Анализ временных рядов

Данный notebook содержит анализ влияния стационарности и стратегий прогнозирования на качество предсказаний цен акций Apple (AAPL) за период 2010-2020.

Гипотеза 1: Приведение данных к стационарным улучшает прогноз

Гипотеза 2: Периодическое переобучеие даёт лучшие результаты, чем прямое прогнозирование.

Устанавливаем библиотеку yfinance для загрузки финансовых данных

In [1]:
```
import sys
!{sys.executable} -m pip install yfinance
```

```
Requirement already satisfied: yfinance in /opt/anaconda3/lib/python
3.13/site-packages (0.2.66)
Requirement already satisfied: pandas>=1.3.0 in /opt/anaconda3/lib/p
ython3.13/site-packages (from yfinance) (2.2.3)
Requirement already satisfied: numpy>=1.16.5 in /opt/anaconda3/lib/p
ython3.13/site-packages (from yfinance) (2.1.3)
Requirement already satisfied: requests>=2.31 in /opt/anaconda3/lib/
python3.13/site-packages (from yfinance) (2.32.3)
Requirement already satisfied: multitasking>=0.0.7 in /opt/anaconda
3/lib/python3.13/site-packages (from yfinance) (0.0.12)
Requirement already satisfied: platformdirs>=2.0.0 in /opt/anaconda
3/lib/python3.13/site-packages (from yfinance) (4.3.7)
Requirement already satisfied: pytz>=2022.5 in /opt/anaconda3/lib/py
thon3.13/site-packages (from yfinance) (2024.1)
Requirement already satisfied: frozendict>=2.3.4 in /opt/anaconda3/l
ib/python3.13/site-packages (from yfinance) (2.4.2)
Requirement already satisfied: peewee>=3.16.2 in /opt/anaconda3/lib/
python3.13/site-packages (from yfinance) (3.18.3)
Requirement already satisfied: beautifulsoup4>=4.11.1 in /opt/anacon
da3/lib/python3.13/site-packages (from yfinance) (4.12.3)
Requirement already satisfied: curl_cffi>=0.7 in /opt/anaconda3/lib/
python3.13/site-packages (from yfinance) (0.13.0)
Requirement already satisfied: protobuf>=3.19.0 in /opt/anaconda3/li
b/python3.13/site-packages (from yfinance) (5.29.3)
Requirement already satisfied: websockets>=13.0 in /opt/anaconda3/li
b/python3.13/site-packages (from yfinance) (15.0.1)
Requirement already satisfied: soupsieve>1.2 in /opt/anaconda3/lib/p
ython3.13/site-packages (from beautifulsoup4>=4.11.1->yfinance) (2.
5)
Requirement already satisfied: cffi>=1.12.0 in /opt/anaconda3/lib/py
thon3.13/site-packages (from curl_cffi>=0.7->yfinance) (1.17.1)
Requirement already satisfied: certifi>=2024.2.2 in /opt/anaconda3/l
ib/python3.13/site-packages (from curl_cffi>=0.7->yfinance) (2025.4.
26)
Requirement already satisfied: pycparser in /opt/anaconda3/lib/pytho
n3.13/site-packages (from cffi>=1.12.0->curl_cffi>=0.7->yfinance) (
2.21)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anacon
da3/lib/python3.13/site-packages (from pandas>=1.3.0->yfinance) (2.
9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in /opt/anaconda3/lib/
python3.13/site-packages (from pandas>=1.3.0->yfinance) (2025.2)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/python
3.13/site-packages (from python-dateutil>=2.8.2->pandas>=1.3.0->yfin
ance) (1.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/anac
onda3/lib/python3.13/site-packages (from requests>=2.31->yfinance) (
3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/py
thon3.13/site-packages (from requests>=2.31->yfinance) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/anaconda3/
lib/python3.13/site-packages (from requests>=2.31->yfinance) (2.3.0)
```

Импортируем все необходимые библиотеки для анализа временных рядов

In [2]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from datetime import datetime, timedelta
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.tsa.holtwinters import SimpleExpSmoothing, Exponen
from sklearn.metrics import mean_absolute_error, mean_squared_error
import time
import warnings
warnings.filterwarnings('ignore')

print("Библиотеки загружены успешно")
```

```
Библиотеки загружены успешно
```

Загружаем исторические данные цен акций Apple за 10 лет

In [3]:
```
ticker_symbol = "AAPL"
data = yf.download(ticker_symbol, start="2010-01-01", end="2020-01-
print(f"Данные загружены: {len(data)} наблюдений")
```

```
[**********************100%**********************]  1 of 1 completed
Данные загружены: 2516 наблюдений
```

Извлекаем цены закрытия, заполянем пропуски методом forward fill

In [4]:
```
prices = data['Close'].ffill()
prices.name = 'prices'

print(f"Загружено {len(prices)} наблюдений")
print(f"Период: с {prices.index[0].date()} по {prices.index[-1].dat
print(f"Пропусков в данных: {prices.isna().sum()}")
```

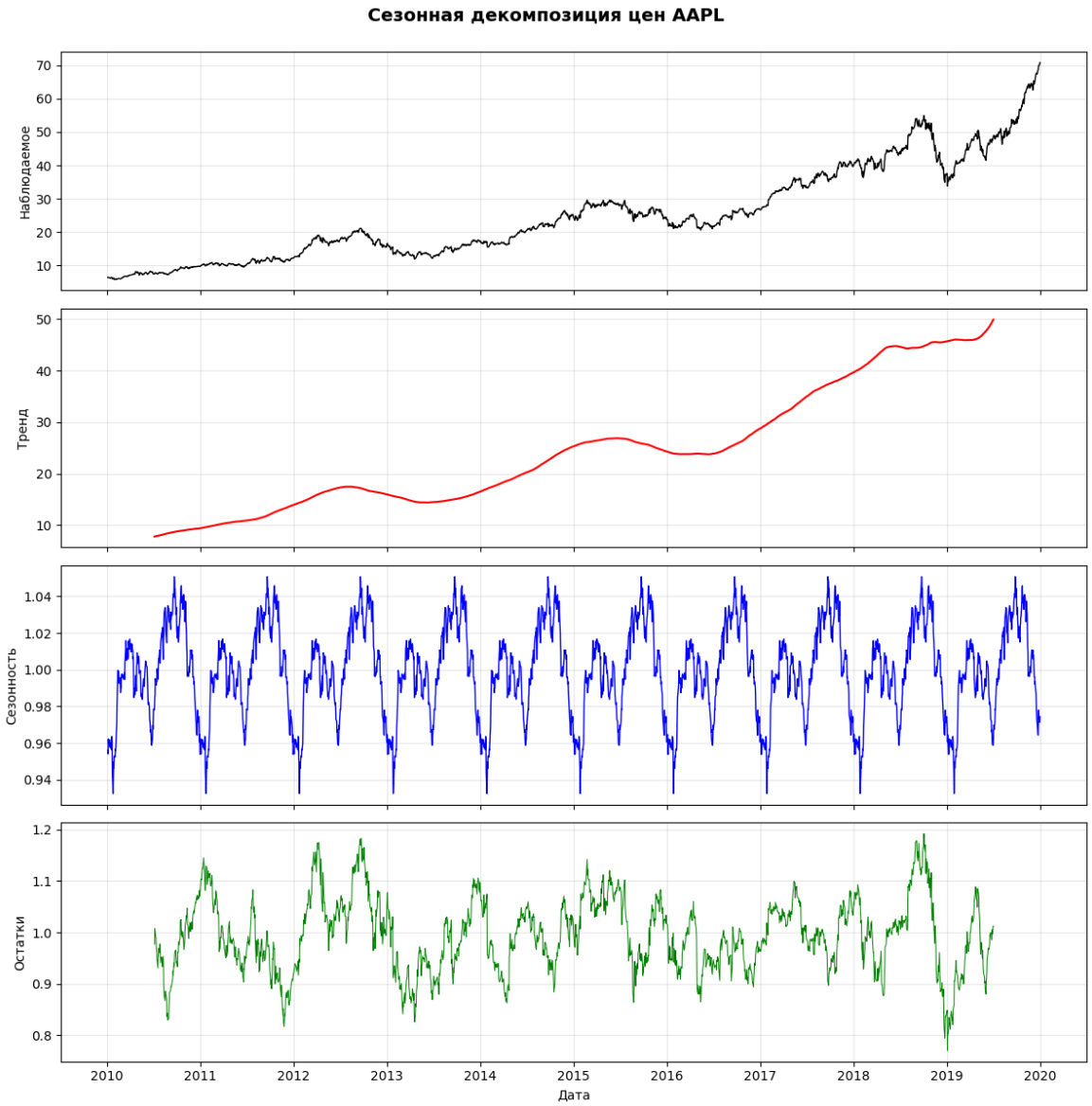
```
Загружено 2516 наблюдений
Период: с 2010-01-04 по 2019-12-31
Пропусков в данных: Ticker
AAPL    0
dtype: int64
```

Строим график цен закртия акций во времени

In [5]:
```
plt.figure(figsize=(12, 6))
plt.plot(prices.index, prices, linewidth=1, color='blue')
plt.title(f"{ticker_symbol}: Цены закрытия (2010-2020)", fontsize=1
plt.xlabel("Дата", fontsize=12)
plt.ylabel("Цена закрытия (USD)", fontsize=12)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Разбиваем ряд на тренд, сезонность и остатки (мультипликативная модель)

In [6]:
```
result = seasonal_decompose(prices, model='multiplicative', period=

fig, axes = plt.subplots(4, 1, figsize=(12, 12), sharex=True)

axes[0].plot(prices.index, result.observed, color='black', linewidt
axes[0].set_ylabel('Наблюдаемое')
axes[0].grid(True, alpha=0.3)

axes[1].plot(prices.index, result.trend, color='red', linewidth=1.5
axes[1].set_ylabel('Тренд')
axes[1].grid(True, alpha=0.3)

axes[2].plot(prices.index, result.seasonal, color='blue', linewidth
axes[2].set_ylabel('Сезонность')
axes[2].grid(True, alpha=0.3)

axes[3].plot(prices.index, result.resid, color='green', linewidth=0
axes[3].set_ylabel('Остатки')
axes[3].set_xlabel('Дата')
axes[3].grid(True, alpha=0.3)

plt.suptitle(f'Сезонная декомпозиция цен {ticker_symbol}', fontsize
plt.tight_layout()
plt.show()
```



