四种策略整体对比

本 notebook 汇总并对比以下四种策略的回测结果（以上证50为基准）：

- 等权配置策略（`equal_weights`）
- 动态权重策略（有市场状态， `dynamic_weights`）
- 动态权重策略（无市场状态， `dynamic_weights_without_regime`）
- 规则配置策略（`rule_based`）

下面将统一读取回测 JSON 结果，构建：

1. 四种策略的核心绩效指标对比表（使用第5组组合：高因子组合）
2. 四种策略与基准的累计收益率对比曲线

运行顺序：从上到下依次运行所有代码单元。

In [4]:
```python
import json
import os
from pathlib import Path

import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['STHeiti', 'SimHei', 'Microsoft YaHei', '
plt.rcParams['axes.unicode_minus'] = False

base_path = Path('回测') / 'res'

strategy_files = {
    '等权配置': 'equal_weights.json',
    '动态权重(有状态)': 'dynamic_weights.json',
    '动态权重(无状态)': 'dynamic_weights_without_regime.json',
    '规则配置': 'rule_based.json',
}

strategy_data = {}
for name, fname in strategy_files.items():
    fpath = base_path / fname
    if not fpath.exists():
        raise FileNotFoundError(f'未找到回测结果文件: {fpath}')
    with open(fpath, 'r', encoding='utf-8') as f:
        strategy_data[name] = json.load(f)

print('已成功读取策略回测结果:', list(strategy_data.keys()))
```

```
已成功读取策略回测结果: ['等权配置', '动态权重(有状态)', '动态权重(无状态)', '规则配置']
```

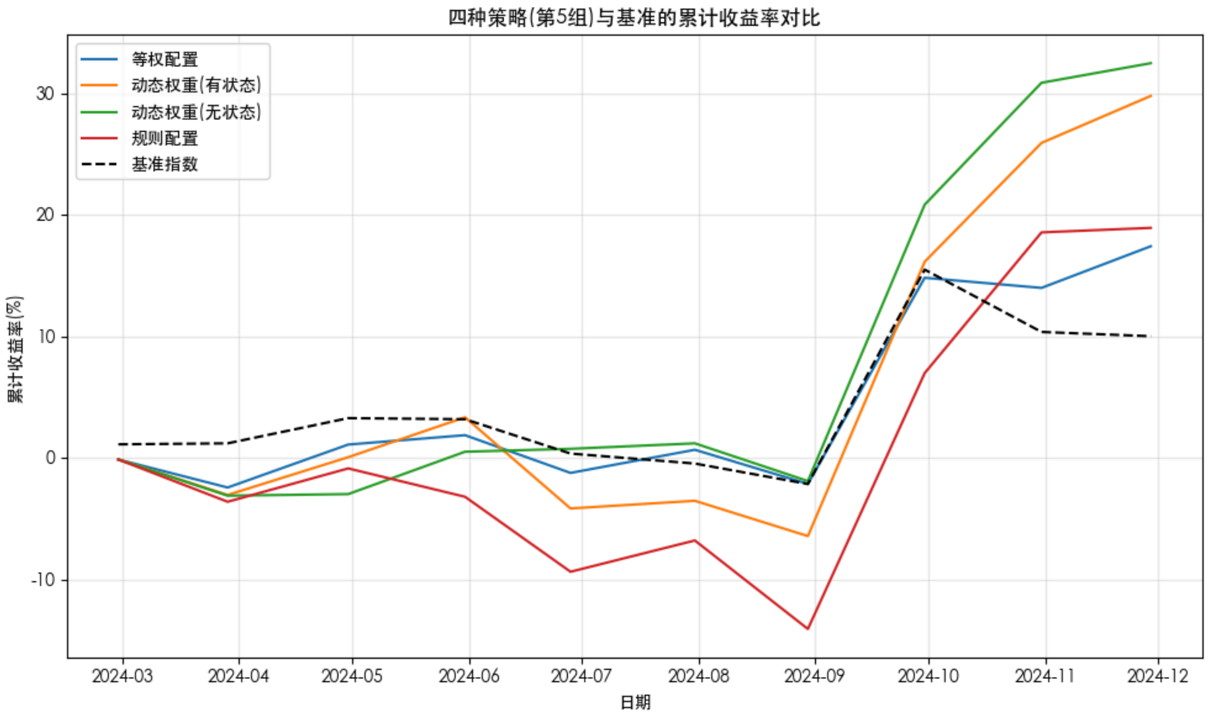
In [5]:
```python
# 构建四个策略的核心绩效指标对比表（第5组 = 高因子组合）

metrics = [
    '整体@平均收益(%)',
    '整体@标准差(%)',
    '整体@夏普比率',
    '整体@胜率(%)',
    '整体@最大回撤%(日线)',
]

summary_rows = []
for name, data in strategy_data.items():
    df_ret = pd.DataFrame(data['收益率检验'])
    row5 = df_ret[df_ret['组名'] == '第5组'].iloc[0]

    # 最终累计收益率（取“累计收益率”中第5组与基准的最后一个值）
    df_cum = pd.DataFrame(data['累计收益率'])
    final_port = df_cum['第5组'].iloc[-1]
    final_bench = df_cum['基准'].iloc[-1]

    summary = {
        '策略': name,
        '最终累计收益(第5组, %)': final_port,
        '基准最终累计收益(%)': final_bench,
    }
    for m in metrics:
        summary[m] = row5[m]
    summary_rows.append(summary)

summary_df = pd.DataFrame(summary_rows).set_index('策略')
summary_df.round(2)
```

