大批量接口请求的前端优化

大批量接口请求的前端优化

背景

> 接到一个需求，需要做一个运维的功能，对公司下平台子系统做一个开关界面，要求可以大批量对系统进行修改。这样大批量的修改，如果有个老哥手一抖点了全选，那可能同时会发送几十上百个接口，对浏览器和服务器都会是一个比较大的负担。
>
> 相似的应用场景还有：文件批量上传等
>
> 使用技术栈：react hooks + antd4

分析

> 维护一个任务池，先默认发出最大并发数量的接口请求，一个萝卜一个坑，当上一个请求结束（无论成功失败），则让下一个请求进入任务队列，之道所有请求列表都请求完成，返回这个任务池的最终结果。

部分代码实现

主要业务逻辑

```
// ... 点击确定，先进行前置校验
const onSubmitHandle = (event) => {
    event.preventDefault(); // 防止触发原生submit，导致浏览器页面刷新
    validateFields((err, values) => {
        // 校验通过
        if(!err){
            // 通过遍历组装参数
            const arrParam = concateDynamicParams(values);
            concurrentControl(arrParm, 5)
            .then(res)=> {
                // 获取成功列表和失败列表
                const { sucSystem, faildSystem } = getSucAndFailedResult(res);
                const content = (
                  <p>
                    {'修改成功的系统：'}
                    <br />
                    {`${sucSystem}`}
                    <br />
                    {'修改失败的系统：'}
```

```
                    <br />
                    {`${faildSystem}`}
                  </p>
                );
                Modal.info({
                    title: "提示",
                    content,
                    oKText: "确定",
                })
                // 取消loading，重置列表等操作...
            }
        }
    })
}

```

维护任务池，控制并发数

```
const concurrentControl = (arrParam, maxNum = 10) =>
        new Promise((resolve) => {
        // 无任务数据时
        if(!arrParam.length){
            resolve([]);
            return;
        }
        const results = []; // 最终的结果集合，用于保存接口返回的信息
        let index = 0; // 用于记录下一个接口下标
        let count = 0; // 用于记录发送了多少条数据



        const request = async () => {
            if(index === arrParam.length) return; // 出口
            const i = index; // 当前下标（备份）
            // 组装请求路径与参数
            const url = arrParam[i].url;
            const param = arrParam[i].data;
            index++; // 预先记录下一个请求的下标
            try{
                // 发送请求
                const resp = await dispatchSqlGuardSwitchProcess(url, param);
                result[i] = resp; // 记录返回结果
            } catch( error ){
                result[i] = error;
            } finally {
```

```
                count++ ; // 当前接口完成，下一个接口调用开始
                if(count === arrParam.length){ // 全部任务池中的接口都调用完毕
                    resolve(results)
                }
                request()(); // 继续下一个请求
            }
        }

        // 初始化请求数
        const maxParallelRequests = Math.min(maxNum, arrParam.length);
        for(let i = 0; i < maxParallelRequests; i++){
            request()
        }

    })
```