Реализуем функцию для проверки стационарности с помощью ADF и KPSS тестов

In [7]:
```
def check_stationary(timeseries):

    timeseries_name = timeseries.name if timeseries.name else 'Seri
    timeseries = timeseries.dropna()

    # ADF тест (H0: ряд нестационарен)
    adf_result = adfuller(timeseries, autolag='AIC')
    adf_pvalue = adf_result[1]

    print('Результаты теста Дики-Фуллера (ADF):')
    print(f'  Статистика теста: {adf_result[0]:.4f}')
    print(f'  p-значение: {adf_pvalue:.4f}')

    if adf_pvalue < 0.05:
        print(f'Ряд {timeseries_name} СТАЦИОНАРЕН (отвергаем H0 при
    else:
        print(f'Ряд {timeseries_name} НЕСТАЦИОНАРЕН (не отвергаем H

    # KPSS тест (H0: ряд стационарен)
```

```
    kpss_result = kpss(timeseries, regression='c', nlags="auto")
    kpss_pvalue = kpss_result[1]

    print('\nРезультаты теста KPSS:')
    print(f'  Статистика теста: {kpss_result[0]:.4f}')
    print(f'  p-значение: {kpss_pvalue:.4f}')

    if kpss_pvalue < 0.05:
        print(f'Ряд {timeseries_name} НЕСТАЦИОНАРЕН (отвергаем H0 п
    else:
        print(f'Ряд {timeseries_name} СТАЦИОНАРЕН (не отвергаем H0
```

Применим её к исходному ряду

In [8]:
```
print("Проверка стационарности исходных цен:")

check_stationary(prices)
```

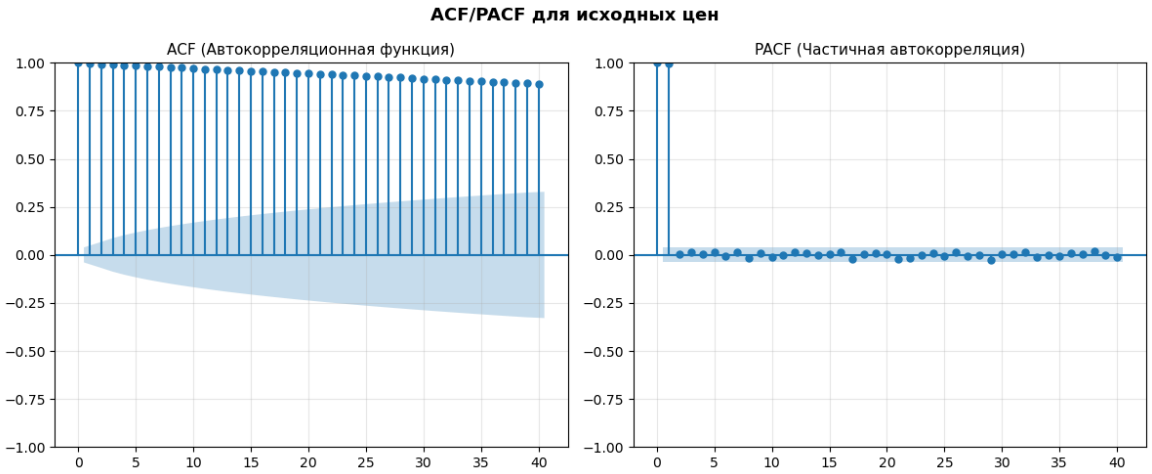
```
Проверка стационарности исходных цен:
Результаты теста Дики-Фуллера (ADF):
  Статистика теста: 1.4931
  p-значение: 0.9975
Ряд prices НЕСТАЦИОНАРЕН (не отвергаем H0 при alpha=0.05)

Результаты теста KPSS:
  Статистика теста: 7.2505
  p-значение: 0.0100
Ряд prices НЕСТАЦИОНАРЕН (отвергаем H0 при alpha=0.05)
```

Создадим функцию, которая остроит графики автокрреляции и частичной автокорреляционной функции

In [9]:
```
def plot_acf_pacf(timeseries, title="ACF/PACF", lags=40):

    clean_series = timeseries.dropna()

    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

    plot_acf(clean_series, ax=axes[0], lags=lags)
    axes[0].set_title('ACF (Автокорреляционная функция)', fontsize=
    axes[0].grid(True, alpha=0.3)

    plot_pacf(clean_series, ax=axes[1], lags=lags, method='ywm')
    axes[1].set_title('PACF (Частичная автокорреляция)', fontsize=1
    axes[1].grid(True, alpha=0.3)

    plt.suptitle(title, fontsize=13, fontweight='bold')
    plt.tight_layout()
    plt.show()
```

Строим графики ACF и PACF для исходных цен

In [10]:
```
plot_acf_pacf(prices, title='ACF/PACF для исходных цен', lags=40)
```



Применяем логарифмирование и первую разность для получения стационарного ряда

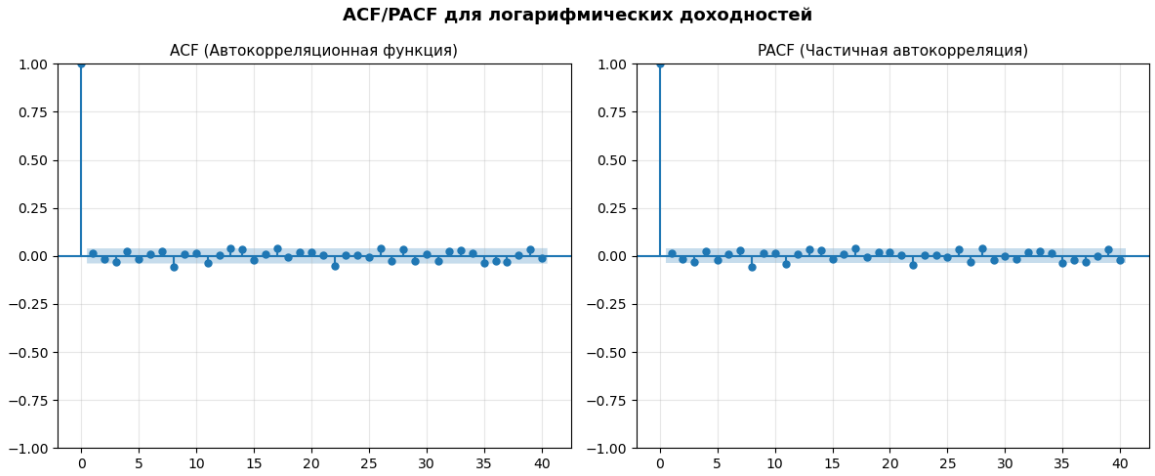
In [11]:
```
# Логарифмирование делается стабилизации волатильности
# Первая разность логарифмов = логарифмическая доходность
log_returns = np.log(prices).diff().dropna()
log_returns.name = 'log_returns'

print(f"Исходный ряд: {len(prices)} наблюдений")

print(f"Логарифмические доходности: {len(log_returns)} наблюдений")
```

```
Исходный ряд: 2516 наблюдений
Логарифмические доходности: 2515 наблюдений
```

Проверим стационарность ряда лог-доходностей, а также построим для него ACF и PACF

In [12]:
```
print("Проверка стационарности логарифмов цен:")
check_stationary(log_returns)
plot_acf_pacf(log_returns, title="ACF/PACF для логарифмических дохо
```

```
Проверка стационарности логарифмов цен:
Результаты теста Дики-Фуллера (ADF):
  Статистика теста: -12.8523
  p-значение: 0.0000
Ряд log_returns СТАЦИОНАРЕН (отвергаем H0 при alpha=0.05)

Результаты теста KPSS:
  Статистика теста: 0.0635
  p-значение: 0.1000
Ряд log_returns СТАЦИОНАРЕН (не отвергаем H0 при alpha=0.05)
```



Визуализируем наш стационарный ряд

In [13]:
```
plt.figure(figsize=(12, 6))
plt.plot(log_returns.index, log_returns, label='log_returns', color
plt.title('Логарифмические доходности (стационарный ряд)', fontsize
plt.xlabel("Дата", fontsize=11)
plt.ylabel("Логарифмическая доходность", fontsize=11)
plt.axhline(y=0, color='black', linestyle='--', alpha=0.5)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Разделим данные на train и test

Первые 80% возьмём для обучения, оставшиеся 20% для тестирования и расчёта метрик

In [14]:
```
# Нестационарные данные (исходные цены)
train_size_nonstationary = int(len(prices) * 0.8)
train_nonstationary = prices.iloc[:train_size_nonstationary]
test_nonstationary = prices.iloc[train_size_nonstationary:]

# Стационарные данные (логарифмические доходности)
train_size_stationary = int(len(log_returns) * 0.8)
```

```
train_stationary = log_returns.iloc[:train_size_stationary]
test_stationary = log_returns.iloc[train_size_stationary:]

# Для корректного сравнения убираем первое наблюдение из test_nonst
test_nonstationary_aligned = test_nonstationary.iloc[1:]

print(f"Нестационарные данные:")
print(f"  Train: {len(train_nonstationary)} наблюдений")
print(f"  Test: {len(test_nonstationary)} наблюдений")
print(f"  Test (aligned): {len(test_nonstationary_aligned)} наблюде

print(f"\nСтационарные данные:")
print(f"  Train: {len(train_stationary)} наблюдений")
print(f"  Test: {len(test_stationary)} наблюдений")
```

```
Нестационарные данные:
  Train: 2012 наблюдений
  Test: 504 наблюдений
  Test (aligned): 503 наблюдений

Стационарные данные:
  Train: 2012 наблюдений
  Test: 503 наблюдений
```

Визуализируем полученное разбиение

In [15]:
```
plt.figure(figsize=(12, 6))
plt.plot(train_nonstationary.index, train_nonstationary, label='Tra
plt.plot(test_nonstationary.index, test_nonstationary, label='Test'
plt.axvline(train_nonstationary.index[-1], color='red', linestyle='
plt.title('Разделение данных на обучающую и тестовую выборки', font
plt.xlabel("Дата")
plt.ylabel("Цена закрытия (USD)")
plt.legend(fontsize=10)
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

Гипотеза 1: Влияние стационарности на

качество прогноза

Baseline модель (naive forecast)

In [16]:
```
naive_value = float(train_nonstationary.iloc[-1])
naive_pred = pd.Series([naive_value] * len(test_nonstationary),
                       index=test_nonstationary.index)
naive_pred.name = 'Naive'

print(f"Naive прогноз: все {len(naive_pred)} значений равны {naive_
```

```
Naive прогноз: все 504 значений равны 40.07 USD
```

Обучаем модель SES на нестационарных и стационарных данных

In [17]:
```
# SES на нестационарных данных
ses_model_nonstat = SimpleExpSmoothing(train_nonstationary).fit(opt
ses_pred_nonstat = ses_model_nonstat.forecast(len(test_nonstationar
ses_pred_nonstat.name = 'SES nonstat'

print(f"SES (нестационарные): alpha = {ses_model_nonstat.params['sm

# SES на стационарных данных
ses_model_stat = SimpleExpSmoothing(train_stationary).fit(optimized
ses_pred_stat = ses_model_stat.forecast(len(test_stationary))
ses_pred_stat.name = 'SES stat'

print(f"SES (стационарные): alpha = {ses_model_stat.params['smoothi
```

```
SES (нестационарные): alpha = 1.0000
SES (стационарные): alpha = 0.0010
```