Out[5]:

	最终累计收益(第5组, %)	基准最终累计收益(%)	整体@平均收益(%)	整体@标准差(%)	整体@夏普比率	整体@胜率(%)	整体@最大回撤%(日线)
策略							
等权配置	17.42	10.03	1.97	6.26	0.32	55.56	-13.96
动态权重(有状态)	29.77	10.03	3.28	9.04	0.36	66.67	-16.91
动态权重(无状态)	32.46	10.03	3.45	8.14	0.42	77.78	-11.99
规则配置	18.93	10.03	2.36	9.99	0.24	55.56	-21.54

In [6]:

```
# 四个策略与基准的累计收益率对比曲线（第5组组合）

cum_series = {}
benchmark = None

for name, data in strategy_data.items():
    df_cum = pd.DataFrame(data['累计收益率']).copy()
    df_cum['截止日'] = pd.to_datetime(df_cum['截止日'])
    df_cum.set_index('截止日', inplace=True)

    # 记录策略本身（第5组）
    cum_series[name] = df_cum['第5组']

    # 记录一次基准曲线
    if benchmark is None:
        benchmark = df_cum['基准']

cum_all = pd.concat(cum_series, axis=1)

plt.figure(figsize=(10, 6))
for col in cum_all.columns:
    plt.plot(cum_all.index, cum_all[col], label=col)

plt.plot(benchmark.index, benchmark.values, label='基准指数', linestyle='--',
plt.xlabel('日期')
plt.ylabel('累计收益率(%)')
plt.title('四种策略(第5组)与基准的累计收益率对比')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```



1. 数据解析与对齐（统一为第 5 组多头组合）

本节将从四个 JSON 文件中抽取：

- 月度组合收益（第 5 组，多头组合）
- 对应基准（月度收益）

并对时间索引进行统一对齐（按月）。

In [7]:
```python
import numpy as np

MONTHS_PER_YEAR = 12


def extract_return_series(strategy_data_dict, group_name='第5组'):
    """从 JSON 中提取:
    - 第 5 组多头组合月度收益（百分比 -> 小数）
    - 基准指数月度收益（小数）
    并统一时间索引。
    """
    ret_series = {}
    bench_series = None

    for strat_name, data in strategy_data_dict.items():
        df = pd.DataFrame(data['每期收益率']).copy()
        df['截止日'] = pd.to_datetime(df['截止日'])
        df = df.set_index('截止日').sort_index()

        r = df[group_name] / 100.0  # 转为小数
        b = df['基准'] / 100.0

        ret_series[strat_name] = r
        if bench_series is None:
            bench_series = b
        else:
            # 只要第一次基准，后续仅用于交集对齐
            bench_series = bench_series

    # 对齐所有策略的共同时间区间
    common_index = bench_series.index
    for s in ret_series.values():
        common_index = common_index.intersection(s.index)
    common_index = common_index.sort_values()

    bench_aligned = bench_series.reindex(common_index)
    ret_aligned = {k: v.reindex(common_index) for k, v in ret_series.items()

    return bench_aligned, ret_aligned


bench_ret, strat_ret = extract_return_series(strategy_data)

bench_ret.head(), {k: v.head() for k, v in strat_ret.items()}
```

```
Out[7]: (截止日
         2024-02-29    0.0115
         2024-03-29    0.0007
         2024-04-30    0.0204
         2024-05-31   -0.0008
         2024-06-28   -0.0273
         Name: 基准, dtype: float64,
         {'等权配置': 截止日
          2024-02-29   -0.0010
          2024-03-29   -0.0231
          2024-04-30    0.0362
          2024-05-31    0.0076
          2024-06-28   -0.0305
          Name: 第5组, dtype: float64,
          '动态权重(有状态)': 截止日
          2024-02-29   -0.0010
          2024-03-29   -0.0292
          2024-04-30    0.0320
          2024-05-31    0.0328
          2024-06-28   -0.0725
          Name: 第5组, dtype: float64,
          '动态权重(无状态)': 截止日
          2024-02-29   -0.0010
          2024-03-29   -0.0298
          2024-04-30    0.0014
          2024-05-31    0.0358
          2024-06-28    0.0024
          Name: 第5组, dtype: float64,
          '规则配置': 截止日
          2024-02-29   -0.0010
          2024-03-29   -0.0348
          2024-04-30    0.0285
          2024-05-31   -0.0235
          2024-06-28   -0.0637
          Name: 第5组, dtype: float64})
```

2. 收益与风险指标计算（逐策略）

本节对每个策略（第 5 组多头组合）计算：

- 年化收益率（CAGR）
- 年化波动率
- 最大回撤与回撤持续时间
- 胜率（正收益月份占比）

风险调整后收益指标：

- Sharpe Ratio
- Sortino Ratio
- Calmar Ratio

```
In [8]: def compute_drawdown_stats(returns: pd.Series, periods_per_year: int = MONTH
            """基于月度收益率（小数）计算最大回撤及回撤持续时间（单位：月）。"""
            wealth = (1 + returns).cumprod()
            running_max = wealth.cummax()
            drawdown = wealth / running_max - 1.0

            max_dd = drawdown.min()

