

AbstractChargeMission 使用手册

目录

- [概述](#)
- [核心功能](#)
- [快速开始](#)
- [配置参数详解](#)
- [高级功能](#)
- [扩展点说明](#)
- [常见场景示例](#)
- [故障排查](#)

概述

`AbstractChargeMission` 是一个抽象的充电维护任务类，提供了完整的车辆充电管理功能，包括：

- 多种充电策略（必须充电、空闲充电、低电量充电）
- 任务优先级与抢占机制
- 补电与电量校准
- 充电异常检测与自动切换充电桩
- 待命点管理

任务优先级

生产任务 > 必须充电 > 低电量充电 > 空闲充电 > 回待命点

核心功能

1. 智能充电调度

1.1 必须充电（MustCharge）

当车辆电量低于 `mustChargeSoc` 阈值时，系统会强制车辆去充电。

触发条件：

- $SOC < mustChargeSoc$ （默认 20%）
- 延迟 5 秒后触发（防止电量波动误触发）

行为：

- 添加 `shouldCharge` 标签
- 优先级高于普通任务
- 记录日志：`小车:{车辆名称}, 必须充电, 当前电量:{SOC}`

1.2 空闲充电 (IdleCharge)

车辆空闲一段时间后，如果电量未滿，自动去充电。

触发条件：

- $SOC < idleChargeSoc$ (默认 90%)
- 空闲时间 $> idleChargeSeconds$ (默认 30 秒)
- 有空闲充电桩
- 待执行任务数 $\leq minAllowFreeCarToChargeTaskCnt$

1.3 低电量充电

电量较低但未到必须充电的阈值时，如果有空闲充电桩，优先去充电。

触发条件：

- $SOC \leq taskAvailableSoc$ (默认 60%)
- 有空闲充电桩

2. 任务抢占机制

2.1 充电任务打断

当满足以下条件时，正在充电的车辆可以被打断去执行生产任务：

条件：

- $allowInterruptTask = true$
- 待执行任务数 > 1
- 当前 $SOC > allowInterruptSoc$ (默认 45%)
- 没有空闲车辆
- 充电时长 $> mustChargeSeconds$ (默认 60 秒)

日志：

小车:{车辆名称}，因有任务且电量{SOC}高于{allowInterruptSoc}，中断充电去执行任务

2.2 维护任务打断

在充电或待命的车辆，可以被系统打断去执行生产任务。

防重入机制：

- 使用 `interruptingCarId` 确保同一时刻只有一辆车被打断
- 线程安全的双重检查锁

3. 补电功能 (Top-Up)

补电功能允许将指定车辆充电到 100% 并保持一段时间，用于电量校准或长期存储前的满电操作。

3.1 启动补电

方法:

```
[MethodMember(Name = "补电功能")]  
public void StartTopUp()
```

使用步骤:

1. 在界面上选中需要补电的车辆
2. 点击"补电功能"按钮
3. 系统自动标记车辆并调度其去充电

标签流转:

```
topup (requested) → topupInProgress → 完成后清除
```

3.2 补电流程

1. 去充电站阶段

- 添加 `topup=requested` 标签
- 添加 `shouldCharge=true` 标签
- 车辆调度到最近的充电站

2. 充电到 100%

- 当 $SOC \geq 100\%$ 时, 开始计时
- 添加 `topupInProgress=true` 标签
- 记录开始时间 `topupStart` 和结束时间 `topupEnd`

3. 保持充电

- 默认保持 `topUpMinutes` (默认 20 分钟)
- 可通过 `topupMinutes` 标签自定义时长

4. 补电完成

- 清除所有 `topup` 相关标签
- 添加 `mustToStandby=true`, 车辆返回待命点

注意事项:

- 补电期间车辆不会被任何任务打断
- 即使电量已满也会继续保持充电状态
- 补电时长可通过车辆标签 `topupMinutes` 自定义

4. 充电异常处理

系统可自动检测充电异常并采取应对措施。

4.1 启用异常检测

配置参数：

```
enableErrorChargeDetection = true
```

4.2 检测机制

默认判断逻辑：

```
protected virtual bool IsChargePointError(Car car, int chargeSiteId)
{
    // 电流 < 0 认为充电异常
    return car != null && Commons.CarValue(car, "electricCurrent") < 0;
}
```