Обучаем модель Holt с учетом линейного тренда

In [18]:
```
# Holt на нестационарных данных
holt_model_nonstat = ExponentialSmoothing(train_nonstationary, tren
holt_pred_nonstat = holt_model_nonstat.forecast(len(test_nonstation
holt_pred_nonstat.name = 'Holt nonstat'

print(f"Holt (нестационарные): alpha = {holt_model_nonstat.params['
      f"beta = {holt_model_nonstat.params['smoothing_trend']:.4f}")

# Holt на стационарных данных
holt_model_stat = ExponentialSmoothing(train_stationary, trend='add
holt_pred_stat = holt_model_stat.forecast(len(test_stationary))
holt_pred_stat.name = 'Holt stat'

print(f"Holt (стационарные): alpha = {holt_model_stat.params['smoot
      f"beta = {holt_model_stat.params['smoothing_trend']:.4f}")
```

```
Holt (нестационарные): alpha = 1.0000, beta = 0.0000
Holt (стационарные): alpha = 0.0000, beta = 0.0000
```

Обучаем модель Holt-Winters

In [19]:
```
# Holt-Winters на нестационарных данных
hw_model_nonstat = ExponentialSmoothing(train_nonstationary, trend=
hw_pred_nonstat = hw_model_nonstat.forecast(len(test_nonstationary)
hw_pred_nonstat.name = 'Holt-Winters nonstat'

print(f"Holt-Winters (нестационарные): alpha = {hw_model_nonstat.pa

# Holt-Winters на стационарных данных
hw_model_stat = ExponentialSmoothing(train_stationary).fit(optimize
hw_pred_stat = hw_model_stat.forecast(len(test_stationary))
hw_pred_stat.name = 'Holt-Winters stat'

print(f"Holt-Winters (стационарные): alpha = {hw_model_stat.params[
```

```
Holt-Winters (нестационарные): alpha = 1.0000
Holt-Winters (стационарные): alpha = 0.0000
```

Зададим функцию восстановления цен из лог-доходностей

In [20]:
```
def restore_prices_from_log_returns(log_returns_series, initial_pri

    # Извлекаем скалярное значение если передан Series
    if isinstance(initial_price, (pd.Series, pd.DataFrame)):
        initial_price_value = float(initial_price.iloc[0])
    else:
        initial_price_value = float(initial_price)

    # Заполняем NaN нулями и вычисляем кумулятивную сумму
    log_returns_clean = log_returns_series.fillna(0)
    accumulated_log_returns = log_returns_clean.cumsum()

    # Восстанавливаем цены: P_t = P_0 * exp(sum(r_i))
    restored_prices = initial_price_value * np.exp(accumulated_log_

    return restored_prices
```

Применяем эту функцию для восстановления цен всех моделей, обученных на стационарных данных

In [21]:
```
# Восстанавливаем цены из прогнозов логарифмических доходностей
restore_prices_ses = restore_prices_from_log_returns(ses_pred_stat,
restore_prices_ses.name = 'SES Stationary'

restore_prices_holt = restore_prices_from_log_returns(holt_pred_sta
restore_prices_holt.name = 'Holt Stationary'

restore_prices_hw = restore_prices_from_log_returns(hw_pred_stat, t
restore_prices_hw.name = 'Holt-Winters Stationary'

# Проверка корректности
print("Проверка восстановленных прогнозов:")
print(f"SES: {restore_prices_ses.isna().sum()} NaN, длина: {len(res
      f"диапазон: [{restore_prices_ses.min():.2f}, {restore_prices_
print(f"Holt: {restore_prices_holt.isna().sum()} NaN, длина: {len(r
      f"диапазон: [{restore_prices_holt.min():.2f}, {restore_prices
```

```
print(f"HW: {restore_prices_hw.isna().sum()} NaN, длина: {len(resto
      f"диапазон: [{restore_prices_hw.min():.2f}, {restore_prices_h
```

```
Проверка восстановленных прогнозов:
SES: 0 NaN, длина: 503, диапазон: [40.10, 65.59]
Holt: 0 NaN, длина: 503, диапазон: [40.09, 52.51]
HW: 0 NaN, длина: 503, диапазон: [40.10, 63.16]
```

Приведём все прогнозы к единым индексам (из-за разной длины индексы разные)

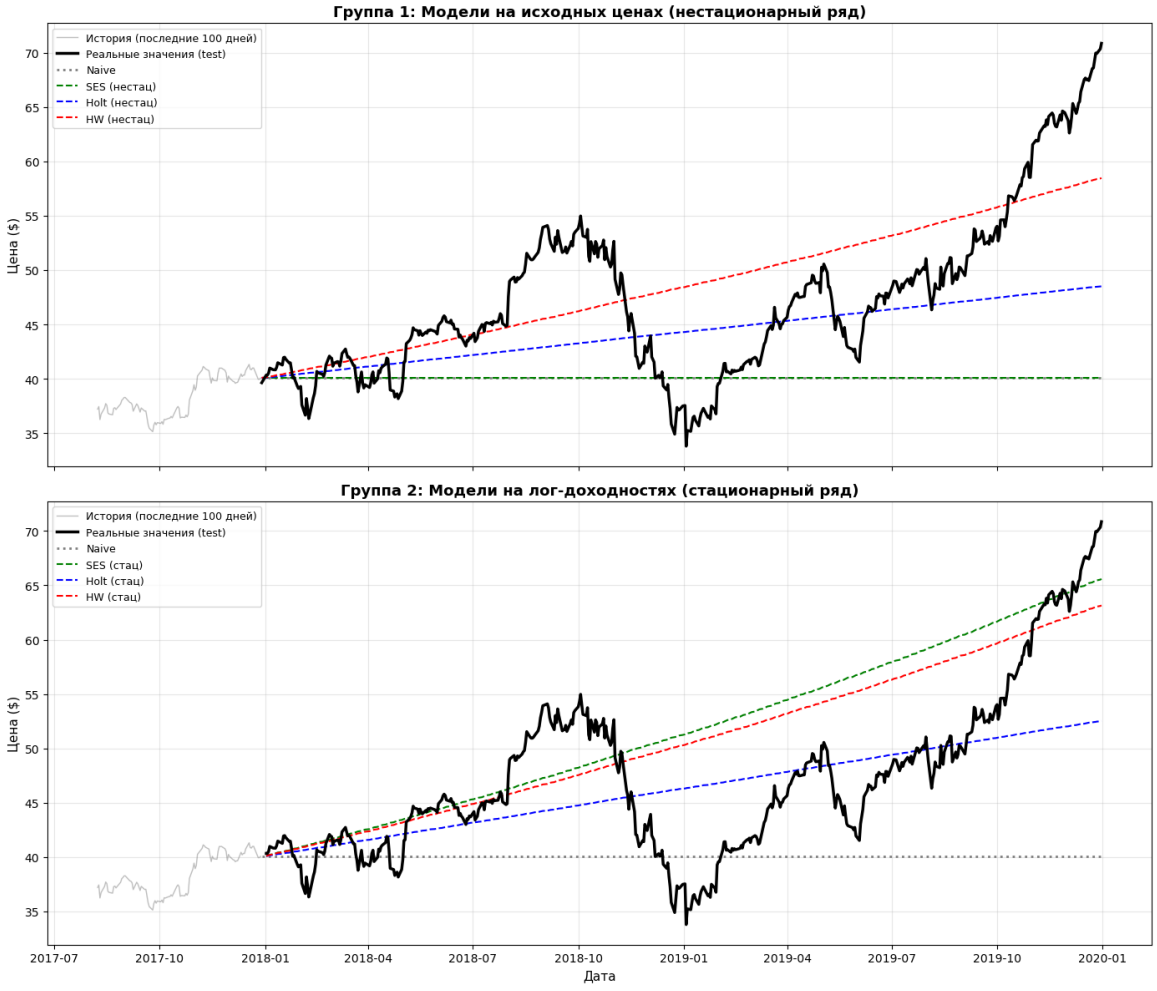
In [22]:
```
target_index = test_stationary.index

# Нестационарные прогнозы
naive_pred.index = test_nonstationary.index
ses_pred_nonstat.index = test_nonstationary.index
holt_pred_nonstat.index = test_nonstationary.index
hw_pred_nonstat.index = test_nonstationary.index

# Стационарные прогнозы (восстановленные)
restore_prices_ses.index = target_index
restore_prices_holt.index = target_index
restore_prices_hw.index = target_index

# Реальные значения для сравнения
test_nonstationary_aligned = test_nonstationary.iloc[1:]

print(f"Все прогнозы приведены к единому индексу: {len(target_index
```

```
Все прогнозы приведены к единому индексу: 503 наблюдений
```

Визуально сравним прогнозы на нестационарных и на стаицонарных данных

In [23]:
```
fig, axes = plt.subplots(2, 1, figsize=(14, 12), sharex=True)

# График 1: Модели на нестационарных данных
axes[0].plot(train_nonstationary.index[-100:], train_nonstationary.
             color='gray', alpha=0.5, label='История (последние 100
axes[0].plot(test_nonstationary.index, test_nonstationary,
             color='black', linewidth=2.5, label='Реальные значения

axes[0].plot(naive_pred.index, naive_pred,
             label='Naive', linestyle=':', color='gray', linewidth=
axes[0].plot(ses_pred_nonstat.index, ses_pred_nonstat,
             label='SES (нестац)', linestyle='--', color='green', l
axes[0].plot(holt_pred_nonstat.index, holt_pred_nonstat,
             label='Holt (нестац)', linestyle='--', color='blue', l
axes[0].plot(hw_pred_nonstat.index, hw_pred_nonstat,
             label='HW (нестац)', linestyle='--', color='red', line

axes[0].set_title('Группа 1: Модели на исходных ценах (нестационарн
                  fontsize=13, fontweight='bold')
axes[0].legend(loc='upper left', fontsize=9)
axes[0].grid(True, alpha=0.3)
axes[0].set_ylabel('Цена ($)', fontsize=11)
```

```
# График 2: Модели на стационарных данных
axes[1].plot(train_nonstationary.index[-100:], train_nonstationary.
             color='gray', alpha=0.5, label='История (последние 100
axes[1].plot(test_nonstationary_aligned.index, test_nonstationary_a
             color='black', linewidth=2.5, label='Реальные значения

axes[1].plot(naive_pred.index, naive_pred,
             label='Naive', linestyle=':', color='gray', linewidth=
axes[1].plot(restore_prices_ses.index, restore_prices_ses,
             label='SES (стац)', linestyle='--', color='green', lin
axes[1].plot(restore_prices_holt.index, restore_prices_holt,
             label='Holt (стац)', linestyle='--', color='blue', lin
axes[1].plot(restore_prices_hw.index, restore_prices_hw,
             label='HW (стац)', linestyle='--', color='red', linewi

axes[1].set_title('Группа 2: Модели на лог-доходностях (стационарны
                  fontsize=13, fontweight='bold')
axes[1].legend(loc='upper left', fontsize=9)
axes[1].grid(True, alpha=0.3)
axes[1].set_xlabel('Дата', fontsize=11)
axes[1].set_ylabel('Цена ($)', fontsize=11)

plt.tight_layout()
plt.show()
```