            # 回撤持续时间：连续处于回撤状态（drawdown < 0）的最长长度
            dd_flag = drawdown < 0
            max_duration = 0
            cur = 0
            for in_dd in dd_flag:
                if in_dd:
                    cur += 1
                    max_duration = max(max_duration, cur)
                else:
                    cur = 0

            return max_dd, max_duration


        def compute_risk_metrics(returns: pd.Series, periods_per_year: int = MONTHS_
            """给定月度收益率（小数），计算收益与风险及风险调整后指标。"""
            r = returns.dropna()
            n = len(r)
            if n == 0:
                return {}

            # 年化收益（CAGR）
            total_return = (1 + r).prod() - 1
            years = n / periods_per_year
            cagr = (1 + total_return) ** (1 / years) - 1 if years > 0 else np.nan

            # 年化波动
            vol = r.std(ddof=1) * np.sqrt(periods_per_year)

            # 最大回撤及持续时间
            max_dd, dd_duration = compute_drawdown_stats(r, periods_per_year)

            # 胜率（月度正收益占比）
            win_rate = (r > 0).mean()

            # Sharpe（假设无风险利率为 0）
            sharpe = (r.mean() / r.std(ddof=1)) * np.sqrt(periods_per_year) if r.std

            # Sortino（只考虑下行波动）
            downside = r[r < 0]
            downside_std = downside.std(ddof=1)
            sortino = (r.mean() / downside_std) * np.sqrt(periods_per_year) if downs

            # Calmar
            calmar = cagr / abs(max_dd) if max_dd < 0 else np.nan

            return {
```

```
        'CAGR': cagr,
        'Vol': vol,
        'MaxDD': max_dd,
        'DD_Duration_Months': dd_duration,
        'WinRate': win_rate,
        'Sharpe': sharpe,
        'Sortino': sortino,
        'Calmar': calmar,
    }


risk_metrics = {name: compute_risk_metrics(ret) for name, ret in strat_ret.i

pd.DataFrame(risk_metrics).T[['CAGR', 'Vol', 'Sharpe', 'Sortino', 'MaxDD', '
```

```
/var/folders/49/t84jntk56wn101s6026p72cc0000gn/T/ipykernel_8011/2697567406.p
y:69: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.m
ap instead.
  pd.DataFrame(risk_metrics).T[['CAGR', 'Vol', 'Sharpe', 'Sortino', 'MaxDD',
'DD_Duration_Months', 'WinRate', 'Calmar']].applymap(lambda x: round(x, 4) i
f isinstance(x, (int, float, np.floating)) else x)
```

Out[8]:

	CAGR	Vol	Sharpe	Sortino	MaxDD	DD_Duration_Months	WinRate	Calmar
等权配置	0.2124	0.2056	1.0294	4.6733	-0.0393	3.0	0.5	5.4106
动态权重(有状态)	0.3671	0.2977	1.1856	3.4556	-0.0945	3.0	0.6	3.8853
动态权重(无状态)	0.4009	0.2687	1.3806	6.3496	-0.0306	2.0	0.7	13.1028
规则配置	0.2313	0.3275	0.7754	2.3725	-0.1393	6.0	0.5	1.6603

3. 相对基准的超额收益与风险（CAPM / 信息比率）

本节对每个策略相对于基准指数计算：

- 年化超额收益（策略 CAGR − 基准 CAGR）
- 跟踪误差（Tracking Error）

- 信息比率（Information Ratio）
- CAPM Alpha / Beta（基于月度收益，Alpha 年化）
- 与基准的相关系数

In [9]:
```python
def compute_capm_relative_metrics(returns: pd.Series, bench: pd.Series, peri
    """相对基准的超额收益、跟踪误差、信息比率以及 CAPM Alpha/Beta。"""
    df = pd.concat({'ret': returns, 'bench': bench}, axis=1).dropna()
    r = df['ret']
    b = df['bench']
    n = len(df)
    if n == 0:
        return {}

    # 基准与策略各自 CAGR
    rm = compute_risk_metrics(r, periods_per_year)
    bm = compute_risk_metrics(b, periods_per_year)

    # 年化超额收益（CAGR 差）
    ann_excess_return = rm['CAGR'] - bm['CAGR']

    # 跟踪误差与信息比率
    diff = r - b
    te = diff.std(ddof=1) * np.sqrt(periods_per_year)
    ir = (diff.mean() / diff.std(ddof=1)) * np.sqrt(periods_per_year) if dif

    # CAPM: r_t = alpha + beta * b_t + eps
    cov_rb = np.cov(r, b, ddof=1)[0, 1]
    var_b = np.var(b, ddof=1)
    beta = cov_rb / var_b if var_b > 0 else np.nan
    alpha_monthly = r.mean() - beta * b.mean() if beta == beta else np.nan
    alpha_annual = alpha_monthly * periods_per_year if alpha_monthly == alph

    corr = r.corr(b)

    return {
        'Ann_Excess_Return': ann_excess_return,
        'Tracking_Error': te,
        'Information_Ratio': ir,
        'CAPM_Alpha_Annual': alpha_annual,
        'CAPM_Beta': beta,
        'Corr_with_Bench': corr,
    }


