第四章 报表分析

1. 设备不同状态统计

1.1 需求分析

系统内要在首页以饼图的方式显示不同状态设备的占比百分比分布，具体效果如下图：

1.2 实现思路

（1）编写从ElasticSearch中获取断网设备总数、断网数量、告警数量的方法

（2）通过运算的到正常设备数，将数据组合成前端需要的数据格式返回

1.3 代码实现

1.3.1 离线与在线数量统计

（1）在ESRepository添加统计全部设备数量的方法：

```
    /**
     * 获取所有的设备数量
     * @return
     */
    public Long getAllDeviceCount(){
        CountRequest countRequest = new CountRequest("devices");
        countRequest.query(QueryBuilders.matchAllQuery());
        try {
            CountResponse response =
client.count(countRequest,RequestOptions.DEFAULT);

            return response.getCount();
        } catch (IOException e) {
            log.error("es count error",e);

            return 0L;
        }
    }
```

（2）在ESRepository添加统计离线设备数量的方法：

```
    /**
     * 获取当前离线设备总数
     * @return
     */
    public Long getOfflineCount(){
        CountRequest countRequest = new CountRequest("devices");
        BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
        boolQueryBuilder.must(QueryBuilders.termQuery("online",false));
        countRequest.query(boolQueryBuilder);
        try {
            CountResponse response =
client.count(countRequest,RequestOptions.DEFAULT);

            return response.getCount();
        } catch (IOException e) {
            log.error("es count error",e);

            return 0L;
        }
    }
```

（3）在ESRepository添加统计报警设备数量的方法：

```
    /**
     * 获取当前告警设备总数
     * @return
     */
    public Long getAlarmCount(){
        CountRequest countRequest = new CountRequest("devices");
        BoolQueryBuilder boolQueryBuilder = QueryBuilders.boolQuery();
        boolQueryBuilder.must(QueryBuilders.termQuery("online",false));
        boolQueryBuilder.must(QueryBuilders.termQuery("alarm",true));
        countRequest.query(boolQueryBuilder);
        try {
            CountResponse response =
client.count(countRequest,RequestOptions.DEFAULT);

            return response.getCount();
        } catch (IOException e) {
            log.error("es count error",e);

            return 0L;
        }
    }
```

1.3.2 报表数据封装

（1）添加vo，用于状态占比的封装

```
package com.yikekong.vo;

import lombok.Data;

import java.io.Serializable;
import java.math.BigDecimal;

@Data
public class DeviceStatusCollectVO implements Serializable{
    /**
     * 状态名
     */
    private String name;
    /**
     * 占比
     */
    private BigDecimal value;
}
```

（2）创建报表服务接口ReportService，并定义获取设备状态分布的方法

```
package com.yikekong.service;
import com.yikekong.vo.DeviceStatusCollectVO;
import java.util.List;

/**
 * 报表服务
 */
public interface ReportService {

    /**
     * 获取设备状态分布
     * @return
     */
    List<DeviceStatusCollectVO> getStatusCollect();

}
```

创建实现类，并在实现类中实现该方法

```
package com.yikekong.service.impl;
import com.google.common.collect.Lists;
import com.yikekong.es.ESRepository;
import com.yikekong.service.ReportService;
import com.yikekong.util.Calculator;
import com.yikekong.vo.DeviceStatusCollectVO;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
@Slf4j
public class ReportServiceImpl implements ReportService {

    @Autowired
    private ESRepository esRepository;

    @Override
    public List<DeviceStatusCollectVO> getStatusCollect() {
        List<DeviceStatusCollectVO> result = Lists.newArrayList();

        //当前所有上报过指标的设备
        Long totalDevicesCount = esRepository.getAllDeviceCount();

        //获取所有离线设备数
        Long disconnectCounts = esRepository.getOfflineCount();

        //获取当前所有告警设备总数
        Long alarmCounts = esRepository.getAlarmCount();

        //当前所有正常的设备
        Long normalDevicesCount = totalDevicesCount - alarmCounts -
disconnectCounts;
        if(normalDevicesCount < 0){
            normalDevicesCount = 0L;
        }

        DeviceStatusCollectVO normalCollect = new
DeviceStatusCollectVO();

        normalCollect.setName("正常运转");
```

```
normalCollect.setValue(Calculator.getRate(normalDevicesCount,totalDevices
Count));
        result.add(normalCollect);

        DeviceStatusCollectVO disconnectCollect = new
DeviceStatusCollectVO();
        disconnectCollect.setName("离线");

disconnectCollect.setValue(Calculator.getRate(disconnectCounts,totalDevic
esCount));
        result.add(disconnectCollect);

        DeviceStatusCollectVO alarmCollect = new DeviceStatusCollectVO();
        alarmCollect.setName("报警");

alarmCollect.setValue(Calculator.getRate(alarmCounts,totalDevicesCount));
        result.add(alarmCollect);

        return result;
    }
}
```

（2）DeviceController新增方法

```
package com.yikekong.controller;
import com.yikekong.service.ReportService;
import com.yikekong.vo.DeviceStatusCollectVO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;

/**
 * 报表控制器类
 */
@RestController
@RequestMapping("/report")
public class ReportController {

    @Autowired
    private ReportService reportService;


    /**
     * 获取设备状态分布
     * @return
     */
    @GetMapping("/statusCollect")
    public List<DeviceStatusCollectVO> getStatusCollect(){
        return reportService.getStatusCollect();
    }

}
```

1.3.3 实时监控数据

（1）创建vo类 MonitorVO

```
/**
 * 实时监控数据
 */
@Data
public class MonitorVO implements Serializable{
    /**
     * 设备数量
     */
    private Long deviceCount;
    /**
     * 告警设备数
     */
    private Long alarmCount;
}
```