Запишем функцию, которая позволит рассчитывать метрики качества

Метрику DA решено не считать, т.к. модели по сути ловят только тренд (так мы задали их параметры) поэтому у всех прогнозом будет то, что цена все 503 дня прогнозов растёт, а значит считать метрику, которая бы показывала как точно угадываются движения рынка особого смысла нет (у всех моделей DA будет одинаковой, кроме разве что SES (non-stat))

In [24]:
```
def calculate_metrics(y_true, y_pred, model_name):
    y_true_values = np.asarray(y_true).flatten()
    y_pred_values = np.asarray(y_pred).flatten()

    mae = np.mean(np.abs(y_true_values - y_pred_values))
    rmse = np.sqrt(np.mean((y_true_values - y_pred_values) ** 2))
    mape = np.mean(np.abs((y_true_values - y_pred_values) / y_true_

    return {
        'Model': model_name,
        'MAE': mae,
        'RMSE': rmse,
        'MAPE': mape
    }
```

Теперь рассчитаем метрики для всех наших моделей, запишем их в таблицу

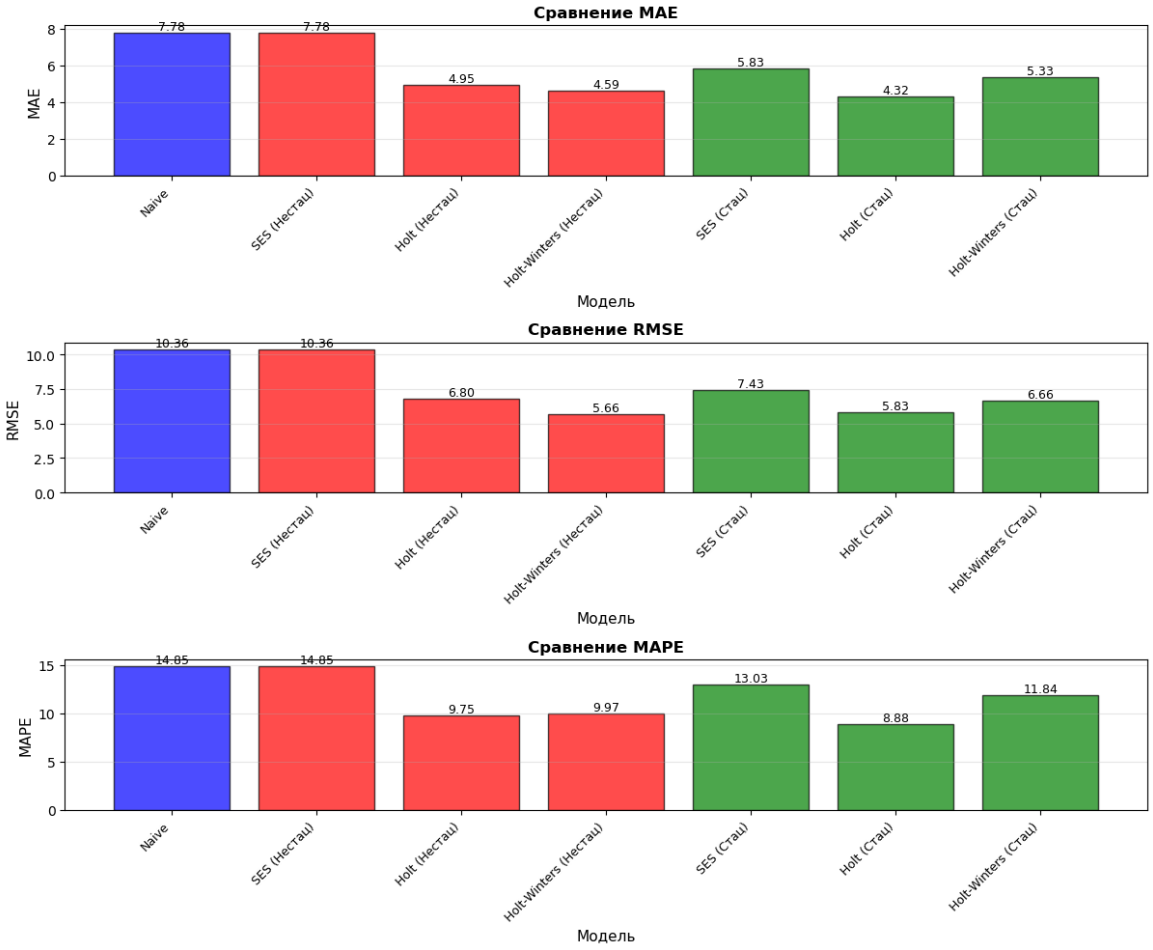
In [25]:
```
results = []

# Нестационарные модели
results.append(calculate_metrics(test_nonstationary, naive_pred, 'N
results.append(calculate_metrics(test_nonstationary, ses_pred_nonst
results.append(calculate_metrics(test_nonstationary, holt_pred_nons
results.append(calculate_metrics(test_nonstationary, hw_pred_nonsta

# Стационарные модели (восстановленные)
results.append(calculate_metrics(test_nonstationary_aligned, restor
results.append(calculate_metrics(test_nonstationary_aligned, restor
results.append(calculate_metrics(test_nonstationary_aligned, restor

# Создаем таблицу результатов
results_df = pd.DataFrame(results)

print("\n" + "=" * 70)
print("ТАБЛИЦА МЕТРИК КАЧЕСТВА ПРОГНОЗА (ГИПОТЕЗА 1)")
print("=" * 70)
print(results_df.to_string(index=False))
print("=" * 70)
```

```
================================================================
==
ТАБЛИЦА МЕТРИК КАЧЕСТВА ПРОГНОЗА (ГИПОТЕЗА 1)
================================================================
==
                Model      MAE      RMSE      MAPE
                Naive 7.778209 10.362531 14.852147
         SES (Нестац) 7.778209 10.362531 14.852147
        Holt (Нестац) 4.952924  6.801911  9.749431
Holt-Winters (Нестац) 4.591515  5.662002  9.968676
           SES (Стац) 5.833016  7.431664 13.026864
          Holt (Стац) 4.316301  5.829222  8.883631
  Holt-Winters (Стац) 5.325432  6.659052 11.836583
================================================================
==
```

Визуализируем полученные данные, для лучшего их понимания

In [26]:

```
fig, axes = plt.subplots(3, 1, figsize=(12, 10))

metrics = ['MAE', 'RMSE', 'MAPE']
colors = ['blue', 'red', 'red', 'red', 'green', 'green', 'green']

for idx, metric in enumerate(metrics):
    ax = axes[idx]
    x = np.arange(len(results_df))
    bars = ax.bar(x, results_df[metric], color=colors, alpha=0.7, e

    # Добавляем значения над столбцами
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height,
                f'{height:.2f}',
                ha='center', va='bottom', fontsize=9)

    ax.set_xlabel('Модель', fontsize=11)
    ax.set_ylabel(metric, fontsize=11)
    ax.set_title(f'Сравнение {metric}', fontsize=12, fontweight='bo
    ax.set_xticks(x)
    ax.set_xticklabels(results_df['Model'], rotation=45, ha='right'
    ax.grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()
```



Для того чтобы обоснованно говорить насколько прогнозы моделей различны, а результаты одной хуже/лучше другой нужно провести статистический тест.

DM-тест: H_0 модели обладают одинаковой точностью H_1 модели имеют разную точность

In [27]:
```python
def diebold_mariano_test(y_true, y_pred1, y_pred2):

    y_true = np.asarray(y_true).flatten()
    y_pred1 = np.asarray(y_pred1).flatten()
    y_pred2 = np.asarray(y_pred2).flatten()

    # Абсолютные ошибки
    e1 = np.abs(y_true - y_pred1)
    e2 = np.abs(y_true - y_pred2)

    # Разность потерь
    d = e1 - e2

    # DM статистика
    dm_stat = np.mean(d) / np.sqrt(np.var(d) / len(d))

    # p-value (двусторонний тест)
    p_value = 2 * (1 - stats.norm.cdf(np.abs(dm_stat)))

    return dm_stat, p_value
```

Для начала определим лучшие модели по метрике MAE (она наиболее легко интерпретируемая) среди моделей основанных на стационарных данных и на нестационарных

In [28]:
```
# Разделяем модели на группы
nonstat_mask = results_df['Model'].str.contains('Нестац', regex=Fal
stat_mask = results_df['Model'].str.contains('Стац', regex=False)

nonstat_models = results_df[nonstat_mask].copy()
stat_models = results_df[stat_mask].copy()

# Находим лучшие модели по MAE
best_nonstat_idx = nonstat_models['MAE'].idxmin()
best_stat_idx = stat_models['MAE'].idxmin()

best_nonstat_name = results_df.loc[best_nonstat_idx, 'Model']
best_stat_name = results_df.loc[best_stat_idx, 'Model']

print("Модели для сравнения:")
print(f"  Лучшая нестационарная модель: {best_nonstat_name}")
print(f"    MAE: {results_df.loc[best_nonstat_idx, 'MAE']:.4f}")
print(f"\n  Лучшая стационарная модель: {best_stat_name}")
print(f"    MAE: {results_df.loc[best_stat_idx, 'MAE']:.4f}")
```

```
Модели для сравнения:
  Лучшая нестационарная модель: Holt-Winters (Нестац)
    MAE: 4.5915

  Лучшая стационарная модель: Holt (Стац)
    MAE: 4.3163
```