触发条件：

- 车辆在充电站
- 检测到充电异常持续 10 秒

4.3 应对策略

情况 1：有空闲充电桩

```
清除 shouldCharge 标签
→ 添加 mustToAnotherChargeSite={当前充电桩ID}
→ 寻找其他充电桩
```

情况 2：无空闲充电桩

```
清除 shouldCharge 标签
→ 添加 mustToStandby=true
→ 返回待命点
```

日志：

```
小车:{车辆名称}, 充电桩{充电桩ID}异常, 无空闲充电桩, 返回待命点
小车:{车辆名称}, 充电桩{充电桩ID}异常, 切换到其他充电桩
```

5. 待命点管理

5.1 返回待命点条件

- $SOC > fullChargeSoc$ (默认 90%)
- 空闲时间 $> idleSeconds$ (默认 5 秒)
- 当前不在待命点
- 不在充电中
- 未被其他车辆阻塞

5.2 待命点选择

筛选条件:

- `standby=true` 的站点
- 未被占用
- 不在 `undeliverableSite` 黑名单中
- 未标记为 `unavailable`

选择策略:

- 选择距离最近的符合条件的待命点

快速开始

1. 继承并实现抽象类

```
using StandardScene.Charge;

public class MyChargeMission : AbstractChargeMission
{
    public override string GetChargeType(AbstractCar car)
    {
        // 返回充电站类型，例如根据车型区分
        return "ChargeStation";
    }

    public override string GetStandbyType(AbstractCar car)
    {
        // 返回待命点类型
        return "standby";
    }

    public override int WaitingMissions(AbstractCar car)
    {
        // 返回待执行的任务数量
        var mission = SimpleProject.proj.Missions
            .OfType<TransportMission>()
            .FirstOrDefault();

        if (mission == null) return 0;

        return mission.GetDeliveries()
            .Count(d => d.GetStatus() == DeliveryStatus.Waiting);
    }

    public override float LowerCarSoc(AbstractCar car, List<Site> allChargeSite)
    {
        // 返回所有非充电车辆中的最低电量
        var cars = SimpleLib.GetAllCars()
            .Where(c => !allChargeSite.Any(s => c.status.holdingLocks.Contains(s.id)));

        if (!cars.Any()) return 100;
    }
}
```

```

        return (float)cars.Min(c => Commons.CarValue((Car)c, "Soc"));
    }

    public override bool IsOnlineCar(Car car)
    {
        // 判断车辆是否在线
        return car.lstatus == "正常" || car.lstatus == "上线";
    }
}

```

2. 配置参数

在项目配置文件或界面中设置以下参数：

```

public class MyChargeMission : AbstractChargeMission
{
    public MyChargeMission()
    {
        // 基础充电参数
        fields["mustChargeSoc"] = "20";           // 必须充电阈值 20%
        fields["idleChargeSoc"] = "90";           // 空闲充电阈值 90%
        fields["taskAvailableSoc"] = "60";         // 任务可用电量 60%
        fields["fullChargeSoc"] = "90";           // 满电阈值 90%

        // 时间参数
        fields["idleChargeSeconds"] = "30";         // 空闲充电延迟 30 秒
        fields["mustChargeSeconds"] = "60";         // 必须充电时长 60 秒
        fields["idleSeconds"] = "5";               // 空闲延迟 5 秒

        // 任务抢占参数
        fields["allowInterruptTask"] = "true";       // 允许任务打断
        fields["allowInterruptSoc"] = "45";          // 打断最低电量 45%
        fields["minAllowFreeCarToChargeTaskNub"] = "0";

        // 充电策略参数
        fields["useLowerSocForCharge"] = "true";     // 使用最低电量车优先策略

        // 补电参数
        fields["topUpMinutes"] = "20";              // 补电保持时长 20 分钟

        // 异常检测参数
        fields["enableErrorChargeDetection"] = "true"; // 启用充电异常检测
    }
}

```

3. 启动充电任务

```

var chargeMission = new MyChargeMission();
chargeMission.Execute(); // 调用"启动进程"