（2）ReportController新增方法

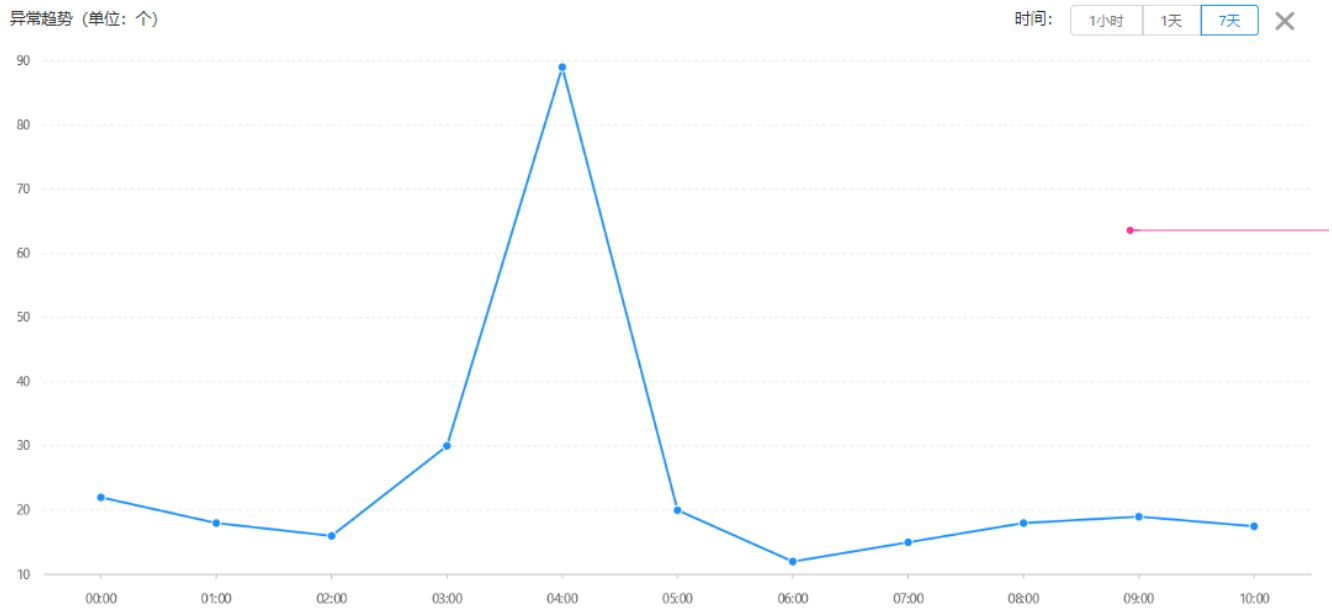
```
@Autowired
private ESRepository esRepository;

/**
 * 获取实时监控数据
 * @return
 */
@GetMapping("/monitor")
public MonitorVO getMonitorData(){
    MonitorVO monitor = new MonitorVO();
    monitor.setDeviceCount(esRepository.getAllDeviceCount());
    monitor.setAlarmCount(esRepository.getAlarmCount());
    return monitor;
}
```

2. 异常告警趋势报表展示

2.1 需求分析

系统要能够根据当前1小时内（每分钟第一次数据，当前分钟无数据，显示上一次数据）、当前1天内（每小时第一次数据，当前小时无数据，显示上一次数据）、当前7天内（7天小时数据的展示），汇总出每一个时间点上报的异常总数，然后前端根据这些数据以折现图的形式进行展现。

