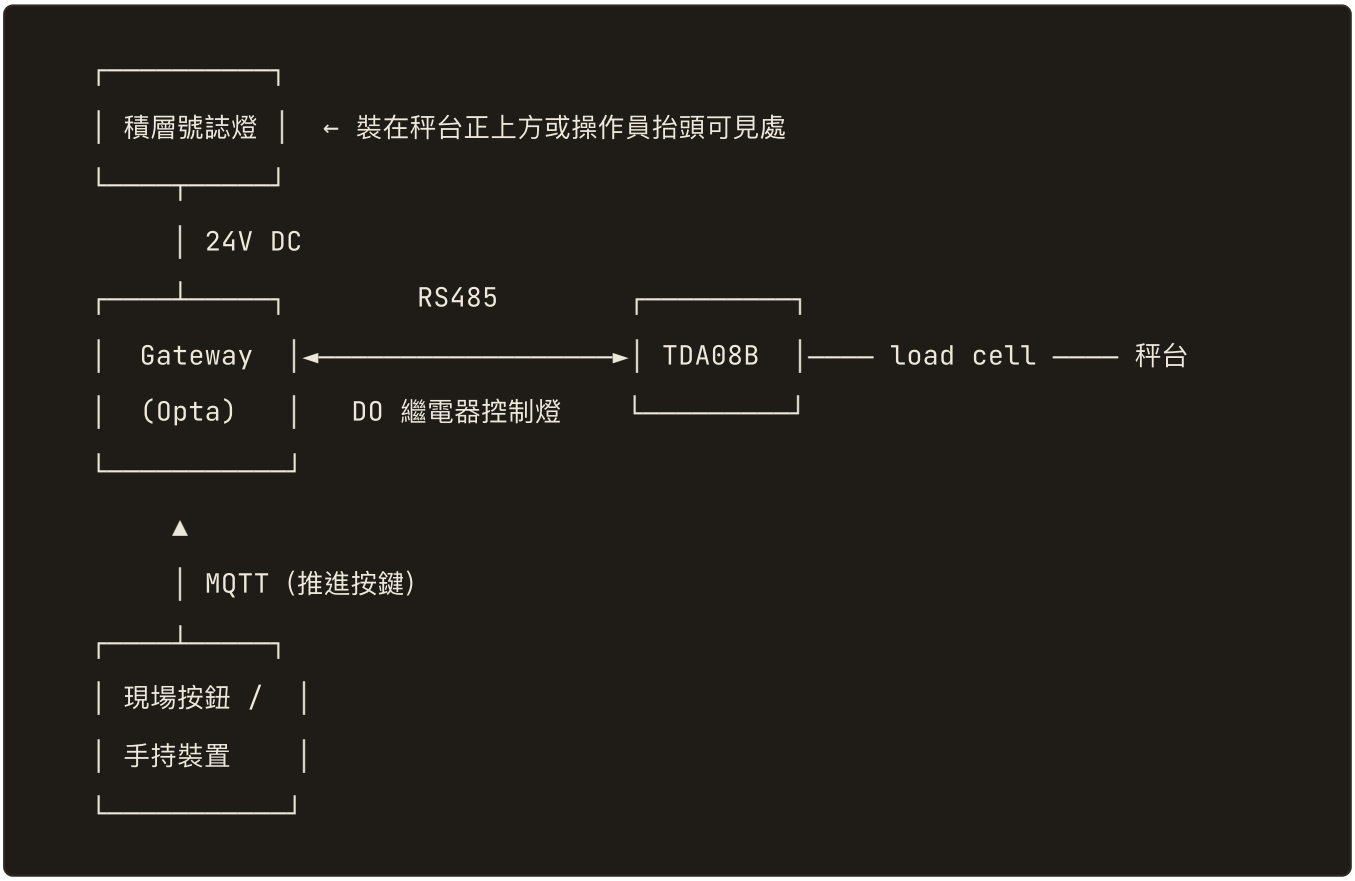
7. 號誌燈搭配（三色燈 / 積層燈）

> **磅秤和號誌燈是兩件各自獨立的設備**，彼此不知道對方存在，也不一定裝在一起。把它們串起來的是 Gateway 的 Rule Engine：校正精靈推進到某個狀態時，觸發一個 Action 去切換 DO 繼電器，繼電器再點亮外接的燈。**號誌燈不是磅秤的配件，也不是必要的** —— 沒有燈一樣能完成校正，只是操作員少了現場提示。