relative_metrics = {name: compute_capm_relative_metrics(ret, bench_ret) for

pd.DataFrame(relative_metrics).T.applymap(lambda x: round(x, 4) if isinstanc
```

```
/var/folders/49/t84jntk56wn101s6026p72cc0000gn/T/ipykernel_8011/707394721.p
y:43: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.m
ap instead.
  pd.DataFrame(relative_metrics).T.applymap(lambda x: round(x, 4) if isinsta
nce(x, (int, float, np.floating)) else x)
```

Out[9]:

	Ann_Excess_Return	Tracking_Error	Information_Ratio	CAPM_Alpha_Annual	CAP
等权配置	0.0909	0.0733	1.0552	0.0910	
动态权重(有状态)	0.2455	0.1743	1.2547	0.2014	
动态权重(无状态)	0.2794	0.1600	1.4792	0.2363	
规则配置	0.1098	0.2154	0.5554	0.0986	

4. 因子有效性与稳定性（多空收益 & IC 统计）

本节基于 JSON 中的：

- `多空月度收益(原始)`：第 1 组 vs 第 5 组合多空收益
- `因子延续性检验`：月度 IC 序列
- `因子区分度检验`：整体 T-Stat / P-Value

评估因子的：

- 多空组合年化收益与 Sharpe
- IC 的均值、标准差与 IC_IR（年化）
- 因子是否具有统计显著性（基于 T-Stat / P-Value）

In [10]:
```python
def extract_long_short_returns(strategy_data_dict):
    """提取多空（月度）收益（第 1 组 vs 第 5 组），返回小数收益率。"""
    ls_ret = {}
    for strat_name, data in strategy_data_dict.items():
        df = pd.DataFrame(data['多空月度收益(原始)']).copy()
        df['截止日_str'] = pd.to_datetime(df['截止日_str'])
        df = df.set_index('截止日_str').sort_index()
        r_ls = df['第一组vs最后一组'] / 100.0
        ls_ret[strat_name] = r_ls
    return ls_ret
```

```python
def extract_ic_series(strategy_data_dict):
    """提取月度 IC 序列。"""
    ic_series = {}
    for strat_name, data in strategy_data_dict.items():
        df_ic = pd.DataFrame(data['因子延续性检验']).copy()
        df_ic['检验截止日'] = pd.to_datetime(df_ic['检验截止日'])
        df_ic = df_ic.set_index('检验截止日').sort_index()
        ic_series[strat_name] = df_ic['IC']
    return ic_series


ls_ret = extract_long_short_returns(strategy_data)
ic_series = extract_ic_series(strategy_data)

# 多空组合年化收益与 Sharpe
ls_metrics = {name: compute_risk_metrics(r) for name, r in ls_ret.items()}

# IC 统计
ic_stats = {}
for name, ic in ic_series.items():
    ic_clean = ic.dropna()
    if len(ic_clean) == 0:
        ic_stats[name] = {}
        continue
    ic_mean = ic_clean.mean()
    ic_std = ic_clean.std(ddof=1)
    ic_ir = (ic_mean / ic_std) * np.sqrt(MONTHS_PER_YEAR) if ic_std > 0 else
    ic_stats[name] = {
        'IC_Mean': ic_mean,
        'IC_Std': ic_std,
        'IC_IR_Annual': ic_ir,
    }

# 区分度检验的 T-Stat / P-Value (第一组 vs 最后一组)
discrimination_stats = {}
for name, data in strategy_data.items():
    df_disc = pd.DataFrame(data['因子区分度检验'])
    row = df_disc.iloc[0]
    discrimination_stats[name] = {
        'T_Stat': row['整体@T-Stat'],
        'P_Value': row['整体@P-Value'],
    }

ls_df = pd.DataFrame(ls_metrics).T[[
    'CAGR', 'Vol', 'Sharpe', 'MaxDD', 'WinRate'
]].applymap(lambda x: round(x, 4) if isinstance(x, (int, float, np.floating)

ic_df = pd.DataFrame(ic_stats).T.applymap(lambda x: round(x, 4) if isinstanc

disc_df = pd.DataFrame(discrimination_stats).T.applymap(lambda x: round(x, 4

ls_df, ic_df, disc_df
/var/folders/49/t84jntk56wn101s6026p72cc0000gn/T/ipykernel_8011/3456618257.p
y:58: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.m
ap instead.
  ]].applymap(lambda x: round(x, 4) if isinstance(x, (int, float, np.floatin
g)) else x)
/var/folders/49/t84jntk56wn101s6026p72cc0000gn/T/ipykernel_8011/3456618257.p
y:60: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.m
ap instead.
  ic_df = pd.DataFrame(ic_stats).T.applymap(lambda x: round(x, 4) if isinsta
nce(x, (int, float, np.floating)) else x)
/var/folders/49/t84jntk56wn101s6026p72cc0000gn/T/ipykernel_8011/3456618257.p
y:62: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.m
ap instead.
  disc_df = pd.DataFrame(discrimination_stats).T.applymap(lambda x: round(x,
4) if isinstance(x, (int, float, np.floating)) else x)
```

Out[10]:

```
(              CAGR     Vol  Sharpe   MaxDD  WinRate
 等权配置       0.0074  0.0964  0.1195 -0.0445      0.4
 动态权重(有状态) -0.2450  0.1678 -1.5749 -0.2403      0.3
 动态权重(无状态) -0.2655  0.2161 -1.3010 -0.2506      0.5
 规则配置      -0.0715  0.2393 -0.1966 -0.2295      0.4,
            IC_Mean  IC_Std  IC_IR_Annual
 等权配置       -0.0300  0.1416       -0.7339
 动态权重(有状态)  -0.0878  0.1892       -1.6072
 动态权重(无状态)  -0.0833  0.2575       -1.1209
 规则配置        0.0167  0.2819        0.2048,
            T_Stat  P_Value
 等权配置         0.11    54.18
 动态权重(有状态)   -1.45     9.29
 动态权重(无状态)   -1.19    13.38
 规则配置        -0.18    43.13)
```

5. 可实施性与交易成本代理（换手率分析）

本节使用 JSON 中的 `换手率`：

- 计算第 5 组组合的平均月度换手率
- 定义单位换手收益 = 平均月度收益 / 平均月度换手率（均使用百分比口径）

In [11]:

```python
turnover_stats = {}

for name, data in strategy_data.items():
    df_to = pd.DataFrame(data['换手率']).copy()
    df_to['开始日'] = pd.to_datetime(df_to['开始日'])
    df_to = df_to.set_index('开始日').sort_index()

    avg_turnover = df_to['第5组'].mean()  # 单位：%

    # 平均月度收益（同口径，取 JSON 中“月度绝对收益(原始)”第 5 组）
    df_ret_raw = pd.DataFrame(data['月度绝对收益(原始)']).copy()
    df_ret_raw['截止日_str'] = pd.to_datetime(df_ret_raw['截止日_str'])
    df_ret_raw = df_ret_raw.set_index('截止日_str').sort_index()
    avg_ret_pct = df_ret_raw['第5组'].mean()  # 单位：%
    unit_ret_per_turnover = avg_ret_pct / avg_turnover if avg_turnover not i

    turnover_stats[name] = {
        'Avg_Turnover_pct': avg_turnover,
        'Avg_Monthly_Return_pct': avg_ret_pct,
        'Unit_Return_per_Turnover': unit_ret_per_turnover,
    }

turnover_df = pd.DataFrame(turnover_stats).T.applymap(lambda x: round(x, 4)
turnover_df
```

```
/var/folders/49/t84jntk56wn101s6026p72cc0000gn/T/ipykernel_8011/1519177830.p
y:24: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.m
ap instead.
  turnover_df = pd.DataFrame(turnover_stats).T.applymap(lambda x: round(x,
4) if isinstance(x, (int, float, np.floating)) else x)
```

Out[11]:

	Avg_Turnover_pct	Avg_Monthly_Return_pct	Unit_Return_per_Turnover
等权配置	51.1111	1.764	0.0345
动态权重(有状态)	24.4444	2.941	0.1203
动态权重(无状态)	42.2222	3.091	0.0732
规则配置	32.2222	2.116	0.0657

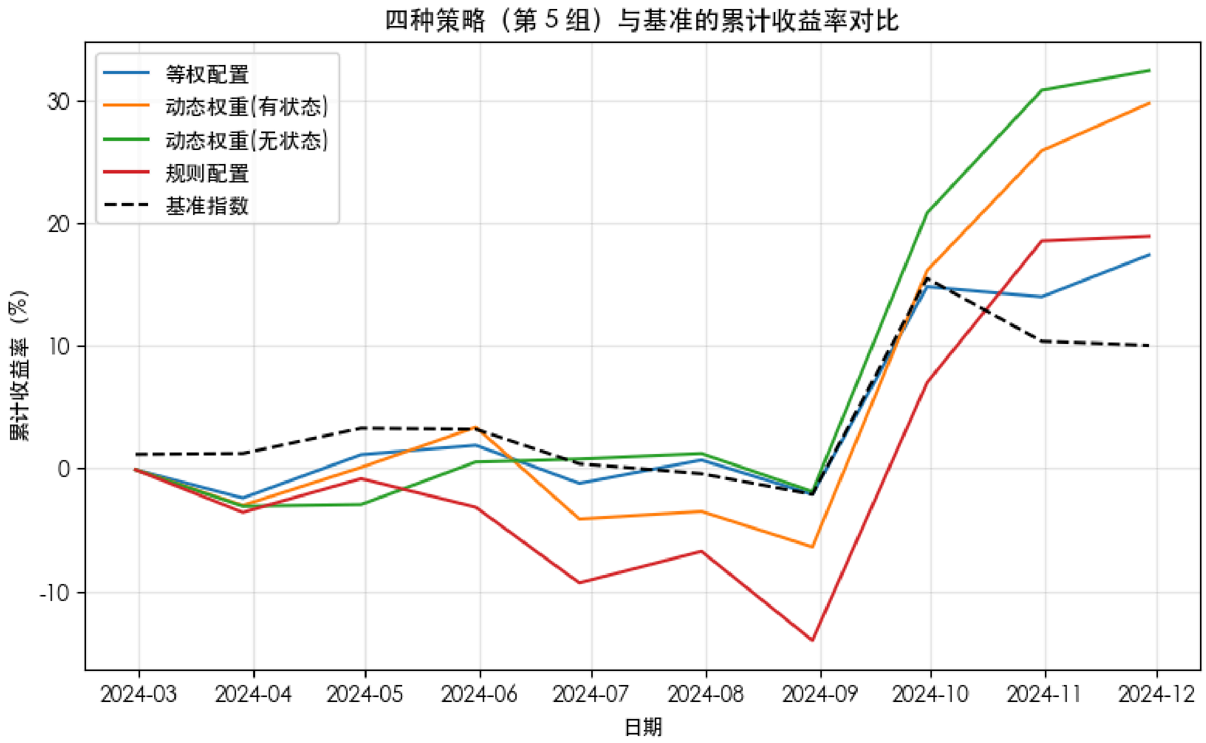
6. 可视化：累计收益 / 回撤 / 风险收益比

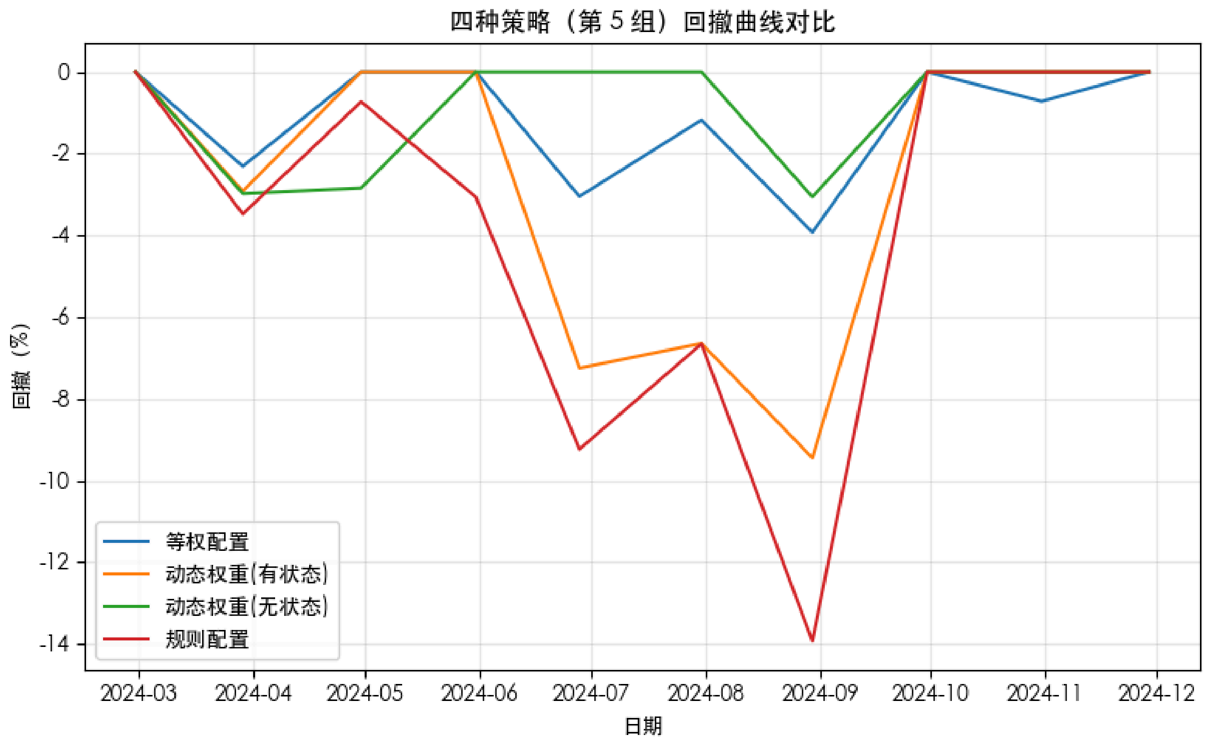
本节生成：

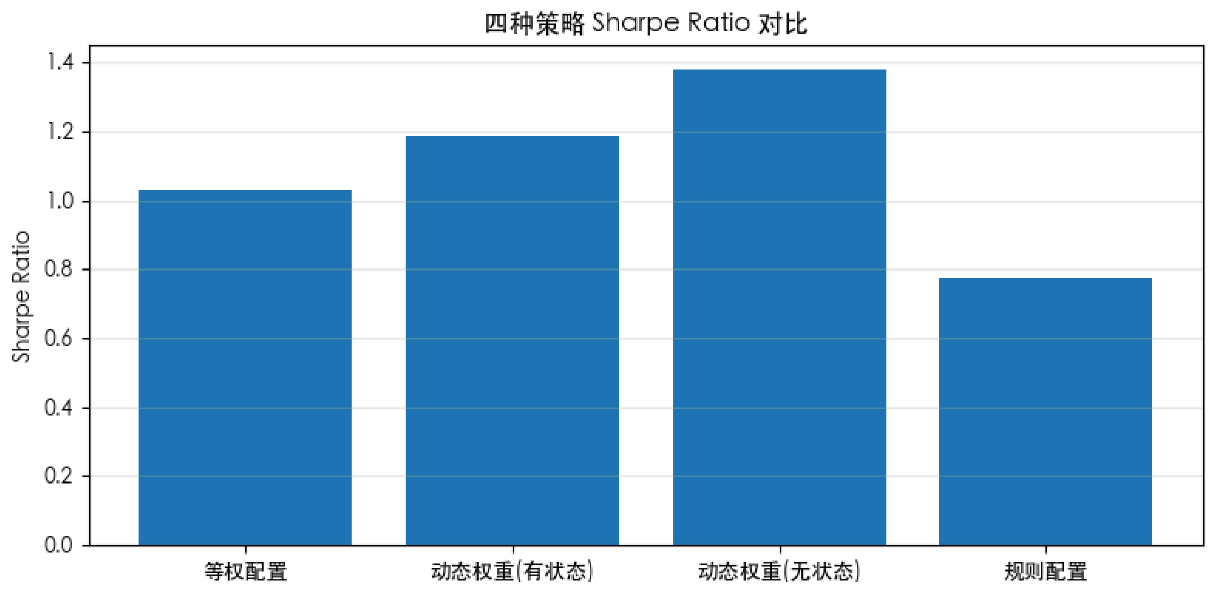
1. 四个策略的累计收益曲线对比图（多头第 5 组）
2. 四个策略的回撤曲线对比图
3. 四个策略 Sharpe / Information Ratio 的柱状图

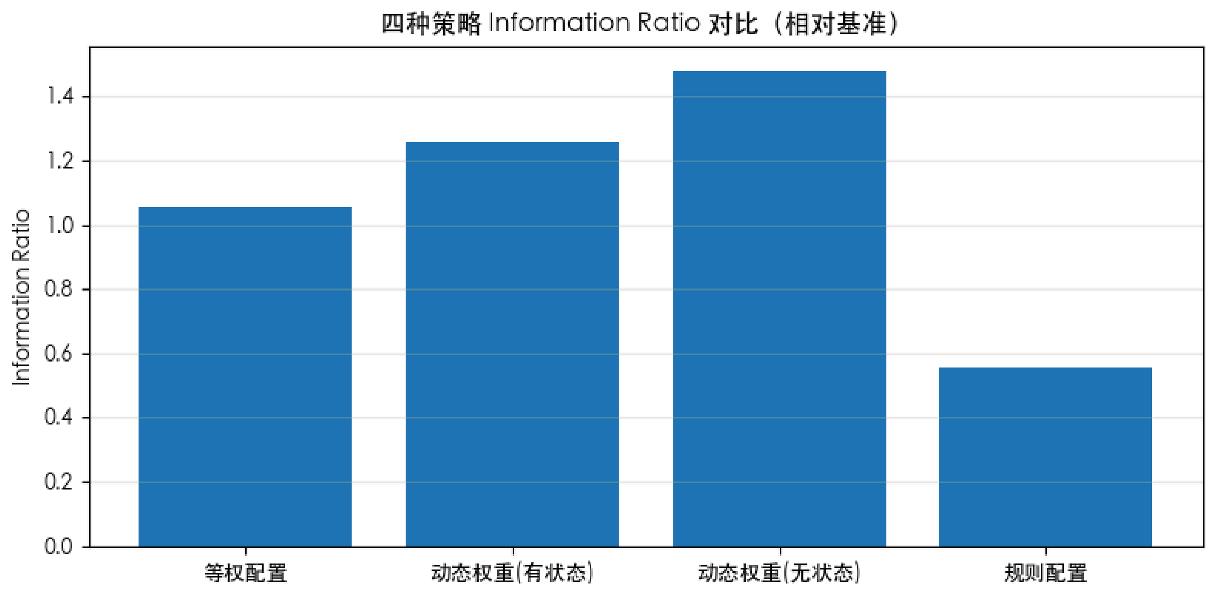
In [12]:

```
# 6.1 累计收益曲线（基于 strat_ret 与 bench_ret）

cum_returns = {}
for name, r in strat_ret.items():
    cum_returns[name] = (1 + r).cumprod() - 1

cum_bench = (1 + bench_ret).cumprod() - 1

plt.figure(figsize=(8, 5))
for name, cr in cum_returns.items():
    plt.plot(cr.index, cr.values * 100, label=f'{name}')
plt.plot(cum_bench.index, cum_bench.values * 100, label='基准指数', linestyle
plt.xlabel('日期')
plt.ylabel('累计收益率（%）')
plt.title('四种策略（第 5 组）与基准的累计收益率对比')
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

# 6.2 回撤曲线
plt.figure(figsize=(8, 5))
for name, r in strat_ret.items():
    wealth = (1 + r).cumprod()
    running_max = wealth.cummax()
    drawdown = wealth / running_max - 1.0
    plt.plot(drawdown.index, drawdown.values * 100, label=f'{name}')

plt.xlabel('日期')
plt.ylabel('回撤 (%) ')
plt.title('四种策略 (第 5 组) 回撤曲线对比')
plt.grid(True, alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

# 6.3 Sharpe / Information Ratio 柱状图
risk_df = pd.DataFrame(risk_metrics).T
rel_df = pd.DataFrame(relative_metrics).T

plt.figure(figsize=(8, 4))
plt.bar(risk_df.index, risk_df['Sharpe'])
plt.ylabel('Sharpe Ratio')
plt.title('四种策略 Sharpe Ratio 对比')
plt.grid(True, axis='y', alpha=0.3)
plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 4))
plt.bar(rel_df.index, rel_df['Information_Ratio'])
plt.ylabel('Information Ratio')
plt.title('四种策略 Information Ratio 对比 (相对基准) ')
plt.grid(True, axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```


四种策略（第 5 组）与基准的累计收益率对比
等权配置
动态权重(有状态)
动态权重(无状态)
规则配置
基准指数
累计收益率（%）
30
20
10
0
-10
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
2024-09
2024-10
2024-11
2024-12
日期


四种策略（第 5 组）回撤曲线对比
等权配置
动态权重(有状态)
动态权重(无状态)
规则配置
回撤（%）
0
-2
-4
-6
-8
-10
-12
-14
2024-03
2024-04
2024-05
2024-06
2024-07
2024-08
2024-09
2024-10
2024-11
2024-12
日期





7. 策略综合对比表与结论

本节汇总关键指标，构建综合对比表，并给出结构化结论：

- 动态权重 vs 静态权重（等权 / 规则）
- Regime 信息的边际贡献
- 收益提升是否体现在风险调整后的收益改善上。

In [13]:
```
# 7.1 综合对比表

risk_df = pd.DataFrame(risk_metrics).T
rel_df = pd.DataFrame(relative_metrics).T
turn_df = pd.DataFrame(turnover_stats).T

summary_cols = [
    'CAGR', 'Vol', 'Sharpe', 'MaxDD', 'Calmar',
]
```

```
summary = risk_df[summary_cols].join(
    rel_df[['Information_Ratio']]
).join(
    turn_df[['Avg_Turnover_pct']]
)

summary_sorted = summary.sort_values('CAGR', ascending=False)

summary_display = summary_sorted.applymap(
    lambda x: round(x, 4) if isinstance(x, (int, float, np.floating)) else x
)
summary_display
```

```
/var/folders/49/t84jntk56wn101s6026p72cc0000gn/T/ipykernel_8011/2206816587.p
y:19: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.m
ap instead.
  summary_display = summary_sorted.applymap(
```