具体时间筛选和展现的格式如下：



横轴显示时间，纵轴显示的是告警次数汇总值数量。

2.2 实现思路

（1）在InfluxDBRepository实现具体的从InfluxDB中查询数据的通用方法

（2）定义需要前端需要展示的数据格式类及从InfluxDB中检索出的具体pojo类

（3）在服务层实现数据转化，转化成前端需要的数据格式

（4）定义对应的Controller调用服务层代码拉取数据返回给前端方便展示

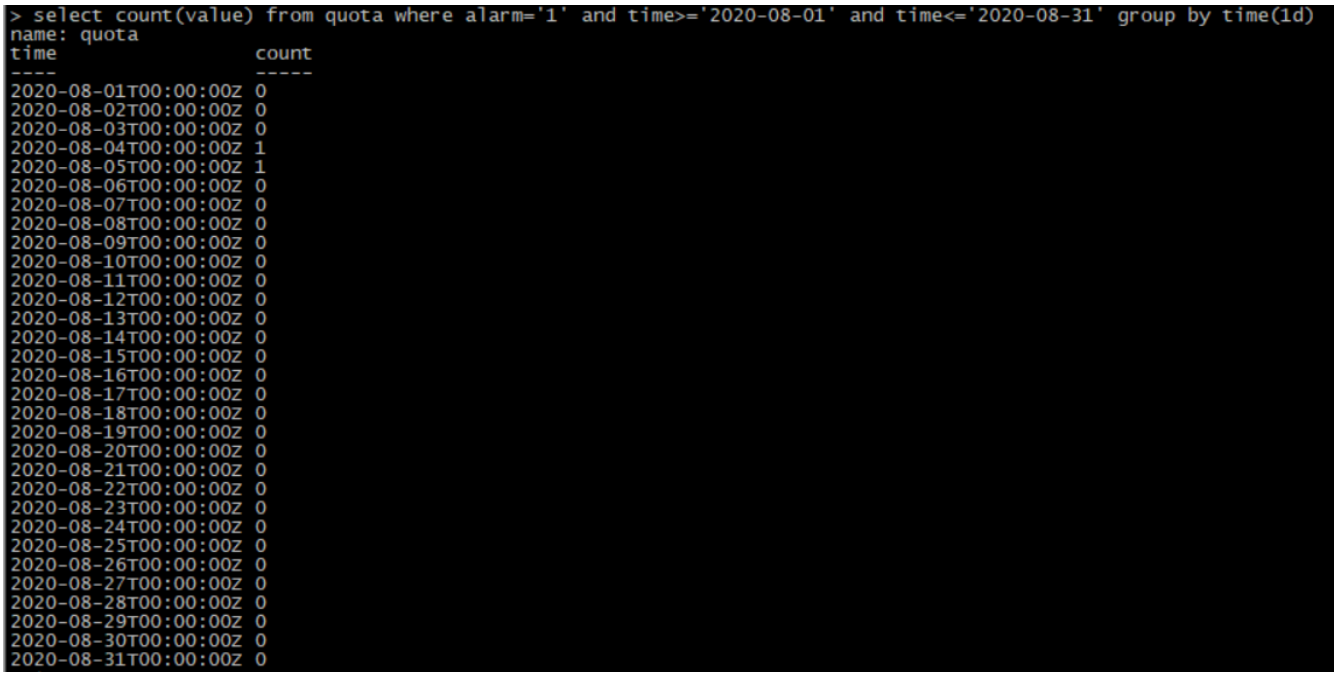
2.3 代码实现

2.3.1 异常数量分组统计

（1）编写查询语句

```
select count(value) from quota where alarm='1' and time>='2020-08-01' and
time<='2020-08-31' group by time(1d)
```

得到运行结果



这里需要说明的是：

在要执行的sql里的group by分组里使用了InfluxDB的time时间函数

- group by time(1m):按每分钟进行分组汇总
- group by time(1h):按每小时进行分组汇总
- groupby time(1d):按每天进行分组汇总

之后配合select中的count函数就可以获取到具体时间维度里汇总里的总数了，这样就能获取到我们想要的汇总数据了。

InfluxDB除了会根据sql语句返回pointValue也会同时返回每一条数据对应的time时间列。

（2）定义从InfluxDB中查询结果的pojo类：

```java
package com.yikekong.dto;

import lombok.Data;
import org.influxdb.annotation.Column;
import org.influxdb.annotation.Measurement;
import java.io.Serializable;

/**
 * 趋势指标点
 */
@Data
@Measurement(name = "quota")
public class TrendPoint implements Serializable{

    /**
     * 时间
     */
    @Column(name = "time")
    private String time;

    /**
     * 时间点数据
     */
    @Column(name = "pointValue")
    private Integer pointValue;
}
```

（3）在ReportService接口中定义获取异常指标趋势的方法：

```java
/**
 * 获取异常趋势指标
 * @param start 开始时间 yyyy-MM-dd HH:mm:ss
 * @param end 结束时间 yyyy-MM-dd HH:mm:ss
 * @param type 时间统计类型(1:60分钟之内,2:当天24小时,3:7天内)
 * @return
 */
List<TrendPoint> getAlarmTrend(String start, String end, int type);
```

在实现类ReportServiceImpl中实现该方法：

```
@Autowired
private InfluxDBRepository influxDBRepository;

@Override
public List<TrendPoint> getAlarmTrend(String start, String end, int type)
{
    StringBuilder sbSql = new StringBuilder("select count(value) as
pointValue from quota where alarm='1' and time>='");
    sbSql.append(start);
    sbSql.append("' and ");
    sbSql.append("time<=");
    sbSql.append("'");
    sbSql.append(end);
    sbSql.append("'");
    sbSql.append(" group by ");
    if(type == 1){
        sbSql.append("time(1m)");
    }else if(type == 2){
        sbSql.append("time(1h)");
    }else if(type == 3){
        sbSql.append("time(1d)");
    }
    List<TrendPoint> trendPointList =
influxDBRepository.query(sbSql.toString(), TrendPoint.class);
    for(TrendPoint trendPoint:trendPointList){
        trendPoint.setTime( formatTime(trendPoint.getTime(),type)  );
    }
    return trendPointList;
}


/**
 * 格式化日期串
 * @param time
 * @param type
 * @return
 */
private String formatTime(String time,int type){


    LocalDateTime localTime =
```

```
LocalDateTime.parse(time,DateTimeFormatter.ISO_OFFSET_DATE_TIME);

    if(type == 1){
       return  localTime.getMinute()+"";
    }else if(type == 2){
        return localTime.getHour()+"";
    }else if(type == 3){
        return
localTime.getMonthValue()+"月"+localTime.getDayOfMonth()+"日";
    }
    return time;
}
```

2.3.2 报表数据封装

（1）定义前端展示需要的VO类，代码如下：

```java
package com.yikekong.vo;

import lombok.Data;

import java.io.Serializable;
import java.util.List;

/**
 * 趋势数据
 */
@Data
public class TrenCollectVO implements Serializable {


    /**
     * X轴
     */
    private List<String> xdata;

    /**
     * 数据
     */
    private List<Long> series;

}
```

（2）在ReportController中添加方法调用getAlarmTrend获取数据并做转换返回给前端

```
/**
 * 获取告警趋势
 * @return
 */
@GetMapping("/trend/{startTime}/{endTime}/{type}")
public TrenCollectVO getQuotaTrendCollect(
        @PathVariable String startTime, @PathVariable  String endTime,
@PathVariable  Integer type){
    List<TrendPoint> trendPoints =
reportService.getAlarmTrend(startTime,endTime,type);
    if(trendPoints == null) return null;
    TrenCollectVO result = new TrenCollectVO();
    result.setSeries(Lists.newArrayList());
    result.setXdata(Lists.newArrayList());
    trendPoints.forEach(t->{
        //设置X轴
        result.getXdata().add(
                LocalDateTime.parse(t.getTime(),
DateTimeFormatter.ISO_OFFSET_DATE_TIME)
                        .format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss")));
        //设置数据
        result.getSeries().add(t.getPointValue().longValue());
    });
    return result;
}
```