7.1 應用情境：為什麼要用燈，而不是看螢幕

校正是**雙手都被佔用**的工作：一手扶籃、一手放砝碼，人站在秤台前，眼睛盯著秤盤。這種情況下「回頭去看電腦螢幕確認現在第幾步」是不切實際的。

典型現場長這樣：



操作員視角的完整流程：

燈況	操作員知道要做什麼	做完按推進鍵
全滅	待機。把**空籃**放上秤台	按「推進」→ 標零
第一顆燈亮	零點已抓。在**籃內放砝碼**	按「推進」→ 標增益
第二顆燈亮	**校正完成**，可以開始秤料	（不用按）
全滅（從第二顆燈熄滅）	已重置，要重新校正	回到第一步

重點是**操作員完全不需要知道 reg107/reg113 是什麼**，也不需要看螢幕 —— 抬頭看燈就知道現在在哪一步、下一步該做什麼。這就是把它做成「引導精靈 + 燈號」而不是一道命令的原因。

常見場所：食品／五金／電子廠的分裝秤重站、產線旁的計重工位、需要定期複校的計量點。這些地方共通點是：環境吵（聽不到語音提示）、戴手套（不方便操作螢幕）、Gateway 通常裝在 配電箱裡（現場看不到）。

> 💡 **不裝燈也可以**：若操作員本來就在電腦前作業（例如實驗室環境），直接看 MQTT 的 `…/cmd/signal/0` 狀態值或網頁即可，本章可整章略過。

7.2 先釐清：這裡講的燈不是 Opta 板載 LED

	外接號誌燈（本章）	Opta 板載 LED
位置	機台上的積層燈 / 三色燈塔	Opta 本體面板
由誰控制	Rule Engine → DO 繼電器	韌體內部（心跳、開機狀態）
顏色	**由現場接線決定**	硬體固定
跟校正的關係	本章要設定的東西	**無關**

⚠️ 板載那顆藍燈（LEDB / PE_5）**只有 8320（WiFi 版）能亮**；8310 沒有 WiFi 模組，藍燈硬體 不存在，韌體改用紅+綠同亮的琥珀色代替。這跟校正燈號完全無關，但若文件用藍色圓點標示校正 步驟，很容易讓人誤以為看的是板載燈 —— 特此區分。

7.3 需要幾顆燈

精靈只有三種狀態，**兩顆燈就夠**：

state	意義	第一顆燈	第二顆燈
0	待機 / 該放空籃	滅	滅
1	該放砝碼	**亮**	滅
2	校正完成	滅	**亮**

⚠️ **韌體沒有顏色的概念**，它只控制「第幾顆燈亮/滅」。哪顆燈是什麼顏色，完全取決於現場 把哪一路 DO 接到哪顆燈泡。

用三色燈塔的話，建議這樣配：

燈	建議用途	理由
黃	第一顆（校正進行中）	「注意／進行中」是黃燈的通用語意
綠	第二顆（校正完成）	「完成／可作業」
紅	**不接精靈**，留給故障告警	紅燈被佔用會失去告警意義

七色積層燈（如 PATLITE NE-M1ATB-M）接法相同，只是可選顏色更多。一顆燈也可以由**多路 DO 一起驅動**（某些燈塔一個顏色需要兩路訊號），Action 裡多加一個 output 即可。

7.4 找出 DO 索引（最容易接錯的一步）

Action 的 `CH_DO` output 用一個 byte 編碼位置：

```
index = (expIdx << 4) | ch
  expIdx : 0 = 本機 Opta，1 = 第一台擴充模組，2 = 第二台…
  ch     : 該裝置上的通道，從 0 起算
```

⚠️ **`ch` 是 0-based，但面板和網頁上的標示是 1-based。**韌體內部 `DO_PINS[] = {D0,D1,D2,D3}`，`writeDO(ch)` 直接索引，所以：

面板標示	內部 ch	index（本機）
DO1	0	`0`
DO2	1	`1`
DO3	2	`2`
DO4	3	`3`

第一台擴充模組的 ch2、ch3 → index `18`、`19`（`(1<<4)|2`、`(1<<4)|3`）。

🔧 實測步驟（強烈建議，兩分鐘）

接線後**不要只照表填**，用實測確認對應關係：

1. 打開 Gateway 網頁 → **I/O 頁**。
2. 逐一手動切換 DO1 → DO2 → DO3 → DO4，每切一路就看現場**哪顆燈亮了**。
3. 記錄下來，例如：