Теперь перейдём к извлечению прогнозов этих моделей

In [29]:
```
# Определяем прогнозы нестационарной модели
if 'SES' in best_nonstat_name:
    pred_nonstat = ses_pred_nonstat.values
elif 'Holt' in best_nonstat_name and 'Winters' not in best_nonstat_
    pred_nonstat = holt_pred_nonstat.values
else:
    pred_nonstat = hw_pred_nonstat.values

# Определяем прогнозы стационарной модели
if 'SES' in best_stat_name:
    pred_stat = restore_prices_ses.values
elif 'Holt' in best_stat_name and 'Winters' not in best_stat_name:
    pred_stat = restore_prices_holt.values
else:
    pred_stat = restore_prices_hw.values

# Реальные значения для сравнения
y_true_compare = test_nonstationary_aligned.values

# Выравниваем длину прогнозов
pred_nonstat_aligned = pred_nonstat[1:] if len(pred_nonstat) > len(
```

```
print(f"Длины данных для DM-теста:")
print(f"  y_true: {len(y_true_compare)}")
print(f"  pred_nonstat: {len(pred_nonstat_aligned)}")
print(f"  pred_stat: {len(pred_stat)}")
```

```
Длины данных для DM-теста:
  y_true: 503
  pred_nonstat: 503
  pred_stat: 503
```

Проводим статистический тест:

In [30]:
```
dm_stat, p_value = diebold_mariano_test(y_true_compare, pred_nonsta

print("\n" + "=" * 70)
print("ТЕСТ ДИБОЛЬДА-МАРИАНО (DIEBOLD-MARIANO TEST)")
print("=" * 70)

print(f"\nСравниваемые модели:")
print(f"  Модель 1: {best_nonstat_name}")
print(f"  Модель 2: {best_stat_name}")

print(f"\nРезультаты теста:")
print(f"  DM статистика: {dm_stat:.4f}")
print(f"  p-value:       {p_value:.4f}")

print(f"\nИнтерпретация:")
if p_value < 0.05:
    if dm_stat < 0:
        better_model = f"Модель 1 ({best_nonstat_name})"
    else:
        better_model = f"Модель 2 ({best_stat_name})"
    print(f"  Различия статистически значимы (p < 0.05)")
    print(f"  {better_model} ЗНАЧИМО ЛУЧШЕ")
else:
    print(f"  Различия НЕ значимы (p ≥ 0.05)")
    print(f"  Модели показывают ОДИНАКОВУЮ точность")

print("=" * 70)
```

```
==========================================================================
==
ТЕСТ ДИБОЛЬДА-МАРИАНО (DIEBOLD-MARIANO TEST)
==========================================================================
==

Сравниваемые модели:
  Модель 1: Holt-Winters (Нестац)
  Модель 2: Holt (Стац)

Результаты теста:
  DM статистика: 2.3481
  p-value:       0.0189

Интерпретация:
  Различия статистически значимы (p < 0.05)
  Модель 2 (Holt (Стац)) ЗНАЧИМО ЛУЧШЕ
==========================================================================
==
```

Таким образом гипотеза 1 подтвердилась. Приведение обучающей выборки к стаицонарному виду значимо улучшает прогноз

Гипотеза 2: Влияние частоты переобучения на качество прогноза

Зададим функцию для прогнозирования с периодиским переобучением

In [31]:
```python
def rolling_forecast(model_type, train_data, test_data, retraining_
    """
    Прогнозирование с периодическим переобучением

    Параметры:
    ----------
    model_type : str
        Тип модели ('SES', 'Holt', 'HW')
    train_data : pd.Series
        Обучающие данные
    test_data : pd.Series
        Тестовые данные
    retraining_period : int
        Период переобучения (количество шагов)
    return_errors : bool
        Если True, возвращает также ошибки по дням

    Возвращает:
    -----------
    tuple : (прогнозы, время выполнения) или (прогнозы, время, ошиб
    """
    start_time = time.time()

    train = train_data.copy()
    forecasts = []
```

```
    daily_errors = [] if return_errors else None

    for i in range(0, len(test_data), retraining_period):
        forecast_horizon = min(retraining_period, len(test_data) -

        # Обучаем модель
        if model_type == 'SES':
            model = SimpleExpSmoothing(train).fit(optimized=True)
        elif model_type == 'Holt':
            model = ExponentialSmoothing(train, trend='add', season
        else:  # HW
            model = ExponentialSmoothing(train, trend='add', season

        # Делаем прогноз
        pred = model.forecast(forecast_horizon)
        forecasts.extend(pred.values)

        # Рассчитываем ошибки, если требуется
        if return_errors:
            for j in range(forecast_horizon):
                if i + j < len(test_data):
                    pred_val = pred.iloc[j] if hasattr(pred, 'iloc'
                    real_val = test_data.iloc[i + j]
                    error = np.abs(real_val - pred_val)
                    daily_errors.append(error)

        # Добавляем реальные значения в train
        train = pd.concat([train, test_data.iloc[i:i+forecast_horiz

    elapsed_time = time.time() - start_time

    forecasts_array = np.array(forecasts[:len(test_data)])

    if return_errors:
        return forecasts_array, elapsed_time, np.array(daily_errors
    else:
        return forecasts_array, elapsed_time
```

Также задаём функцию прямого прогнозирования

In [32]:
```
def direct_forecast(model_type, train_data, test_data):
    """
    Прямое прогнозирование без переобучения

    Параметры:
    ----------
    model_type : str
        Тип модели ('SES', 'Holt', 'HW')
    train_data : pd.Series
        Обучающие данные
    test_data : pd.Series
        Тестовые данные

    Возвращает:
    -----------
```

```
    tuple : (прогнозы, время выполнения)
    """
    start_time = time.time()

    # Обучаем модель один раз
    if model_type == 'SES':
        model = SimpleExpSmoothing(train_data).fit(optimized=True)
    elif model_type == 'Holt':
        model = ExponentialSmoothing(train_data, trend='add', seaso
    else:  # HW
        model = ExponentialSmoothing(train_data, trend='add', seaso

    # Делаем прогноз на весь тестовый период
    forecast = model.forecast(len(test_data))

    elapsed_time = time.time() - start_time

    return forecast.values, elapsed_time
```

Зададим различные частоты переобучения моделей для нестационарных данных

In [33]:
```
retraining_periods = [1, 2, 5, 10]
results_hyp2_nonstat = []

print("=" * 70)
print("ГИПОТЕЗА 2: ROLLING FORECAST vs DIRECT (Нестационарные данны
print("=" * 70)

# Direct Forecast
print("\n[1/5] Direct Forecast...")
pred_direct_nonstat, time_direct_nonstat = direct_forecast('HW', tr

mae_direct = mean_absolute_error(test_nonstationary, pred_direct_no
rmse_direct = np.sqrt(mean_squared_error(test_nonstationary, pred_d
mape_direct = np.mean(np.abs((test_nonstationary.values - pred_dire

results_hyp2_nonstat.append({
    'Strategy': 'Direct',
    'Period k': '-',
    'MAE': mae_direct,
    'RMSE': rmse_direct,
    'MAPE': mape_direct,
    'Time (s)': time_direct_nonstat
})

print(f"  MAE: {mae_direct:.4f}, RMSE: {rmse_direct:.4f}, MAPE: {ma

# Rolling Forecast для разных k
for idx, k in enumerate(retraining_periods, start=2):
    print(f"\n[{idx}/5] Rolling Forecast k={k}...")
    pred_rolling, time_rolling = rolling_forecast('HW', train_nonst

    mae_rolling = mean_absolute_error(test_nonstationary, pred_roll
    rmse_rolling = np.sqrt(mean_squared_error(test_nonstationary, p
```

```
    mape_rolling = np.mean(np.abs((test_nonstationary.values - pred

    results_hyp2_nonstat.append({
        'Strategy': 'Rolling',
        'Period k': k,
        'MAE': mae_rolling,
        'RMSE': rmse_rolling,
        'MAPE': mape_rolling,
        'Time (s)': time_rolling
    })

    improvement = ((mae_direct - mae_rolling) / mae_direct) * 100
    print(f"  MAE: {mae_rolling:.4f}, RMSE: {rmse_rolling:.4f}, MAP
    print(f"  Улучшение vs Direct: {improvement:+.2f}%")

# Создаем таблицу результатов
results_hyp2_nonstat_df = pd.DataFrame(results_hyp2_nonstat)

print("\n" + "=" * 70)
print("ТАБЛИЦА РЕЗУЛЬТАТОВ (Нестационарные данные)")
print("=" * 70)
print(results_hyp2_nonstat_df.to_string(index=False))
print("=" * 70)
```

```
================================================================
==
ГИПОТЕЗА 2: ROLLING FORECAST vs DIRECT (Нестационарные данные)
================================================================
==

[1/5] Direct Forecast...
  MAE: 4.9529, RMSE: 6.8019, MAPE: 12.41%, Time: 0.05s

[2/5] Rolling Forecast k=1...
  MAE: 0.5696, RMSE: 0.7878, MAPE: 17.00%, Time: 23.52s
  Улучшение vs Direct: +88.50%

[3/5] Rolling Forecast k=2...
  MAE: 0.7094, RMSE: 0.9592, MAPE: 17.00%, Time: 11.60s
  Улучшение vs Direct: +85.68%

[4/5] Rolling Forecast k=5...
  MAE: 0.9102, RMSE: 1.2049, MAPE: 16.80%, Time: 4.67s
  Улучшение vs Direct: +81.62%

[5/5] Rolling Forecast k=10...
  MAE: 1.2020, RMSE: 1.6467, MAPE: 16.81%, Time: 2.36s
  Улучшение vs Direct: +75.73%

================================================================
==
ТАБЛИЦА РЕЗУЛЬТАТОВ (Нестационарные данные)
================================================================
==
Strategy Period k      MAE     RMSE      MAPE  Time (s)
  Direct        - 4.952924 6.801911 12.406412  0.045574
 Rolling        1 0.569568 0.787834 16.999002 23.521228
 Rolling        2 0.709449 0.959152 17.003260 11.595060
 Rolling        5 0.910164 1.204934 16.801207  4.667912
 Rolling       10 1.202024 1.646701 16.810137  2.364118
================================================================
==
```

Зададим разные периоды переобучения для стационарных данных

In [34]:
```python
results_hyp2_stat = []

print("\n" + "=" * 70)
print("ГИПОТЕЗА 2: ROLLING FORECAST vs DIRECT (Стационарные данные)
print("=" * 70)

# Direct Forecast
print("\n[1/5] Direct Forecast...")
pred_direct_stat, time_direct_stat = direct_forecast('HW', train_st

# Восстанавливаем цены
pred_direct_stat_restored = restore_prices_from_log_returns(
    pd.Series(pred_direct_stat, index=test_stationary.index),
    train_nonstationary.iloc[-1]
)
```