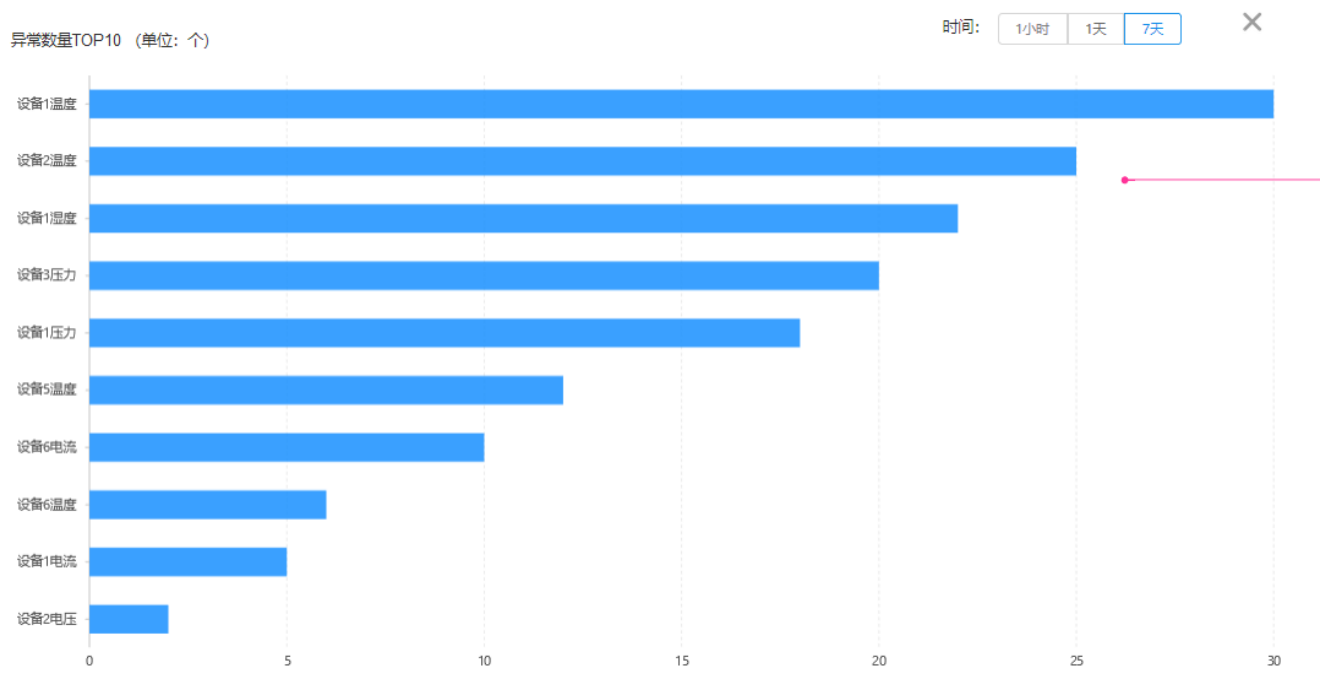
测试：

http://127.0.0.1:9093/report/trend?start=2020-08-01&end=2020-08-31&type=3

3. 告警次数top10设备指标报表展示

3.1 需求分析

系统需要展示出当前一小时内、当前1天内、当前7天内的告警次数最多的前10的设备及指标，以柱状图展示，展示效果如下：



其中纵轴展示的是设备编号+指标名称，横轴用来展示告警次数，按照从多到少进行排列。

3.2 实现思路

（1）定义需要前端需要展示的数据格式类及从InfluxDB中检索出的具体pojo类

（2）在AlarmService服务接口里定义获取在一定时间范围内的top10告警设备指标，在实现类里进行实现

（3） 在Controller中调用Service中的方法获取数据返回给前端

3.3 代码实现

3.3.1 告警次数top10查询

（1）首先编写查询语句，根据设备ID和指标ID分组查询报警数量

```
select count(value) as heapValue
  from quota
  where alarm='1'  and time>='2020-08-01' and time<='2020-08-31'
  group by deviceId,quotaId
```

注意：在标准的SQL语法中，分组的条件是必须出现在select字句后边的，而influxQL不能那么写。

上边的语句运行效果如下：

```
> select count(value) as heapvalue from quota where alarm='1'  and time>='2020-08-01' and time<='2020-08-31' group by deviceId,quotaId
name: quota
tags: deviceId=, quotaId=
time                 heapvalue
----                 ---------
2020-08-01T00:00:00Z 2

name: quota
tags: deviceId=123, quotaId=111
time                 heapvalue
----                 ---------
2020-08-01T00:00:00Z 1

name: quota
tags: deviceId=456, quotaId=222
time                 heapvalue
----                 ---------
2020-08-01T00:00:00Z 1
```

（2）上边的结果其实是我们要的中间结果，我们还应该在此结果基础上得到前10的数据，需要用到top函数

```
select top(heapValue,deviceId,quotaId,quotaName,10) as heapValue
from  (
  select count(value) as heapValue
  from quota
  where alarm='1'  and time>='2020-08-01' and time<='2020-08-31'
  group by deviceId,quotaId  )
order by desc
```

top函数第一个参数是需要排序的字段，最后一个参数是得到的结果数量，中间的参数是需要列出的其它数据。

上边的语句运行结果如下：

```
name: quota
time                 heapValue deviceId quotaId
----                 --------- -------- -------
2020-08-01T00:00:00Z 2
2020-08-01T00:00:00Z 1         123      111
2020-08-01T00:00:00Z 1         456      222
```