面板切換	現場亮的燈	要填的 index
DO2	黃燈	`1`
DO3	綠燈	`2`

4. 用**實測記錄**去填 §7.5 的 Action，不要用猜的。

> 這一步能省掉現場「燈亮錯顆」來回改設定的時間。若接的是擴充模組，同樣在擴充模組頁逐路測。

7.5 精確設定：建立三個 Action

每個狀態一個 Action。**關鍵：該亮的設 ON、該滅的要明寫 OFF** —— 只設 ON 的話上一顆燈不會熄，會變成兩顆一起亮。

Action	對應 state	第一顆燈	第二顆燈
校正標零	1	`outputMode=0` (ON)	`outputMode=4` (OFF)
校正標增益	2	`outputMode=4` (OFF)	`outputMode=0` (ON)
校正重置	0	`outputMode=4` (OFF)	`outputMode=4` (OFF)

`outputMode` 對照：`0`=直接輸出(ON)、`1`=映射輸出、`2`=脈衝(配 `pulseMs`)、`4`=**強制 OFF**、`5`=toggle。

完整 API 範例

以 §7.4 實測結果「黃燈=index 1、綠燈=index 2」為例（`type: 2` 即 `CH_DO`）：

```bash
TOKEN=smmsadmin        # 設備 admin token
GW=http://192.168.10.55

# Action 0：校正標零 → 黃燈亮、綠燈滅
curl -X POST "$GW/api/actions" -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" -d \
'{"index":0,"enabled":true,"name":"校正標零燈號","outputs":[{"type":2,"index":1,"ou

# Action 1：校正標增益 → 黃燈滅、綠燈亮
curl -X POST "$GW/api/actions" -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" -d \
'{"index":1,"enabled":true,"name":"校正完成燈號","outputs":[{"type":2,"index":1,"ou

# Action 2：校正重置 → 兩顆全滅
curl -X POST "$GW/api/actions" -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" -d \
'{"index":2,"enabled":true,"name":"校正重置燈號","outputs":[{"type":2,"index":1,"ou
```

必讀注意事項：

- ⚠️ `"enabled":true` **一定要明寫**。不帶這個欄位會**靜默停用**，Action 建起來卻不動作，而且不會報錯 —— 這是最常見的卡關原因。
- `index` 是**陣列位置**（0 起算），不是 `actionId`。`actionId` 由韌體自動指派，建立後用 `GET /api/actions` 查。
- 規則（Rule）裡引用 Action 時用的欄位是單數 **`actionId`**，不是 `index`。
- JSON 必須 **compact**（不能有空格）—— 韌體用 `indexOf("\"key\":value")` 比對，有空格會靜默失敗。`Content-Length` 要用 **byte 數**（中文名稱是多位元組）。
- 每個 POST 後用 `GET /api/actions` 驗證 `outputCount` 是否等於預期（本例為 2）。

建完 Action 後，在 §5 的 Rules 步驟把三條規則分別指向這三個 `actionId` 即可。

7.6 加蜂鳴器（選配）

吵雜環境可加一顆蜂鳴器，在「完成」的 Action 裡多加一個 output。**用脈衝模式**，否則會一直叫：

```JSON
{"type":2,"index":3,"outputMode":2,"pulseMs":500}
```

（`index:3` = 面板 DO4，依 §7.4 實測結果調整。）

7.7 驗收

設定完成後，依序送三個推進命令，對照燈號：

送出	預期燈況
`cmd/signal/1`	第一顆燈**亮**、第二顆**滅**
`cmd/signal/2`	第一顆**滅**、第二顆**亮**
`cmd/signal/3`	兩顆**全滅**

三步都對，燈號就跟精靈綁好了。

排錯：

症狀	原因	處理
燈亮錯顆	DO index 填錯（多半是 0-based/1-based 搞混）	回 §7.4 重新實測
兩顆一起亮	該滅的沒明寫 `outputMode=4`	補上 OFF output
完全沒反應	Action 沒帶 `"enabled":true`	重送並確認
燈一直亮不熄	蜂鳴器/燈用了 `outputMode=0` 而非脈衝	改 `outputMode=2` + `pulseMs`

> 燈亮錯顆一律是**設定或接線**問題，不需要改韌體。

08

8. 相關文件

- 工作流引擎規格 / API：`../specs/spec-mqtt-chain-workflow.md`
- MQTTX 連線設定：`../guides/guide-web-login.md`、`mqttx-tls` SOP