```

配置参数详解

充电阈值参数

参数名	默认值	说明	推荐范围
mustChargeSoc	20	必须充电的最低电量（%）	10-30
idleChargeSoc	90	空闲时开始充电的电量阈值（%）	70-95
taskAvailableSoc	60	可执行任务的最低电量（%）	40-70
fullChargeSoc	90	认为充满电的阈值（%）	85-95
allowInterruptSoc	45	允许打断充电去执行任务的最低电量（%）	40-60

时间参数

参数名	默认值	说明	推荐范围
idleChargeSeconds	30	空闲多久后触发充电（秒）	10-60
mustChargeSeconds	60	充电多久后可以被打断（秒）	30-120
idleSeconds	5	空闲多久后返回待命点（秒）	3-10
topUpMinutes	5	补电保持时长（分钟）	5-60

策略参数

参数名	默认值	说明
allowInterruptTask	false	是否允许打断充电/待命车辆去执行任务
useLowerSocForCharge	true	是否只允许电量最低的车去充电
minAllowFreeCarToChargeTaskNub	0	当任务数 ≤ 此值时，允许车辆去任务，否则还是充电
enableErrorChargeDetection	false	是否启用充电异常检测

参数调优建议

高频任务场景

```
mustChargeSoc = 15           // 降低必须充电阈值
taskAvailableSoc = 50         // 降低任务可用电量
allowInterruptTask = true     // 启用任务打断
allowInterruptSoc = 40        // 降低打断阈值
```

低频任务场景

```
mustChargeSoc = 25      // 提高必须充电阈值
idleChargeSoc = 95      // 提高空闲充电阈值
allowInterruptTask = false // 禁用任务打断
idleChargeSeconds = 10   // 缩短空闲充电延迟
```

电池保养场景

```
topUpMinutes = 10      // 延长补电时长
enableErrorChargeDetection = true // 启用异常检测
```

高级功能

1. 自定义充电异常判断

重写 `IsChargePointError` 方法，实现自定义的充电异常判断逻辑：

```
protected override bool IsChargePointError(Car car, int chargeSiteId)
{
    var site = SimpleLib.GetSite(chargeSiteId);

    // 方式 1: 检查充电桩设备状态
    if (site.fields.TryGetValue("chargerStatus", out var status))
    {
        if (status == "fault" || status == "offline")
            return true;
    }

    // 方式 2: 检查车辆充电电流
    var current = Commons.CarValue(car, "electricCurrent");
    if (current < 0.1) // 电流太小
        return true;

    // 方式 3: 检查电压异常
    var voltage = Commons.CarValue(car, "voltage");
    if (voltage < 40 || voltage > 60) // 电压异常
        return true;

    // 方式 4: 检查车辆报错
    if (car.tags.TryGetValue("chargeError", out var error))
        return true;

    return false;
}
```


2. 自定义到达/离开动作

```
public override void ArriveAction(Car car, Site site)
{
    // 到达充电站时的自定义操作
    if (site.fields.ContainsKey("Charge"))
    {
        // 发送充电指令给车辆
        SendChargeCommand(car, true);

        // 记录充电开始时间
        car.tags["chargeStartTime"] = DateTime.Now.ToString("o");

        Diagnosis.Log($"车辆 {car.name} 到达充电站 {site.name}", "Charge", true);
    }

    // 到达待命点时的自定义操作
    if (site.fields.ContainsKey("standby"))
    {
        // 发送待机指令
        SendStandbyCommand(car);

        Diagnosis.Log($"车辆 {car.name} 到达待命点 {site.name}", "Standby", true);
    }
}

public override void LeaveAction(Car car, Site site)
{
    // 离开充电站时的自定义操作
    if (site.fields.ContainsKey("Charge"))
    {
        // 停止充电
        SendChargeCommand(car, false);

        // 统计充电时长
        if (car.tags.TryGetValue("chargeStartTime", out var startStr) &&
            DateTime.TryParse(startStr, out var startTime))
        {
            var duration = (DateTime.Now - startTime).TotalMinutes;
            Diagnosis.Log($"车辆 {car.name} 充电时长: {duration:F1} 分钟", "Charge",
true);
        }

        car.tags.Remove("chargeStartTime");
    }
}
```

3. 车辆筛选逻辑

```
protected override bool ValidateCarForProcessing(Car car)
{
    // 基础验证
    if (!base.ValidateCarForProcessing(car))
        return false;
```

```
// 自定义验证：跳过特殊车辆
if (car.tags.Contains("maintenance"))
    return false;

// 自定义验证：只处理特定车型
if (car.fields.TryGetValue("carType", out var carType))
{
    if (carType != "AGV" && carType != "AMR")
        return false;
}

return true;
}
```