3.3.2 报表数据查询与封装

（1）定义top10对应的pojo类

```
package com.yikekong.dto;
import lombok.Data;
import org.influxdb.annotation.Column;
import org.influxdb.annotation.Measurement;
import java.io.Serializable;
/**
 * 累积指标
 */
@Data
@Measurement(name = "quota")
public class HeapPoint implements Serializable{
    @Column(name = "deviceId")
    private String deviceId;
    @Column(name = "heapValue")
    private Double heapValue;
    @Column(name = "quotaId")
    private String quotaId;
    @Column(name = "quotaName")
    private String quotaName;
}
```

（2）在ReportService服务接口里定义获取数据的方法

```
/**
 * 获取一定时间范围之内的报警次数最多的设备指标
 * @return
 */
List<HeapPoint> getTop10Alarm(String startTime, String endTime);
```

在ReportServiceImpl实现类里实现该方法：

```
@Override
public List<HeapPoint> getTop10Alarm(String startTime, String endTime) {
    StringBuilder sbSql =
            new StringBuilder("select
top(heapValue,deviceId,quotaId,quotaName,10) as heapValue " +
                    " from(select count(value) as heapValue from quota
where alarm='1' ");
    sbSql.append("and time>='");
    sbSql.append(startTime);
    sbSql.append("' and time<='");
    sbSql.append(endTime);
    sbSql.append("' group by deviceId,quotaId) order by desc");

    return influxDBRepository.query(sbSql.toString(),HeapPoint.class);
}
```

这里sql语句的具体实现思路是：sql子查询里获取的是一定时间范围之内根据设备和指标分组之后的告警总数，然后在外部查询里用top函数对倒序之后的结果集取前十就能获取到我们想要的数据。

（3）在ReportController里添加获取前十数据的方法，在该方法里调用服务层的方法返回给前端：

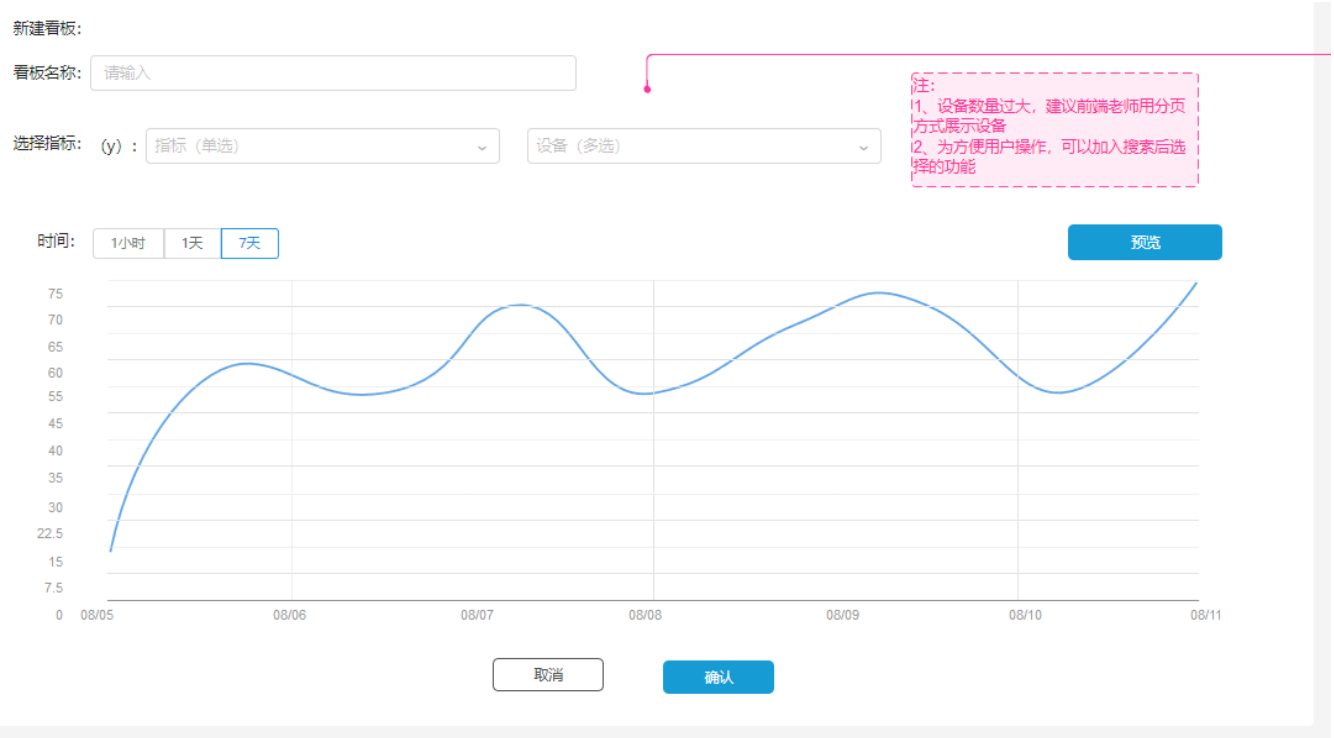
```
/**
 * 获取一定时间范围之内的告警次数前10最多的设备指标
 * @param startTime
 * @param endTime
 * @return
 */
@GetMapping("/top10Alarm/{startTime}/{endTime}")
public List<HeapPoint> getTop10Alarm(@PathVariable String startTime,
@PathVariable String endTime){
    return reportService
                    .getTop10Alarm(startTime,endTime);
}
```