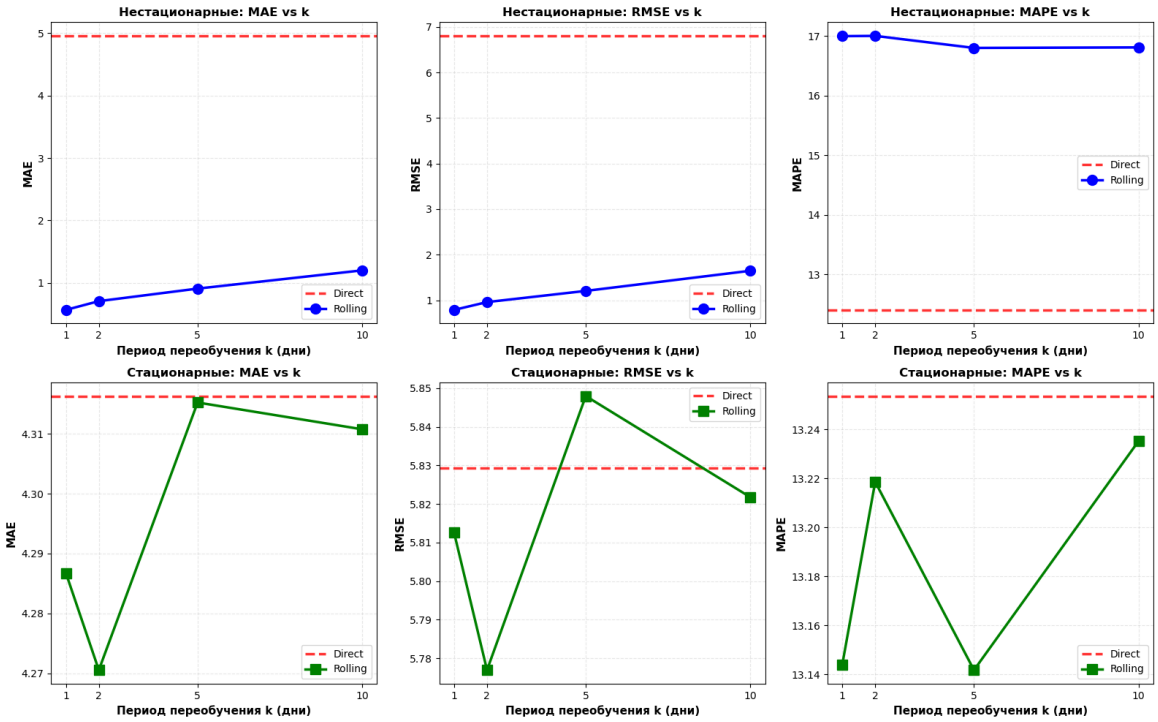
```
pred_direct_stat_aligned = pred_direct_stat_restored[:len(test_nons

mae_direct_stat = mean_absolute_error(test_nonstationary_aligned.va
rmse_direct_stat = np.sqrt(mean_squared_error(test_nonstationary_al
mape_direct_stat = np.mean(np.abs((test_nonstationary_aligned.value

results_hyp2_stat.append({
    'Strategy': 'Direct',
    'Period k': '-',
    'MAE': mae_direct_stat,
    'RMSE': rmse_direct_stat,
    'MAPE': mape_direct_stat,
    'Time (s)': time_direct_stat
})

print(f"  MAE: {mae_direct_stat:.4f}, RMSE: {rmse_direct_stat:.4f},

# Rolling Forecast для разных k
for idx, k in enumerate(retraining_periods, start=2):
    print(f"\n[{idx}/5] Rolling Forecast k={k}...")
    pred_rolling_stat, time_rolling_stat = rolling_forecast('HW', t

    # Восстанавливаем цены
    pred_rolling_stat_restored = restore_prices_from_log_returns(
        pd.Series(pred_rolling_stat, index=test_stationary.index),
        train_nonstationary.iloc[-1]
    )
    pred_rolling_stat_aligned = pred_rolling_stat_restored[:len(tes

    mae_rolling_stat = mean_absolute_error(test_nonstationary_align
    rmse_rolling_stat = np.sqrt(mean_squared_error(test_nonstationa
    mape_rolling_stat = np.mean(np.abs((test_nonstationary_aligned.

    results_hyp2_stat.append({
        'Strategy': 'Rolling',
        'Period k': k,
        'MAE': mae_rolling_stat,
        'RMSE': rmse_rolling_stat,
        'MAPE': mape_rolling_stat,
        'Time (s)': time_rolling_stat
    })

    improvement = ((mae_direct_stat - mae_rolling_stat) / mae_direc
    print(f"  MAE: {mae_rolling_stat:.4f}, RMSE: {rmse_rolling_stat
    print(f"  Улучшение vs Direct: {improvement:+.2f}%")

# Создаем таблицу результатов
results_hyp2_stat_df = pd.DataFrame(results_hyp2_stat)

print("\n" + "=" * 70)
print("ТАБЛИЦА РЕЗУЛЬТАТОВ (Стационарные данные)")
print("=" * 70)
print(results_hyp2_stat_df.to_string(index=False))
print("=" * 70)
```

```
==========================================================================
==
ГИПОТЕЗА 2: ROLLING FORECAST vs DIRECT (Стационарные данные)
==========================================================================
==

[1/5] Direct Forecast...
  MAE: 4.3163, RMSE: 5.8292, MAPE: 13.25%, Time: 0.05s

[2/5] Rolling Forecast k=1...
  MAE: 4.2868, RMSE: 5.8126, MAPE: 13.14%, Time: 26.21s
  Улучшение vs Direct: +0.68%

[3/5] Rolling Forecast k=2...
  MAE: 4.2706, RMSE: 5.7770, MAPE: 13.22%, Time: 13.27s
  Улучшение vs Direct: +1.06%

[4/5] Rolling Forecast k=5...
  MAE: 4.3152, RMSE: 5.8479, MAPE: 13.14%, Time: 5.28s
  Улучшение vs Direct: +0.02%

[5/5] Rolling Forecast k=10...
  MAE: 4.3108, RMSE: 5.8217, MAPE: 13.24%, Time: 2.68s
  Улучшение vs Direct: +0.13%

==========================================================================
==
ТАБЛИЦА РЕЗУЛЬТАТОВ (Стационарные данные)
==========================================================================
==
Strategy Period k      MAE     RMSE      MAPE  Time (s)
  Direct        - 4.316301 5.829222 13.253580  0.052424
 Rolling        1 4.286762 5.812556 13.144024 26.206120
 Rolling        2 4.270557 5.776963 13.218576 13.267260
 Rolling        5 4.315225 5.847944 13.141806  5.279993
 Rolling       10 4.310751 5.821731 13.235237  2.677159
==========================================================================
==
```

Построим графики для анализа влияния частоты переобучения

In [35]:
```python
fig, axes = plt.subplots(2, 3, figsize=(16, 10))

metrics = ['MAE', 'RMSE', 'MAPE']

# Нестационарные данные
for col, metric in enumerate(metrics):
    ax = axes[0, col]

    if metric == 'MAE':
        direct_val = mae_direct
    elif metric == 'RMSE':
        direct_val = rmse_direct
    else:
        direct_val = mape_direct
```

```
    # Горизонтальная линия для Direct
    ax.axhline(direct_val, color='red', linestyle='--', linewidth=2

    # Линия для Rolling
    rolling_data = results_hyp2_nonstat_df[results_hyp2_nonstat_df[
    ax.plot(rolling_data['Period k'], rolling_data[metric],
            marker='o', color='blue', linewidth=2.5, markersize=10,

    ax.set_xlabel('Период переобучения k (дни)', fontsize=11, fontw
    ax.set_ylabel(metric, fontsize=11, fontweight='bold')
    ax.set_title(f'Нестационарные: {metric} vs k', fontsize=12, fon
    ax.legend(fontsize=10)
    ax.grid(True, alpha=0.3, linestyle='--')
    ax.set_xticks([1, 2, 5, 10])

# Стационарные данные
for col, metric in enumerate(metrics):
    ax = axes[1, col]

    if metric == 'MAE':
        direct_val = mae_direct_stat
    elif metric == 'RMSE':
        direct_val = rmse_direct_stat
    else:
        direct_val = mape_direct_stat

    # Горизонтальная линия для Direct
    ax.axhline(direct_val, color='red', linestyle='--', linewidth=2

    # Линия для Rolling
    rolling_data = results_hyp2_stat_df[results_hyp2_stat_df['Strat
    ax.plot(rolling_data['Period k'], rolling_data[metric],
            marker='s', color='green', linewidth=2.5, markersize=10

    ax.set_xlabel('Период переобучения k (дни)', fontsize=11, fontw
    ax.set_ylabel(metric, fontsize=11, fontweight='bold')
    ax.set_title(f'Стационарные: {metric} vs k', fontsize=12, fontw
    ax.legend(fontsize=10)
    ax.grid(True, alpha=0.3, linestyle='--')
    ax.set_xticks([1, 2, 5, 10])

plt.tight_layout()
plt.show()
```

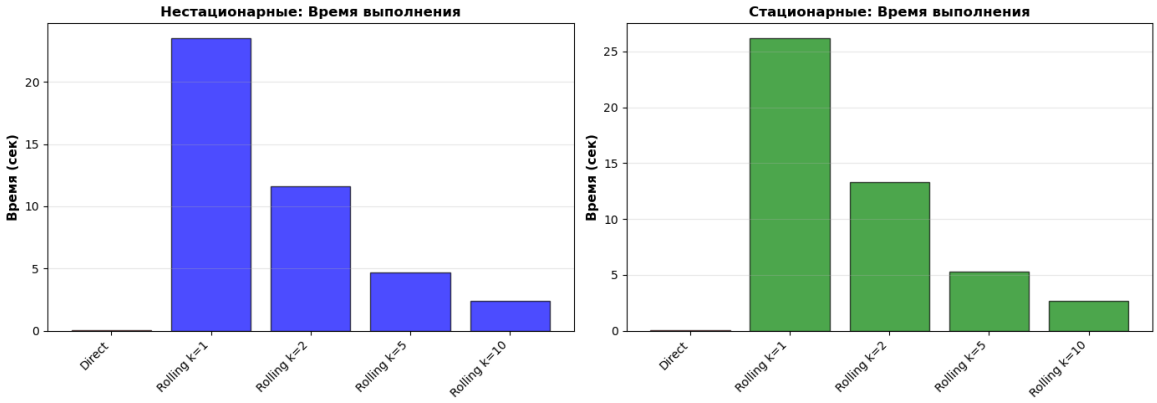


Видно что для моделей, обучаемых на нестационарных данных зависимость, MAE и RMSE зависят от k монотонно, то есть чем чаще мы будем переобучать тем точнее будет прогноз. Для моделей, обученных на стационрных данных, заисимость более сложная, не момнотонная. Точкой наилучшей отдачи является k=2, именно при таком значении частоты переобучения будут минимальные значения метрик.