4. 禁止回充电/待命

```
public override bool ForbidBackToStandbyOrCharge(AbstractCar car)
{
    // 场景 1: 车辆正在执行特殊任务
    if (car.tags.Contains("emergencyTask"))
        return true;

    // 场景 2: 车辆被锁定
    if (car.tags.Contains("locked"))
        return true;

    // 场景 3: 车辆在特定区域
    var currentSite = SimpleLib.GetSite(car.GetLastSite());
    if (currentSite?.fields.TryGetValue("zone", out var zone) == true)
    {
        if (zone == "productionArea")
            return true;
    }

    return false;
}
```

扩展点说明

必须重写的方法

方法名	说明	默认实现
GetChargeType	获取车辆对应的充电站类型	返回 "Charge"
GetStandbyType	获取车辆对应的待命点类型	返回 "standby"
WaitingMissions	获取待执行的任务数量	返回第一个 TransportMission 的等待任务数

方法名	说明	默认实现
LowerCarSoc	获取最低电量车辆的 SOC	返回 0
IsonlineCar	判断车辆是否在线	返回 false

可选重写的方法

方法名	说明	默认实现
ArriveAction	车辆到达充电站/待命点时的动作	空实现
LeaveAction	车辆离开充电站/待命点时的动作	空实现
ForbidBackToStandbyOrCharge	是否禁止车辆回充电/待命	返回 false
ValidateCarForProcessing	验证车辆是否可处理	检查状态和位置
IsChargePointError	判断充电桩是否异常	检查电流 < 0

常见场景示例

场景 1：多车型混合调度

```
public class MultiTypeChargeMission : AbstractChargeMission
{
    public override string GetChargeType(AbstractCar car)
    {
        // 根据车型返回不同的充电站类型
        if (car.fields.TryGetValue("carType", out var type))
        {
            switch (type)
            {
                case "Forklift":
                    return "ForkliftCharger";
                case "AMR":
                    return "AMRCharger";
                case "AGV":
                    return "AGVCharger";
            }
        }
        return "Charge";
    }

    public override string GetStandbyType(AbstractCar car)
    {
        // 不同车型有不同的待命区域
        if (car.fields.TryGetValue("carType", out var type))
        {
            return $"{type}Standby";
        }
        return "standby";
    }
}
```

```

    public override int WaitingMissions(AbstractCar car)
    {
        // 按车型统计任务
        var carType = car.fields.TryGetValue("carType", out var type) ? type :
        "";

        return SimpleProject.proj.Missions
            .OfType<TransportMission>()
            .SelectMany(m => m.GetDeliveries())
            .Count(d => d.GetStatus() == DeliveryStatus.Waiting &&
                d.fields.TryGetValue("carType", out var taskType) &&
                taskType == carType);
    }
}

```

场景 2：分区管理

```

public class ZonedChargeMission : AbstractChargeMission
{
    protected override bool ValidateCarForProcessing(Car car)
    {
        if (!base.ValidateCarForProcessing(car))
            return false;

        // 只处理当前区域的车辆
        var currentSite = SimpleLib.GetSite(car.GetLastSite());
        if (currentSite?.fields.TryGetValue("zone", out var zone) == true)
        {
            // 假设任务配置了管理区域
            if (fields.TryGetValue("managedZone", out var managedZone))
            {
                return zone == managedZone;
            }
        }
        return true;
    }

    public override string GetChargeType(AbstractCar car)
    {
        // 根据车辆所在区域返回充电站类型
        var currentSite = SimpleLib.GetSite(car.GetLastSite());
        if (currentSite?.fields.TryGetValue("zone", out var zone) == true)
        {
            return $"Charge_{zone}";
        }
        return "Charge";
    }
}