4. 自定义看板的添加和展示

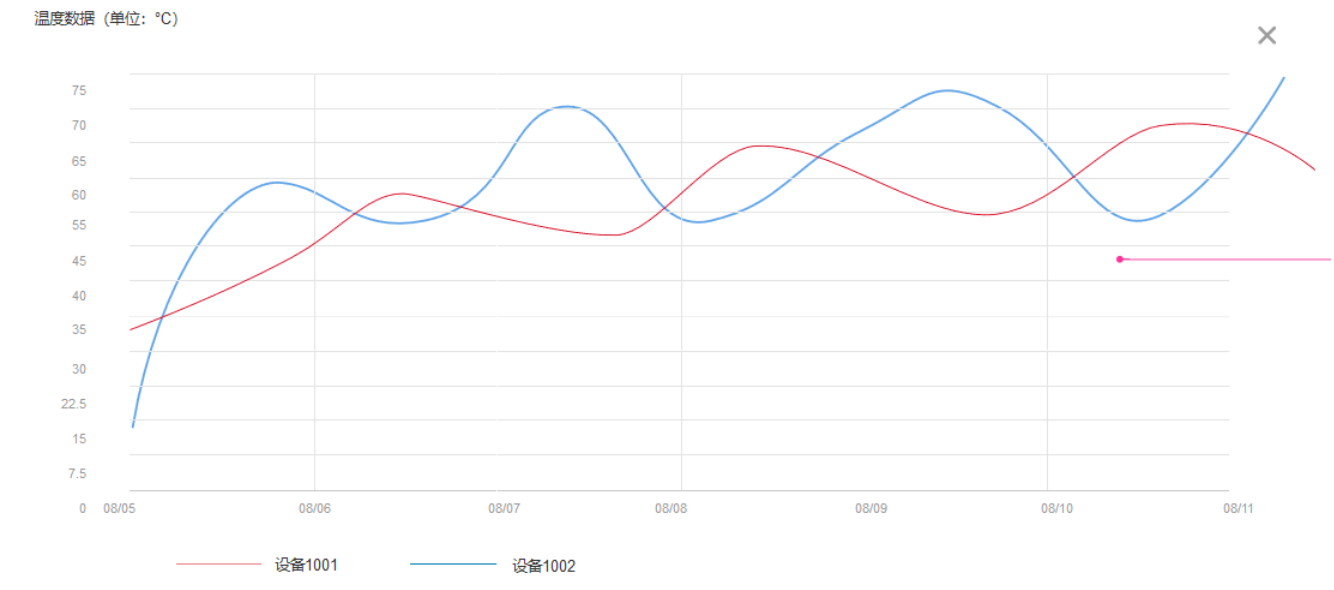
4.1 需求分析

系统管理人员能够在系统里可以添加自定义的看板来展示数据，该自定义看板是用户自己选择的指标，及根据指标自主选取上报该指标的一些设备，以方便跟踪和观察该指标下相关设备的趋势，展示的趋势也是多个维度的，按照小时展示当前1小时内60分钟的指标相关设备趋势；按照1天24小时展示当天每小时的指标设备趋势；按照7天展示最近7天之内24小时的指标设备趋势。

添加看板的原型图如下：



在首页展示已经添加的看板原型设计如下：



横轴展示的是时间，纵轴的数据是每个时间点内的设备首次上报指标数据。

4.2 实现思路

（1）在系统添中加看板时只能选择数值型指标，从mysql数据库中分页加载所有数值型指标，指标是单选

（2）添加看板时选择完指标之后，在选择设备时只能选择之前已经选择好的指标下的上报过该指标的设备，设备是多选，最多选择10个设备

（3）在向前端提供数据时，先从mysql中查询出该面板的指标Id，及对应的所有设备编号，根据前端传入的时间类型从InfluxDB中循环获取相应时间段内，对应时间的所有设备指标的首次指标数值

（4）将所有数据组合成前端想要的数据格式返回

4.3 代码实现

4.3.1 获取数值型指标列表

（1）从mysql数据库中分页加载所有数值型指标，在QuotaService接口中定义方法

```
IPage<QuotaEntity> queryNumberQuota(Long page, Long pageSize);
```

在QuotaServiceImpl实现类中实现该方法：

```
@Override
public IPage<QuotaEntity> queryNumberQuota(Long page, Long pageSize) {
    Page<QuotaEntity> pageResult = new Page<>(page,pageSize);
    LambdaQueryWrapper<QuotaEntity> wrapper = new LambdaQueryWrapper<>();
    wrapper
        .eq(QuotaEntity::getValueType,"Long")
        .or()
        .eq(QuotaEntity::getValueType,"Integer")
        .or()
        .eq(QuotaEntity::getValueType,"Double");
    wrapper.orderByDesc(QuotaEntity::getCreateTime);
    return this.page(pageResult,wrapper);
}
```

（2）在QuotaController中添加获取该数据的接口，调用上述方法返回数据给前端

```
/**
     * 分页获取数值型指标
     * @param page
     * @param pageSize
     * @return
     */
@GetMapping("/numberQuota")
public Pager<QuotaEntity> queryNumberQuota(@RequestParam(value =
"page",required = false,defaultValue = "1") Long page,
                                           @RequestParam(value =
"pageSize",required = false,defaultValue = "10") Long pageSize){
    return new Pager<>(quotaService.queryNumberQuota(page,pageSize));
}
```

4.3.2 获取某指标相关的设备

（1）查询语句

由于设备数量多，所以我们需要采用分页查询。语句如下：

```
select distinct(deviceId) as deviceId
from(select deviceId,value from quota where quotaId='111' group by
deviceId,quotaId)  limit 10 OFFSET 0
```

编写语句获取设备数量，用于分页

```
select count(distinct(deviceId)) as deviceCount from(select
deviceId,value from quota where quotaId='' group by deviceId,quotaId)
```