Сравним время выполнения для различных моделей

In [36]:
```
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Нестационарные данные
strategies_nonstat = results_hyp2_nonstat_df['Strategy'] + ' k=' +
strategies_nonstat = strategies_nonstat.replace('Direct k=-', 'Dire

axes[0].bar(range(len(results_hyp2_nonstat_df)), results_hyp2_nonst
            color=['red'] + ['blue']*4, alpha=0.7, edgecolor='black
axes[0].set_xticks(range(len(results_hyp2_nonstat_df)))
axes[0].set_xticklabels(strategies_nonstat, rotation=45, ha='right'
axes[0].set_ylabel('Время (сек)', fontsize=11, fontweight='bold')
axes[0].set_title('Нестационарные: Время выполнения', fontsize=12,
axes[0].grid(True, alpha=0.3, axis='y')

# Стационарные данные
strategies_stat = results_hyp2_stat_df['Strategy'] + ' k=' + result
strategies_stat = strategies_stat.replace('Direct k=-', 'Direct')

axes[1].bar(range(len(results_hyp2_stat_df)), results_hyp2_stat_df[
            color=['red'] + ['green']*4, alpha=0.7, edgecolor='blac
axes[1].set_xticks(range(len(results_hyp2_stat_df)))
axes[1].set_xticklabels(strategies_stat, rotation=45, ha='right')
axes[1].set_ylabel('Время (сек)', fontsize=11, fontweight='bold')
axes[1].set_title('Стационарные: Время выполнения', fontsize=12, fo
```

```
axes[1].grid(True, alpha=0.3, axis='y')

plt.tight_layout()
plt.show()
```

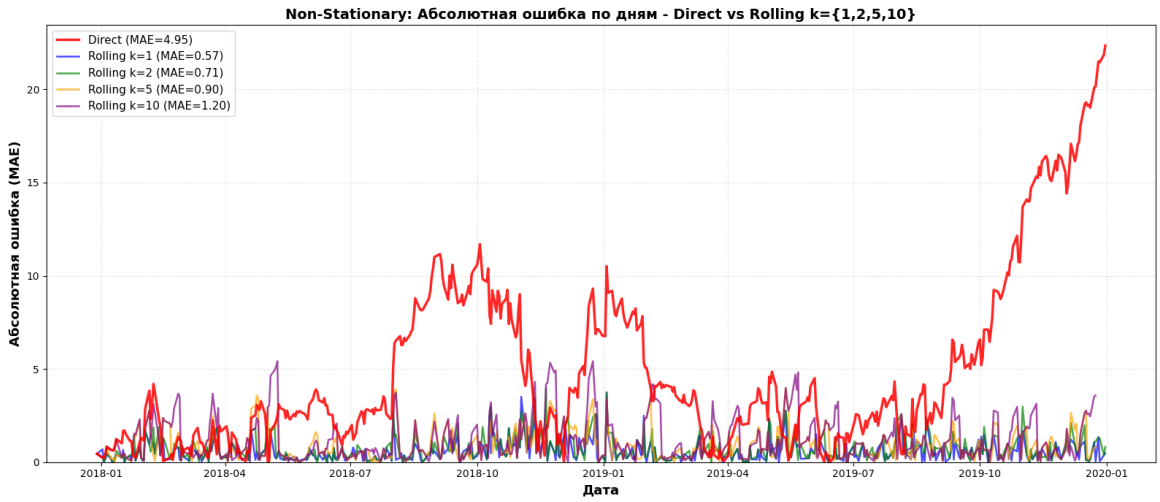


Теперь построим график MAE по дням для различных k на нестационарных данных

In [37]:
```
def rolling_forecast_with_daily_errors(model_type, train_data, test
    daily_errors = np.full(len(test_data), np.nan)
    current_train = train_data.copy()

    num_blocks = len(test_data) // k

    for block_idx in range(num_blocks):
        start_idx = block_idx * k
        end_idx = start_idx + k

        if model_type == 'HW':
            model = ExponentialSmoothing(current_train, trend='add'
        else:
            model = ExponentialSmoothing(current_train, trend='add'

        fitted = model.fit(optimized=True)
        forecast = fitted.forecast(steps=k)

        for i in range(k):
            if start_idx + i < len(test_data):
                pred_val = forecast.iloc[i] if hasattr(forecast, 'i
                real_val = test_data.iloc[start_idx + i]
                error = np.abs(real_val - pred_val)
                daily_errors[start_idx + i] = error

        real_block = test_data.iloc[start_idx:end_idx]
        current_train = pd.concat([current_train, real_block])

    return daily_errors


print("\nРасчёт ошибок по дням:\n")
direct_pred_values = np.asarray(pred_direct_nonstat).flatten()
test_values = np.asarray(test_nonstationary.values).flatten()
direct_errors_daily = np.abs(test_values - direct_pred_values)
```

```
print(f"Direct: {len(direct_errors_daily)} ошибок")

# Rolling ошибки для каждого k
rolling_errors_daily = {}

for k in [1, 2, 5, 10]:
    print(f"  k={k}...", end=' ', flush=True)
    errors = rolling_forecast_with_daily_errors('HW', train_nonstat
    rolling_errors_daily[k] = errors
    print(f"Средн. MAE = {np.nanmean(errors):.4f}")

print("\n" + "=" * 80)



fig, ax = plt.subplots(figsize=(16, 7))

# Direct (ЛИНИЯ по дням)
direct_mae = np.mean(direct_errors_daily)
ax.plot(test_nonstationary.index, direct_errors_daily,
        color='red', linewidth=2.5, alpha=0.85,
        label=f'Direct (MAE={direct_mae:.2f})', zorder=5)

# Rolling для каждого k
colors = ['blue', 'green', 'orange', 'purple']
for k, color in zip([1, 2, 5, 10], colors):
    errors = rolling_errors_daily[k]
    ax.plot(test_nonstationary.index, errors,
            label=f'Rolling k={k} (MAE={np.nanmean(errors):.2f})',
            color=color, linewidth=1.8, alpha=0.7, zorder=4)

ax.set_xlabel('Дата', fontsize=13, fontweight='bold')
ax.set_ylabel('Абсолютная ошибка (MAE)', fontsize=13, fontweight='b
ax.set_title('Non-Stationary: Абсолютная ошибка по дням - Direct vs
             fontsize=14, fontweight='bold')
ax.legend(fontsize=11, loc='upper left')
ax.grid(True, alpha=0.3, linestyle='--')
ax.set_ylim(bottom=0)

plt.tight_layout()
plt.show()
```

```
Расчёт ошибок по дням:

Direct: 504 ошибок
  k=1... Средн. MAE = 0.5696
  k=2... Средн. MAE = 0.7094
  k=5... Средн. MAE = 0.9045
  k=10... Средн. MAE = 1.1987

================================================================
============
```



По графику MAE vs Date мы можем увидеть, что когда происходит большое отклонения DirectPredict от Test, тогда другие прогнозы тоже дают пик, но намного меньшей длительности и величины. Из-за того что модель переобучается и подстраивается от резкие развороты рынка.

Теперь построим этот график для моделей на стационарных данных

In [38]:
```
def rolling_forecast_with_daily_errors_and_predictions(model_type,
    daily_predictions = np.full(len(test_data), np.nan)
    daily_errors = np.full(len(test_data), np.nan)
    current_train = train_data.copy()

    num_blocks = len(test_data) // k

    for block_idx in range(num_blocks):
        start_idx = block_idx * k
        end_idx = start_idx + k

        if model_type == 'HW':
            model = ExponentialSmoothing(current_train, trend='add'
        else:
            model = ExponentialSmoothing(current_train, trend='add'

        fitted = model.fit(optimized=True)
        forecast = fitted.forecast(steps=k)

        for i in range(k):
            if start_idx + i < len(test_data):
                pred_val = forecast.iloc[i] if hasattr(forecast, 'i
                real_val = test_data.iloc[start_idx + i]
                error = np.abs(real_val - pred_val)
                daily_predictions[start_idx + i] = pred_val
                daily_errors[start_idx + i] = error

        real_block = test_data.iloc[start_idx:end_idx]
        current_train = pd.concat([current_train, real_block])

    return daily_predictions, daily_errors

# ===== РАССЧИТЫВАЕМ ОШИБКИ ДЛЯ КАЖДОГО k (STATIONARY) =====
```

```
print("\nРасчёт ошибок по дням:\n")

# Direct ошибки ПО ДНЯМ
direct_pred_stat_values = np.asarray(pred_direct_stat_aligned).flat
test_stat_values = np.asarray(test_nonstationary_aligned.values).fl
direct_errors_stat_daily = np.abs(test_stat_values - direct_pred_st

print(f"Direct: {len(direct_errors_stat_daily)} ошибок, MAE={np.mea

# Rolling ошибки для каждого k (Stationary)
rolling_errors_stat_daily = {}

for k in [1, 2, 5, 10]:
    print(f"  k={k}...", end=' ', flush=True)

    # Получаем ПРОГНОЗЫ в log-returns
    pred_log_returns, _ = rolling_forecast_with_daily_errors_and_pr
        'HW', train_stationary, test_stationary, k
    )

    # Восстанавливаем цены из ПРОГНОЗОВ (не из ошибок!)
    pred_restored = restore_prices_from_log_returns(
        pd.Series(pred_log_returns, index=test_stationary.index[:le
        train_nonstationary.iloc[-1]
    )

    # Берём только нужную длину
    pred_restored_aligned = pred_restored[:len(test_nonstationary_a

    # Вычисляем ошибки между восстановленными ценами и реальными
    errors_stat = np.abs(test_stat_values - pred_restored_aligned)

    rolling_errors_stat_daily[k] = errors_stat
    print(f"Средн. MAE = {np.nanmean(errors_stat):.4f}")



fig, ax = plt.subplots(figsize=(16, 7))

# Direct
direct_mae_stat = np.mean(direct_errors_stat_daily)
ax.plot(test_nonstationary_aligned.index, direct_errors_stat_daily,
        color='red', linewidth=2.5, alpha=0.85,
        label=f'Direct (MAE={direct_mae_stat:.2f})', zorder=5)

# Rolling для каждого k
colors = ['blue', 'green', 'orange', 'purple']
for k, color in zip([1, 2, 5, 10], colors):
    errors = rolling_errors_stat_daily[k]
    ax.plot(test_nonstationary_aligned.index, errors,
            label=f'Rolling k={k} (MAE={np.nanmean(errors):.2f})',
            color=color, linewidth=1.8, alpha=0.7, zorder=4)

ax.set_xlabel('Дата', fontsize=13, fontweight='bold')
ax.set_ylabel('Абсолютная ошибка (MAE)', fontsize=13, fontweight='b
```

```
ax.set_title('Stationary: Абсолютная ошибка по дням - Direct vs Rol
             fontsize=14, fontweight='bold')
ax.legend(fontsize=11, loc='upper left')
ax.grid(True, alpha=0.3, linestyle='--')
ax.set_ylim(bottom=0)

plt.tight_layout()
plt.show()
```

```
Расчёт ошибок по дням:

Direct: 503 ошибок, MAE=4.3163
  k=1... Средн. MAE = 4.2868
  k=2... Средн. MAE = 4.2707
  k=5... Средн. MAE = 4.3158
  k=10... Средн. MAE = 4.3113
```