```

场景 3：优先级车辆

```
public class PriorityChargeMission : AbstractChargeMission
{
    public override float LowerCarSoc(AbstractCar car, List<Site> allChargeSite)
    {
        // 优先级车辆可以随时充电
        if (car.tags.Contains("priority"))
            return 100; // 总是返回 100，使优先级车辆总能满足充电条件

        // 普通车辆需要比较电量
        var normalCars = SimpleLib.GetAllCars()
            .Where(c => !c.tags.Contains("priority") &&
                !allChargeSite.Any(s => c.status.holdingLocks.Contains(s.id)));

        if (!normalCars.Any()) return 100;

        return (float)normalCars.Min(c => Commons.CarValue((Car)c, "Soc"));
    }

    public override bool ForbidBackToStandbyOrCharge(AbstractCar car)
    {
        // 优先级车辆在有任务时不回待命点
        if (car.tags.Contains("priority"))
        {
            var taskCount = WaitingMissions(car);
            if (taskCount > 0)
                return true;
        }

        return base.ForbidBackToStandbyOrCharge(car);
    }
}
```

故障排查

问题 1：车辆不去充电

可能原因：

1. 电量未达到充电阈值
2. 没有空闲充电桩
3. 车辆被其他逻辑阻塞（occupied、blocking 标签）
4. `ForbidBackToStandbyOrCharge` 返回 true
5. `useLowerSocForCharge=true` 但车辆不是电量最低的

排查步骤：

```
// 检查车辆状态
var car = SimpleLib.GetCar(carId);
Console.WriteLine($"SOC: {Commons.CarValue(car, "Soc")}");
Console.WriteLine($"Tags: {string.Join(", ", car.tags.Keys)}");
```

```

Console.WriteLine($"holdingLocks: {string.Join(", ", car.status.holdingLocks)}");
Console.WriteLine($"pendingLocks: {string.Join(", ", car.status.pendingLocks)}");

// 检查充电站
var chargeSites = SimpleLib.GetAllSites()
    .Where(s => s.fields.ContainsKey("Charge"))
    .ToList();
Console.WriteLine($"总充电站: {chargeSites.Count}");
Console.WriteLine($"占用充电站的车: {SimpleLib.GetAllCars()
    .Count(c => chargeSites.Any(s => c.status.holdingLocks.Contains(s.id)))}");

// 检查配置
Console.WriteLine($"mustChargeSoc: {AcmStatus.mustChargeSoc}");
Console.WriteLine($"idleChargeSoc: {AcmStatus.idleChargeSoc}");
Console.WriteLine($"useLowerSocForCharge: {AcmStatus.useLowerSocForCharge}");

```

问题 2：车辆充电后不离开

可能原因：

1. 电量未达到 `fullChargeSoc`
2. 正在补电（`topup` 或 `topupInProgress` 标签）
3. 充电时长未达到 `mustChargeSeconds`
4. `LowerSocCar` 逻辑导致车辆需要继续充电

排查步骤：

```

var car = SimpleLib.GetCar(carId);
Console.WriteLine($"SOC: {Commons.CarValue(car, "Soc")}");
Console.WriteLine($"fullChargeSoc: {AcmStatus.fullChargeSoc}");

if (car.tags.TryGetValue("charging", out var chargeTime))
{
    var elapsed = (DateTime.Now - DateTime.Parse(chargeTime)).TotalSeconds;
    Console.WriteLine($"充电时长: {elapsed} 秒, 要求: {AcmStatus.mustChargeSeconds} 秒");
}

Console.WriteLine($"补电中: {car.tags.Contains("topup") || car.tags.Contains("topupInProgress")}");
Console.WriteLine($"最低电量车: {LowerCarSoc(car, chargeSites)}");