（2）在ReportService接口中定义通过指标ID获取关联设备的方法

```
/**
     * 通过指标获取关联设备
     * @param quotaId
     * @return
     */
Pager<String> getDeviceByQuota(Long page,Long pageSize,String quotaId);
```

在ReportServiceImpl实现类中实现该方法：

```
@Override
public Pager<String> getDeviceByQuota(Long page, Long pageSize, String
quotaId) {
    StringBuilder sbSql = new StringBuilder();
    sbSql.append("select count(distinct(deviceId)) as deviceCount
from(select deviceId,value from quota where ");
    sbSql.append("quotaId='");
    sbSql.append(quotaId);
    sbSql.append("' group by deviceId,quotaId)");
    DeviceCount deviceCount = this.getCountBySql(sbSql.toString());
    Long totalCount=0l;
    if(deviceCount!=null){
        totalCount =deviceCount.getDeviceCount();
    }

    Pager<String> result = new Pager<>(0L,0L);
    result.setItems(Lists.newArrayList());
    result.setPage(0);
    if(totalCount <= 0){
        return result;
    }

    sbSql.setLength(0);
    sbSql.append("select distinct(deviceId) as deviceId from(select
deviceId,value from quota where quotaId='");
    sbSql.append(quotaId);
    sbSql.append("' group by deviceId,quotaId) limit ");
    sbSql.append(pageSize);
    sbSql.append(" OFFSET ");
    sbSql.append((page - 1) * pageSize);
    List<QuotaInfo> queryResults =
influxDBRepository.query(sbSql.toString(),QuotaInfo.class);
    if(queryResults == null || queryResults.size() <= 0){
        return result;
    }
    if(pageSize>totalCount)
        pageSize=totalCount;
    result = new Pager(totalCount,pageSize);


    result.setItems(queryResults.stream().map(q-
```

```
>q.getDeviceId()).collect(Collectors.toList()));
    result.setPage(page);

    return result;
}



/**
 * 获取count数量的方法封装
 * @param sql
 * @return
 */
private DeviceCount getCountBySql(String sql){
    List<DeviceCount> totalResult =
influxDBRepository.query(sql,DeviceCount.class);
    if(totalResult == null || totalResult.size() <= 0){
        return null;
    }
    return totalResult.get(0);
}
```

（3）在ReportController中添加获取该数据的接口，调用上述方法返回数据给前端：

```
/**
 * 通过指标查询设备列表
 * @param quotaId
 * @return
 */
@GetMapping("/devices")
public Pager<String> getDeviceByQuota(
  @RequestParam(value = "page",required = false,defaultValue = "1") Long
page,
  @RequestParam(value = "pageSize",required = false,defaultValue = "10")
Long pageSize,
  @RequestParam(value = "quotaId")String quotaId){
    return reportService.getDeviceByQuota(page,pageSize,quotaId);
}
```

4.3.3 按设备指标查询指标序列

（1）编写查询语句

```
select first(value) as pointValue from quota where time>='2020-09-01
00:00:00' and time<='2020-09-07 00:00:00' and quotaId='1' and
deviceId='123456' group by time(1d)
```

（2）在ReportService接口中定义获取一段时间之内某一指标下的趋势指标点集合的方法

```
/**
 * 获取指标趋势
 * @param start 开始时间 yyyy-MM-dd HH:mm:ss
 * @param end 结束时间 yyyy-MM-dd HH:mm:ss
 * @param quotaId 指标Id
 * @param type 时间统计类型(1:60分钟之内,2:当天24小时,3:7天内)
 * @param deviceId 设备编码
 * @return
 */
List<TrendPoint> getQuotaTrend(
                String start, String end, String quotaId,String
deviceId,int type);
```

在ReportServiceImpl实现类中实现该方法

```
/**
 * 获取指标趋势
 * @param start
 * @param end
 * @param quotaId
 * @param type
 * @return
 */
@Override
public List<TrendPoint> getQuotaTrend(String start, String end, String
quotaId, String deviceId, int type){
    StringBuilder sbSql = new StringBuilder("select first(value) as
pointValue from quota");
    sbSql.append(getWhereStr(start,end,quotaId,deviceId));
    sbSql.append(" group by ");
    if(type == 1){
        sbSql.append("time(1m)");
    }else if(type == 2){
        sbSql.append("time(1h)");
    }else if(type == 3){
        sbSql.append("time(1d)");
    }
    List<TrendPoint> trendPointList =
influxDBRepository.query(sbSql.toString(), TrendPoint.class);
    for(TrendPoint trendPoint:trendPointList){
        trendPoint.setTime( formatTime(trendPoint.getTime(),type)  );
    }
    return trendPointList;
}


/**
 * 构造where条件
 * @param start
 * @param end
 * @param quotaId
 * @param deviceId
 * @return
 */

private String getWhereStr(String start, String end, String
```

```
quotaId,String deviceId){
    StringBuilder whereStr= new StringBuilder(" where time>='");
    whereStr.append(start);
    whereStr.append("' and ");
    whereStr.append("time<=");
    whereStr.append("'");
    whereStr.append(end);
    whereStr.append("' and quotaId='");
    whereStr.append(quotaId);
    whereStr.append("' ");
    whereStr.append(" and deviceId='");
    whereStr.append(deviceId);
    whereStr.append("'");
    return whereStr.toString();
}
```