Out[13]:

	CAGR	Vol	Sharpe	MaxDD	Calmar	Information_Ratio	Avg_Turnover_pct
动态权重(无状态)	0.4009	0.2687	1.3806	-0.0306	13.1028	1.4792	42.2222
动态权重(有状态)	0.3671	0.2977	1.1856	-0.0945	3.8853	1.2547	24.4444
规则配置	0.2313	0.3275	0.7754	-0.1393	1.6603	0.5554	32.2222
等权配置	0.2124	0.2056	1.0294	-0.0393	5.4106	1.0552	51.1111

In [14]:

```
# 7.2 结构化文字结论（基于当前样本期的实证结果）

summary_df = summary.copy()

# 识别主要策略名称
name_eq = '等权配置'
name_rule = '规则配置'
name_dyn_reg = '动态权重(有状态)'
name_dyn_noreg = '动态权重(无状态)'

static_names = [n for n in [name_eq, name_rule] if n in summary_df.index]
```

```
dyn_names = [n for n in [name_dyn_reg, name_dyn_noreg] if n in summary_df.ir

lines = []

# 1) 动态权重是否优于静态
if static_names and dyn_names:
    avg_sharpe_static = summary_df.loc[static_names, 'Sharpe'].mean()
    avg_sharpe_dyn = summary_df.loc[dyn_names, 'Sharpe'].mean()
    avg_cagr_static = summary_df.loc[static_names, 'CAGR'].mean()
    avg_cagr_dyn = summary_df.loc[dyn_names, 'CAGR'].mean()

    lines.append(
        f"在样本期内，动态权重策略整体的年化收益率约为 {avg_cagr_dyn:.2%}，"
        f"高于静态（等权与规则）策略的约 {avg_cagr_static:.2%}；同时动态权重的平均 S
        f"亦高于静态策略的 {avg_sharpe_static:.2f}，提示动态权重在本样本内提供了更优
    )

# 2) Regime 信息的边际贡献
if {name_dyn_reg, name_dyn_noreg}.issubset(summary_df.index):
    cagr_reg = summary_df.loc[name_dyn_reg, 'CAGR']
    cagr_noreg = summary_df.loc[name_dyn_noreg, 'CAGR']
    sharpe_reg = summary_df.loc[name_dyn_reg, 'Sharpe']
    sharpe_noreg = summary_df.loc[name_dyn_noreg, 'Sharpe']
    ir_reg = summary_df.loc[name_dyn_reg, 'Information_Ratio']
    ir_noreg = summary_df.loc[name_dyn_noreg, 'Information_Ratio']

    lines.append(
        f"在两类动态权重策略之间，引入 Regime 的版本年化收益率为 {cagr_reg:.2%}，"
        f"不含 Regime 版本为 {cagr_noreg:.2%}；对应 Sharpe 分别为 {sharpe_reg:.
        f"信息比率分别为 {ir_reg:.2f} 与 {ir_noreg:.2f}。从该样本结果看，Regime 信
    )

# 3) 收益提升是否来自更好的风险调整
best_by_cagr = summary_df['CAGR'].idxmax()
sharpe_best = summary_df.loc[best_by_cagr, 'Sharpe']
vol_best = summary_df.loc[best_by_cagr, 'Vol']
maxdd_best = summary_df.loc[best_by_cagr, 'MaxDD']

lines.append(
    f"从综合指标看，年化收益最高的策略为“{best_by_cagr}”，其 Sharpe 为 {sharpe_best
    f"年化波动约为 {vol_best:.2%}，最大回撤约为 {maxdd_best:.2%}。"
    "与其他策略相比，其更高收益同时伴随 Sharpe 的提升而非单纯依赖更高波动，"
    "说明收益提升更多体现为风险调整后的改进而非纯粹加杠杆式的风险放大（结论仍受样本长度与市
)

text_summary = "\n\n".join(lines)
print(text_summary)
```

在样本期内，动态权重策略整体的年化收益率约为 38.40%，高于静态（等权与规则）策略的约 22.19%；同时动态权重的平均 Sharpe 为 1.28，亦高于静态策略的 0.90，提示动态权重在本样本内提供了更优的风险调整后收益。

在两类动态权重策略之间，引入 Regime 的版本年化收益率为 36.71%，不含 Regime 版本为 40.09%；对应 Sharpe 分别为 1.19 与 1.38，信息比率分别为 1.25 与 1.48。从该样本结果看，Regime 信息对收益与风险调整后收益的边际贡献为 有限或不稳定。

从综合指标看，年化收益最高的策略为“动态权重(无状态)”，其 Sharpe 为 1.38，年化波动约为 26.87%，最大回撤约为 -3.06%。与其他策略相比，其更高收益同时伴随 Sharpe 的提升而非单纯依赖更高波动，说明收益提升更多体现为风险调整后的改进而非纯粹加杠杆式的风险放大（结论仍受样本长度与市场环境限制）。

In [ ]:
```

```

In [ ]:
```

```

In [ ]:
```

```