

課題6 - リアルタイム株式取引分析システム

リアルタイム株式データ処理・可視化システム

実装のきっかけ

- 個人的な興味: 元々株取引に関心があった
- リアルタイム処理: データストリーム処理技術への興味
- 可視化技術: WebSocket + React による動的UI実装

システム概要

主な機能



- **5000名の投資家**による自動取引生成
- 動的株価変動（取引量に応じた価格調整）
- リアルタイムポートフォリオ分析
- 統計情報の可視化（性別・年代別）

（ここに図を挿入、図はシステム全体の概要を視覚的に表す。取引生成→株価変動→ポートフォリオ更新→Web表示の流れ）

標準課題からの主な変更点

アーキテクチャの改善

旧構成: 独立した株価・取引生成

StockPrice.java 
Transaction.java  StockProcessor.java

新構成: 統合されたデータフロー

Transaction.java → PriceManager.java → StockProcessor.java

システム技術スタック

バックエンド

- **Java 17+:** メインサーバー
- **WebSocket:** リアルタイム通信
- **マイクロサービス:** 分散アーキテクチャ

フロントエンド

- **React + TypeScript:** UI フレームワーク
- **Chart.js:** グラフ可視化
- **TailwindCSS:** レスポンシブ対応

メイン処理フロー

（ここに図を挿入、図は以下のフローを視覚的に表す）

1. Transaction.java（取引生成）
↓ Socket通信
2. PriceManager.java（価格計算・更新）
↓ 統合データ送信
3. StockProcessor.java（分析・集計）
↓ WebSocket
4. React Frontend（リアルタイム表示）

データ整合性の保証

問題

- 取引処理時の株価参照タイミング
- 並行アクセスによる不整合
- 空売り防止システム

解決策

- **PriceManager**による一元管理
- 取引時点の価格保証
- 保有株数ベースの売買制限

レスポンシブUI実装

PC (1200px以上) - 3列表示

[統計情報] | [ポートフォリオ] | [取引履歴]

タブレット・スマホ - 2列表示

[ポートフォリオ + 統計] | [取引履歴 + 統計]

(ここに図を挿入、図はPC・タブレット・スマホでの表示レイアウトの違いを視覚的に表す)

工夫した点 1: 現実的な取引生成

スマートな売買量決定

```
if (保有株数 == 0) {  
    買いのみ生成  
} else if (保有株数 <= 50) {  
    バランス売買  
} else {  
    利確傾向 (売り優先)  
}
```

工夫した点 2: パフォーマンス最適化 ⚡

効率的なデータ管理

- **ConcurrentHashMap**: 並行処理対応
- スライディングウィンドウ: 直近5秒の取引のみ保持
- 増分計算: 全データ再計算を回避

(ここに図を挿入、図はスライディングウィンドウによる効率的なデータ管理を視覚的に表す)

工夫した点 3: 美しいデータ可視化

Chart.js による動的グラフ

- 円グラフ: 地域別資産配分
- 棒グラフ: 性別・年代別統計
- リアルタイム更新: スムーズなアニメーション

(ここに図を挿入、図は円グラフと棒グラフの例を視覚的に表す)

苦労した点 1: データ同期問題 🥵

問題の詳細



- **Transaction.java**: 起動時から保有株数を蓄積
- **StockProcessor**: 途中接続でデータ不整合
- 結果: マイナス保有株数の発生

応急対応

```
{portfolioSummary.stocks
    .filter(stock => stock.quantity > 0) // マイナス除外
    .map(stock => (/* 表示処理 */))
}
```

苦勞した点 2: マイクロサービス間通信

複雑な通信フロー

Transaction  PriceManager  StockProcessor

課題

- 双方向通信の設計複雑性
- 接続状態の伝播
- エラーハンドリング

実装できなかった点

主な未実装機能

1. 完全な空売り防止

- 現在: フロントエンド側フィルタリング
- 理想: バックエンドでの根本的解決

2. データ永続化

- 現在: メモリ内保持のみ
- 理想: データベース連携

今後の改善案・展望

短期的改善

- データ同期の根本解決
- エラーハンドリング強化

長期的展望

- 信用取引システム (空売り機能)
- 株式配当システム
- 投資家行動**AI** (機械学習)
- リアルタイム通知機能

デモンストレーション





見どころ

1. リアルタイム取引の瞬間更新
2. 株価変動の連動性
3. ポートフォリオの動的計算
4. 統計グラフのスムーズな更新
5. レスポンシブ画面切り替え

(ここに図を挿入、図は実際のWebアプリケーション画面のスクリーンショットを表す)

まとめ

達成できたこと

-  リアルタイム株式分析システムの構築
-  マイクロサービスアーキテクチャの実装
-  レスポンシブWeb UIの開発
-  現実的な取引ロジックの実現

学んだこと

- **WebSocket**を使ったリアルタイム通信
- **React**での動的データ可視化
- 分散システムの設計課題

ご質問・ディスカッション

ありがとうございました