```

问题 3：充电异常检测误触发

可能原因：

1. `IsChargePointError` 判断条件太严格
2. 电流传感器数据波动
3. 充电桩启动延迟

问题 4：车辆循环在充电站和待命点之间

可能原因：

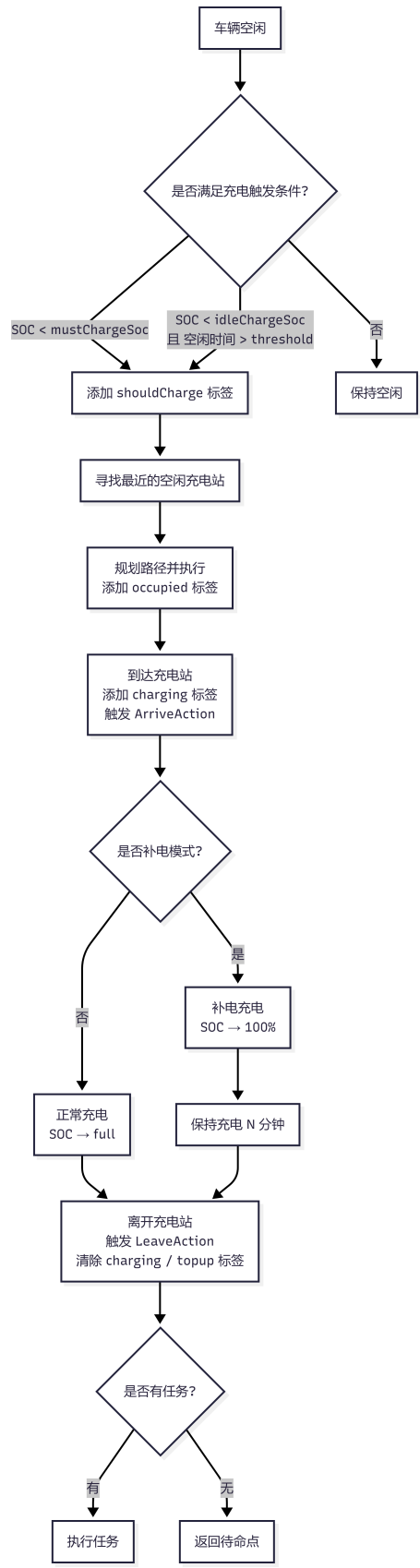
- 1. fullChargeSoc 和 idleChargeSoc 设置不合理
- 2. 电量读数波动
- 3. 充电站和待命点判断逻辑冲突

附录

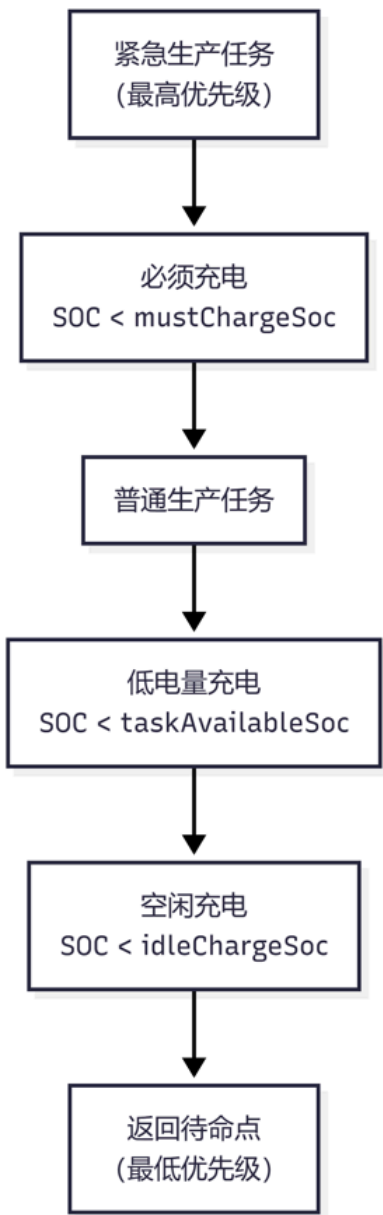
A. 车辆标签说明

标签名	类型	说明
shouldCharge	Flag	标记车辆需要充电
charging	DateTime	开始充电的时间
topup	Flag	请求补电
topupInProgress	Flag	正在补电中
topupStart	DateTime	补电开始时间
topupEnd	DateTime	补电结束时间
topupMinutes	double	自定义补电时长（分钟）
mustToStandby	Flag	必须返回待命点
mustToAnotherChargeSite	int	需要切换到其他充电桩，值为异常充电桩ID
occupied	string	车辆被占用，值为任务类型（如 toCharge、toStandby）
blocking	Flag	车辆被阻塞
idle	DateTime	车辆空闲开始时间
undeliverableSite	string	无法到达的站点列表（逗号分隔）
dest	int	目标站点ID

B. 充电流程状态图



C. 任务优先级示意



D. 配置模板版本历史

版本	日期	更新内容
1.0	2025-01	初始版本，包含基础充电管理功能
1.1	2025-01	新增补电功能和电量校准
1.2	2025-01	新增充电异常检测与自动切换充电桩
1.3	2025-01	优化任务抢占机制，增加防重入保护