4.3.4 空缺数据补全

（1）ReportServiceImpl新增私有方法

```
/**
 * 补充数据
 * @param trendPointList
 * @return
 */
private List<TrendPoint> replenish(List<TrendPoint> trendPointList){

    //获取集合中，第一个不为null的值
    Double previousValue=null;//上一个值
    for (TrendPoint trendPoint:trendPointList ){
        if(trendPoint.getPointValue()!=null){//如果不为空
            previousValue=trendPoint.getPointValue();
            break;
        }
    }
    if(previousValue==null){
        previousValue=0d;
    }
    // 数据填充
    for (TrendPoint trendPoint:trendPointList ){
        if(trendPoint.getPointValue()==null){//如果为空
            trendPoint.setPointValue(previousValue);
        }
        previousValue=trendPoint.getPointValue();
    }
    return trendPointList;
}
```

（2）修改getQuotaTrend方法的返回值

```
List<TrendPoint> trendPointList =
replenish(influxDBRepository.query(sbSql.toString(), TrendPoint.class));
```

4.3.5 报表数据封装

前端要求返回的格式如下：

```
{
  "xdata": [
    "8月1日",
    "8月2日",
    "8月3日"
  ],
  "series": [
    {
      "name": "设备速度",
      "data": [
        70,
        90,
        100
      ]
    },
    {
      "name": "设备温度",
      "data": [
        50,
        60,
        80,
      ]
    }
  ],
  "name": "面板名称"
}
```

（1）定义vo用于封装某个指标的数据序列

```
package com.yikekong.vo;
import lombok.Data;
import java.io.Serializable;
import java.util.List;
/**
 * 面板指标数据
 */
@Data
public class BoardQuotaData implements Serializable{
    /**
     * 指标名称
     */
    private String name;
    /**
     * 指标数据
     */
    private List<Double> data;
}
```

（2）定义vo，用于封装整个报表的数据结构

```java
package com.yikekong.vo;
import lombok.Data;
import java.io.Serializable;
import java.util.List;
/**
 * 面板VO对象
 */
@Data
public class BoardQuotaVO implements Serializable{

    /**
     * x轴数据
     */
    private List<String> xdata;

    /**
     * Y轴数据
     */
    private List<BoardQuotaData> series;

    /**
     * 面板名称
     */
    private String name;
}
```

（3）在ReportService接口中定义获取面板指标数据的方法

```
/**
     * 获取面板数据
     * @param id
     * @param start
     * @param end
     * @param type
     * @return
     */
BoardQuotaVO getBoardData(Integer id, String start, String end, Integer
type);

    /**
     * 获取面板数据
     * @param quotaId
     * @param deviceIds
     * @param start
     * @param end
     * @param type
     * @return
     */
    BoardQuotaVO getBoardData(String quotaId,List<String>
deviceIds,String start,String end,Integer type);
```

在ReportServiceImpl实现类中实现该方法

```
@Autowired
private BoardService boardService;

@Autowired
private QuotaService quotaService;

@Override
public BoardQuotaVO getBoardData(Integer id, String start, String end,
Integer type) {
    //从mysql中获取面板实体数据
    BoardEntity boardEntity = boardService.getById(id);
    if(boardEntity == null) return null;

    List<String> deviceIdList =
Arrays.asList(boardEntity.getDevice().split(","));
    BoardQuotaVO boardQuotaVO  = getBoardData(boardEntity.getQuota() +
"",deviceIdList,start,end,type);
    boardQuotaVO.setName(boardEntity.getName());

    return boardQuotaVO;
}

@Override
public BoardQuotaVO getBoardData(String quotaId,List<String>
deviceIds,String start,String end,Integer type){
    if(deviceIds == null || deviceIds.size() <= 0) return new
BoardQuotaVO();
    BoardQuotaVO boardQuotaVO = new BoardQuotaVO();
    //获取每一个设备的所有指标数据
    for (String deviceId:deviceIds) {
        List<TrendPoint> trendPoints =
getQuotaTrend(start,end,quotaId,deviceId,type);
        if(trendPoints == null || trendPoints.size() <= 0){
            continue;
        }
        //构建X轴数据
        if(boardQuotaVO.getXdata() == null){
            boardQuotaVO.setXdata(trendPoints.stream().map(t->
t.getTime()).collect(Collectors.toList()));

        }
```

```
        //如果Y轴数据为空，则设置空集合
        if(boardQuotaVO.getSeries() == null){
            boardQuotaVO.setSeries(Lists.newArrayList());
        }

        //设置Y轴数据
        BoardQuotaData boardQuotaData = new BoardQuotaData();
        QuotaEntity quotaEntity = quotaService.getById(quotaId);
        boardQuotaData.setName(deviceId + ":" +quotaEntity.getName());
        boardQuotaData.setData(trendPoints.stream().map(t-
>t.getPointValue()).collect(Collectors.toList()));
        boardQuotaVO.getSeries().add(boardQuotaData);
    }

    return boardQuotaVO;
}
```

该方法的主要逻辑是：先从mysql数据库中根据面板Id获取面板中包含的指标Id，及该指标下所有设备Id，然后根据设备Id依次循环获取设备指标数据，将最后获取到的所有指标数据组合成前端需要的数据返回。

（4）在ReportController调用方法将数据返回给前端

```java
/**
 * 获取面板数据
 * @param id
 * @param start
 * @param end
 * @param type
 * @return
 */
@GetMapping("/boardData/{id}/{start}/{end}/{type}")
public BoardQuotaVO getBoardData(@PathVariable Integer id,
                                 @PathVariable String start,
                                 @PathVariable String end,
                                 @PathVariable Integer type){
    return reportService.getBoardData(id,start,end,type);
}


/**
 * 报表预览
 * @param previewVO
 * @return
 */
@PostMapping("/preview")
public BoardQuotaVO getPreviewData(@RequestBody PreviewVO previewVO){
    return
reportService.getBoardData(previewVO.getQuotaId(),previewVO.getDeviceIdLi
st(),previewVO.getStart(),previewVO.getEnd(),previewVO.getType());
}
```