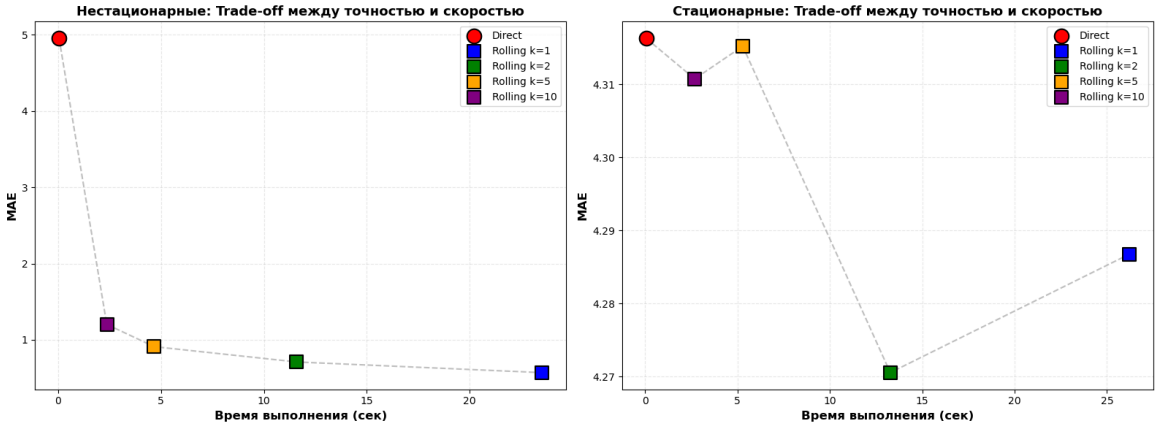
Особенного влияния переобучение на значение MAE не оказывает. Видимо дело в том, что для стационарного ряда прямой прогноз, даёт значительно лучший результат на стационарных данных, но переобучение, основанное на этом ряду занчительно не улучшает прогноз.

Теперь график зависимости MAE от времени

In [ ]:
```
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

#  Нестационарные данные
ax = axes[0]

# СОРТИРУЕМ по времени
sorted_indices_nonstat = np.argsort(results_hyp2_nonstat_df['Time (
times_nonstat = results_hyp2_nonstat_df['Time (s)'].values[sorted_i
maes_nonstat = results_hyp2_nonstat_df['MAE'].values[sorted_indices

# Соединяем точки линиями
ax.plot(times_nonstat, maes_nonstat, color='gray', linestyle='--',

# Direct точка (находим индекс Direct в отсортированных данных)
direct_idx_nonstat = np.where(results_hyp2_nonstat_df['Strategy'] =
direct_time_nonstat = results_hyp2_nonstat_df.loc[direct_idx_nonsta
```

```
direct_mae_nonstat = results_hyp2_nonstat_df.loc[direct_idx_nonstat

ax.scatter(direct_time_nonstat, direct_mae_nonstat, s=200, color='r
           label='Direct', zorder=5, edgecolor='black', linewidth=1

# Rolling точки
colors = ['blue', 'green', 'orange', 'purple']
retraining_periods = [1, 2, 5, 10]

for k, color in zip(retraining_periods, colors):
    rolling_row = results_hyp2_nonstat_df[
        (results_hyp2_nonstat_df['Strategy'] == 'Rolling') &
        (results_hyp2_nonstat_df['Period k'] == k)
    ]
    if not rolling_row.empty:
        t = rolling_row['Time (s)'].values[0]
        m = rolling_row['MAE'].values[0]
        ax.scatter(t, m, s=150, color=color, marker='s', label=f'Ro
                   zorder=4, edgecolor='black', linewidth=1.5)

ax.set_xlabel('Время выполнения (сек)', fontsize=12, fontweight='bo
ax.set_ylabel('MAE', fontsize=12, fontweight='bold')
ax.set_title('Нестационарные: Trade-off между точностью и скоростью
ax.legend(fontsize=10)
ax.grid(True, alpha=0.3, linestyle='--')

# Стационарные данные
ax = axes[1]

# СОРТИРУЕМ по времени
sorted_indices_stat = np.argsort(results_hyp2_stat_df['Time (s)'].v
times_stat = results_hyp2_stat_df['Time (s)'].values[sorted_indices
maes_stat = results_hyp2_stat_df['MAE'].values[sorted_indices_stat]

# Соединяем точки линиями
ax.plot(times_stat, maes_stat, color='gray', linestyle='--', linewi

# Direct точка
direct_idx_stat = np.where(results_hyp2_stat_df['Strategy'] == 'Dir
direct_time_stat = results_hyp2_stat_df.loc[direct_idx_stat, 'Time
direct_mae_stat = results_hyp2_stat_df.loc[direct_idx_stat, 'MAE']

ax.scatter(direct_time_stat, direct_mae_stat, s=200, color='red', m
           label='Direct', zorder=5, edgecolor='black', linewidth=1

# Rolling точки
for k, color in zip(retraining_periods, colors):
    rolling_row = results_hyp2_stat_df[
        (results_hyp2_stat_df['Strategy'] == 'Rolling') &
        (results_hyp2_stat_df['Period k'] == k)
    ]
    if not rolling_row.empty:
        t = rolling_row['Time (s)'].values[0]
        m = rolling_row['MAE'].values[0]
        ax.scatter(t, m, s=150, color=color, marker='s', label=f'Ro
                   zorder=4, edgecolor='black', linewidth=1.5)
```

```
ax.set_xlabel('Время выполнения (сек)', fontsize=12, fontweight='bo
ax.set_ylabel('MAE', fontsize=12, fontweight='bold')
ax.set_title('Стационарные: Trade-off между точностью и скоростью',
ax.legend(fontsize=10)
ax.grid(True, alpha=0.3, linestyle='--')

plt.tight_layout()
plt.show()
```



Теперь перейдём к итоговым выводам:

In [40]:
```
print("=" * 80)
print("ИТОГОВЫЕ ВЫВОДЫ")
print("=" * 80)

print("\nГИПОТЕЗА 1: Влияние стационарности\n")

best_nonstat_row = results_df[results_df['Model'].str.contains('Нес
    results_df[results_df['Model'].str.contains('Нестац')]['MAE'].i
]
best_stat_row = results_df[results_df['Model'].str.contains('Стац')
    results_df[results_df['Model'].str.contains('Стац')]['MAE'].idx
]

print(f"Лучшая нестационарная модель: {best_nonstat_row['Model']}")
print(f"  MAE: {best_nonstat_row['MAE']:.4f}, RMSE: {best_nonstat_r

print(f"\nЛучшая стационарная модель: {best_stat_row['Model']}")
print(f"  MAE: {best_stat_row['MAE']:.4f}, RMSE: {best_stat_row['RM

improvement_h1 = ((best_nonstat_row['MAE'] - best_stat_row['MAE'])
print(f"\nУлучшение при использовании стационарного ряда: {improvem

if improvement_h1 > 0:
    print("ГИПОТЕЗА 1 ПОДТВЕРЖДЕНА: Приведение к стационарности улу
else:
    print("ГИПОТЕЗА 1 НЕ ПОДТВЕРЖДЕНА: Стационарность не улучшает п

print("\n" + "-" * 80)
print("\nГИПОТЕЗА 2: Влияние частоты переобучения\n")

best_nonstat_h2_row = results_hyp2_nonstat_df.loc[results_hyp2_nons
```

```
best_stat_h2_row = results_hyp2_stat_df.loc[results_hyp2_stat_df['M

print(f"Нестационарные данные:")
print(f"  Лучшая стратегия: {best_nonstat_h2_row['Strategy']} k={be
print(f"  MAE: {best_nonstat_h2_row['MAE']:.4f}, Время: {best_nonst

print(f"\nСтационарные данные:")
print(f"  Лучшая стратегия: {best_stat_h2_row['Strategy']} k={best_
print(f"  MAE: {best_stat_h2_row['MAE']:.4f}, Время: {best_stat_h2_

better_nonstat = best_nonstat_h2_row['Strategy'] == 'Rolling'
better_stat = best_stat_h2_row['Strategy'] == 'Rolling'

print(f"\nГипотеза 2 подтверждена?")
print(f"  Нестационарные: {'ДА' if better_nonstat else 'НЕТ'} (Roll
print(f"  Стационарные: {'ДА' if better_stat else 'НЕТ'} (Rolling л

if better_nonstat and better_stat:
    print("\nГИПОТЕЗА 2 ПОДТВЕРЖДЕНА: Периодическое переобучение пр
elif better_nonstat or better_stat:
    print("\nГИПОТЕЗА 2 ЧАСТИЧНО ПОДТВЕРЖДЕНА: Зависит от типа данн
else:
    print("\nГИПОТЕЗА 2 НЕ ПОДТВЕРЖДЕНА: Прямой прогноз не хуже пер

print("\n" + "=" * 80)
```

```
================================================================================
ИТОГОВЫЕ ВЫВОДЫ
================================================================================

ГИПОТЕЗА 1: Влияние стационарности

Лучшая нестационарная модель: Holt-Winters (Нестац)
  MAE: 4.5915, RMSE: 5.6620, MAPE: 9.97%

Лучшая стационарная модель: Holt (Стац)
  MAE: 4.3163, RMSE: 5.8292, MAPE: 8.88%

Улучшение при использовании стационарного ряда: +5.99%
ГИПОТЕЗА 1 ПОДТВЕРЖДЕНА: Приведение к стационарности улучшает прогноз

--------------------------------------------------------------------------------

ГИПОТЕЗА 2: Влияние частоты переобучения

Нестационарные данные:
  Лучшая стратегия: Rolling k=1
  MAE: 0.5696, Время: 23.52s

Стационарные данные:
  Лучшая стратегия: Rolling k=2
  MAE: 4.2706, Время: 13.27s

Гипотеза 2 подтверждена?
  Нестационарные: ДА (Rolling лучше Direct)
  Стационарные: ДА (Rolling лучше Direct)

ГИПОТЕЗА 2 ПОДТВЕРЖДЕНА: Периодическое переобучение превосходит прямой прогноз

================================================================================
```

In [ ]: