Echarts (Line, Bar)

ECharts Line, Bar Chart 구현 가이드

📌 시작하기 전에

- 자바스크립트를 몰라도 괜찮습니다
- 아래 코드를 **전체 복사** 후 필요한 부분만 수정하세요
  - (가장 하단의 `clearInterval(this.myInterval)` 은 destroy tab에)
  - REST API와 DB Query의 코드가 조금 차이가 있습니다.
  - DBQuery는 응답을 Format하는 함수가 추가되어 있습니다.
- `this.`는 각 컴포넌트를 구분하는 표시이므로 그대로 두세요

🔧 전체 코드 구조 (REST API)

```
// 1. 기본 함수 정의
this.createChart = (element) => echarts.init(element)

this.settingChart = fx.curry((chart, option) => (chart.setOption(option)))

this.setXAxis = fx.curry((option, response) => {
    option.xAxis = {
        type: 'category',
        data: response.categories,
    };
    return response;
});

this.setYAxis = fx.curry((option, response) => {
    option.yAxis = {
        type: 'value',
    };
    return response;
});
```

```
this.setLineSeries = fx.curry((option, response) => {
    option.series = response.series.map(serie => ({
        name: serie.name,
        data: serie.data,
        type: 'line',
        smooth: true
    }));
    return response;
});

this.getChartData = async apiArguments => {
    const { response } = await WKit.fetchData(...Object.values(apiArgument
s)).catch(console.error);
    return response;
};

// 2. 차트 생성 및 설정
const chartElement = this.element.querySelector('#echarts');
this.myLineChart = this.createChart(chartElement);

const chartOption = {};

// 🔴 API 연결 정보 (수정 필요)
const apiArguments = {
    page: this.page,
    datasetName: 'api_line_chart',    // ← 여러분의 API 이름으로 변경
    params: { port: 3000, series: 5, days: 10 } // ← 필요한 파라미터로 변경
}

// 3. 차트 로드 함수
const loadChart = () => fx.go(
    this.getChartData(apiArguments),
    this.setXAxis(chartOption),
    this.setYAxis(chartOption),
    this.setLineSeries(chartOption),
    (_) => this.settingChart(this.myLineChart, chartOption)
)
```

```
loadChart();

// 🟡 자동 새로고침 설정 (필요시 수정)
this.REFRESH_INTERVAL = 5000;  // 5초마다 새로고침
this.myInterval = setInterval(loadChart, this.REFRESH_INTERVAL);

// 🔴 4. 정리 (destroy 섹션)
clearInterval(this.myInterval);
```

🔧 전체 코드 구조 (DB Query)

```
this.createChart = (element) => echarts.init(element)

this.settingChart = fx.curry((chart, option) => (chart.setOption(option)))

this.setXAxis = fx.curry((option, response) => {
    option.xAxis = {
        type: 'category',
        data: response.categories,
    };
    return response;
});

this.setYAxis = fx.curry((option, response) => {
    option.yAxis = {
        type: 'value',
    };
    return response;
});

this.setLineSeries = fx.curry((option, response) => {
    option.series = response.series.map(serie => ({
        name: serie.name,
        data: serie.data,
        type: 'line',
        smooth: true
```

```
    }));
    return response;
});


this.getChartData = async apiArguments => {
    const { response } = await WKit.fetchData(...Object.values(apiArgument
s)).catch(console.error);
    return formatChartResponse(response);
};

function formatChartResponse(rows) {
    const categories = [];
    const seriesData = {};
    rows.forEach(row => {
        if (!categories.includes(row.day)) categories.push(row.day);
        if (!seriesData[row.metric_name]) seriesData[row.metric_name] = [];
        seriesData[row.metric_name].push(row.value);
    });
    return {
        categories,
        series: Object.entries(seriesData).map(([name, data]) => ({ name, data
}))
    };
};


const chartElement = this.element.querySelector('#echarts');
this.myLineChart = this.createChart(chartElement);

const chartOption = {};

// 🔴 API 연결 정보 (수정 필요)
const apiArguments = {
    page: this.page,
    datasetName: 'api_line_chart_query',    // ← 여러분의 API 이름으로 변경
    params: {  series: "5", days: "10" } // ← 필요한 파라미터로 변경, query는 par
am을 ""문자열로
```

```
}

// 3. 차트 로드 함수
const loadChart = () => fx.go(
    this.getChartData(apiArguments),
    this.setXAxis(chartOption),
    this.setYAxis(chartOption),
    this.setLineSeries(chartOption),
    (_) => this.settingChart(this.myLineChart, chartOption)
)

loadChart();

// 🟡 자동 새로고침 설정 (필요시 수정)
this.REFRESH_INTERVAL = 5000;  // 5초마다 새로고침
this.myInterval = setInterval(loadChart, this.REFRESH_INTERVAL);

// 🔴 4. 정리 (destroy 섹션)
clearInterval(this.myInterval);
```

📊 데이터 구조 이해하기

API 응답 구조:

```
{
  categories: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'],  // X축 라벨
  series: [
    {
      name: 'Sales',     // 범례에 표시될 이름
      data: [150, 230, 224, 218, 135, 147, 260]  // Y축 값들
    },
    {
      name: 'Profit',
      data: [80, 120, 110, 100, 70, 75, 140]
    }
  ]
}
```

DB Query 응답 구조:


```
[
    {
      "value": 53,
      "day": "Mon",
      "metric_name": "Sales"
    },
    {
      "value": 164,
      "day": "Mon",
      "metric_name": "Profit"
    },
    {
      "value": 83,
      "day": "Mon",
      "metric_name": "Revenue"
    },
    {
      "value": 134,
      "day": "Tue",
      "metric_name": "Sales"
    },
    {
      "value": 227,
      "day": "Tue",
      "metric_name": "Profit"
    },
    {
      "value": 317,
      "day": "Tue",
      "metric_name": "Revenue"
    },
    {
      "value": 166,
      "day": "Wed",
      "metric_name": "Sales"
    },
```

```
    {
      "value": 125,
      "day": "Wed",
      "metric_name": "Profit"
    },
    {
      "value": 226,
      "day": "Wed",
      "metric_name": "Revenue"
    },
    {
      "value": 188,
      "day": "Thu",
      "metric_name": "Sales"
    },
    {
      "value": 125,
      "day": "Thu",
      "metric_name": "Profit"
    },
    {
      "value": 283,
      "day": "Thu",
      "metric_name": "Revenue"
    }
]
```

중요 포인트:

- `categories`: X축에 표시될 라벨들 (날짜, 요일, 월 등)
- `series`: 실제 차트 선들의 데이터
- 각 series는 `name`(이름)과 `data`(값 배열)를 가져야 합니다

🎯 데이터 흐름 설명

`loadChart` 함수의 동작:

```
const loadChart = () ⇒ fx.go(
    this.getChartData(apiArguments),    // 1단계: API 호출
    this.setXAxis(chartOption),         // 2단계: X축 설정
    this.setYAxis(chartOption),         // 3단계: Y축 설정
    this.setLineSeries(chartOption),    // 4단계: 선 데이터 설정
    (_) => this.settingChart(this.myLineChart, chartOption)  // 5단계: 차트 그리기
)
```

단계별 설명:

1. **API 데이터 가져오기:** `{ categories: [...], series: [...] }` 형태의 response
2. **X축 설정:** `response.categories`를 X축에 배치
3. **Y축 설정:** 숫자 값을 표시할 Y축 생성
4. **Series 설정:** 각 데이터를 Line 형태로 변환
5. **차트 렌더링:** 완성된 옵션으로 차트 그리기

데이터 전달 과정:

API 호출 → response 객체 → 각 축과 시리즈 설정 → 차트 표시

✏️ 수정 가능한 부분

1. API 정보 변경 (🔴 필수)

```
const apiArguments = {
    page: this.page,
    datasetName: 'sales_chart',      // API 이름 변경
    params: {
        port: 3000,
        series: 3,                   // 3개 시리즈
        days: 30,                    // 30일 데이터
        department: 'marketing'      // 추가 파라미터
    }
}
```

2. Bar Chart로 변경 (🟡 선택)

```
// setLineSeries를 setBarSeries로 변경
this.setBarSeries = fx.curry((option, response) => {
    option.series = response.series.map(serie => ({
        name: serie.name,
        data: serie.data,
        type: 'bar'     // 'line' → 'bar'로 변경
        // smooth 제거 (bar에는 불필요)
    }));
    return response;
});

// loadChart 함수에서도 변경
const loadChart = () => fx.go(
    this.getChartData(apiArguments),
    this.setXAxis(chartOption),
    this.setYAxis(chartOption),
    this.setBarSeries(chartOption),  // ← 여기 변경
    (_) => this.settingChart(this.myLineChart, chartOption)
)
```

3. 새로고침 주기 변경 (🟡 선택)

```
this.REFRESH_INTERVAL = 10000;  // 10초로 변경
// 또는 자동 새로고침을 원하지 않으면 setInterval 라인 삭제
```

🔍 Preview 섹션 이해

Preview는 개발 중 미리보기용입니다:

```
const targetElement = this.element.querySelector('#echarts')
this.previewChart = echarts.init(targetElement);

// API와 동일한 구조의 샘플 데이터
const response = {
  categories: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'],
```

```
  series: [
    {
      name: 'Sales',
      data: [150, 230, 224, 218, 135, 147, 260]
    },
    {
      name: 'Profit',
      data: [80, 120, 110, 100, 70, 75, 140]
    }
  ]
};

// 실제 코드와 동일한 방식으로 처리
let option = {};
option.xAxis = {
  type: 'category',
  data: response.categories
};
option.yAxis = {
  type: 'value'
};
option.series = response.series.map(serie => ({
  name: serie.name,
  data: serie.data,
  type: 'line',
  smooth: true
}));

this.previewChart.setOption(option);
```

Preview의 역할:

- API 연결 전에 차트가 잘 작동하는지 확인
- 데이터 구조와 처리 방식을 미리 테스트
- 실제 코드와 동일한 로직 사용

⚠️ 자주 발생하는 문제

차트가 안 보일 때

- HTML에 `<div id="echarts"></div>` 확인
- div에 높이 지정: `<div id="echarts" style="height: 400px;"></div>`
- 브라우저 콘솔(F12)에서 에러 메시지 확인

데이터가 안 나올 때

- `datasetName`이 정확한지 확인
- **데이터 구조와 함수 매칭 확인:**
  - `setXAxis`에서 사용하는 속성(예: `response.categories`)이 실제로 있는지
  - `setLineSeries`에서 사용하는 속성(예: `response.series`)이 실제로 있는지
  - 브라우저 콘솔에서 `console.log(response)`로 구조 확인 후 필요시 함수 수정
- **다른 구조의 API라면 함수 수정:**

```
// 예: API가 { labels: [...], datasets: [...] } 구조라면
this.setXAxis = fx.curry((option, response) => {
    option.xAxis = {
        type: 'category',
        data: response.labels,  // categories → labels로 변경
    };
    return response;
});
```

X축 라벨이 겹칠 때

```
this.setXAxis = fx.curry((option, response) => {
    option.xAxis = {
        type: 'category',
        data: response.categories,
        axisLabel: {
            rotate: 45,  // 45도 회전
            interval: 0  // 모든 라벨 표시
        }
    };
```

```
    return response;
});
```

✅ 체크리스트

- ☐ HTML에 `<div id="echarts" style="height: 400px;"></div>` 존재
- ☐ `datasetName`을 내 API 이름으로 변경
- ☐ `params`를 필요한 값으로 수정
- ☐ **각 설정 함수가 올바른 데이터를 받고 있는지 확인**
  - X축 데이터가 배열로 전달되는지
  - Series 데이터가 차트에 맞는 형태인지
  - 필요시 `setXAxis`, `setLineSeries` 함수 내부 수정