# 前端无侵入实现防止重复提交请求技术

最近在代码发布测试的过程中，我发现有些请求非常的消耗服务器资源，而系统测试人员因为响应太慢而不停的点击请求。我是很看不惯系统存在不顺眼的问题，做事喜欢精益求精，也很喜欢和别人争论技术，有时候硬要争得你死我活。

实在看不下去系统存在这个问题，下定决心好好整改一波，我们系统几乎都是使用Ajax技术发请求，使用的是针对我们系统特点对jQuery的post方法进行了封装的newpost方法，这个方法扩展了异常响应，登陆超时，token验证，加载时渲染等自动完成功能，参数不变，也是我写的。由于系统模块很多，功能很强大，代码也很多，为了统一使用此方法，花了不少时间将post修改成newpost。这次为了限制请求次数这个小功能，我请示了领导并把初步的方案说与他听，他同意了我做此次修改。

我调查了几个有防止请求重发控制的页面，几乎都是发起请求后到响应这段时间对按钮置灰警用处理，这种方案有很多的缺点，如：

1. 长时间未响应的请求给人不友好的体验
2. 需要针对每个按钮做请求前后的控制，会因为处理各种小问题而变得难以维护
3. 业务与控制逻辑混合在一起使得函数功能不单一

我最初的想法就是想找到一种类似于面向切面的无侵入页面的方式实现请求限制，因此想到了如下方案：

1. 只控制用户最后一次的请求
2. 第一次请求未响应之前一分钟之内不发相同的请求
3. 如果前一次请求得到了响应，那么第二次相同的请求也不做限制

后来发现这种方案行不通，主要有以下几个方面的原因：

1. 大多数页面每交互一次都会发起多个不同的请求。
2. 没个请求响应的时间也不一样，可能后发的请求先响应

我把这种想法告诉了导师，经过一番争论后认为我无法通过不侵入的方式实现这样的功能，按照他的想法，我得修改整个项目有请求的代码，完全低估了我对JS的了解，后来我独自优化了方案，不在只控制最后一个请求，代码实现的大纲如下：

发起POST请求前处理逻辑

1. 获取全局上下文中的存储请求信息的数组变量
2. 遍历请求响应数组
3. 清理大于两分钟未响应的请求，防止页面长时间不响应
4. 找到了与本次相同的请求
5. 如果请求小于200ms，则判断为系统行为，直接将环境和回调函数压入已有请求的调用栈，否则替换之前的请求的调用栈
6. 不发请求
7. 没有找到与本次相同的请求
8. 将请求信息、时间、调用环境和回调函数压进请求响应数组，请求URL和条件con作为唯一标识，调用栈也是一个由当前环境和回调函数组成的数组，使用数组解决相同的请求（url和con都一样）
9. 发起请求

请求响应后处理逻辑

1. 获取全局上下文中的存储请求信息的数组变量
2. 遍历请求响应数组
3. 找到了与此响应相同的请求
4. 遍历此请求的调用栈并依次调用回调函数
5. 删除已经处理的请求
6. 没有找到与此响应相同的请求则什么都不做

具体实现：

/\*@description 自定义事件

\*@method \_costomEvent

\*@for muphy

\*/

muphypost: function (url, condition, callback, type, option) {

if (typeof condition === 'function') {

option = type;

type = callback;

callback = condition;

condition = {};//jQuery:condition = undefine;

}

let that = this, reqUrl = (url + JSON.stringify(condition)).substr(0, 1000);

//判断是否需要发起请求

let hasReq = beforePost();

if (hasReq === true) {

return that;

}

//新增请求Token

muphy.addToken(condition);

//借助jQuery发请求

$.post(url, condition, function (data, status) {

afterPost(data, status);

}, type);

return that;

//请求前处理函数

function beforePost() {

//获取历史请求

let postHistory = muphy.postHistory, now = new Date();

//遍历请求响应数组，找到与此请求相同的请求

for (let i = 0; i < postHistory.length; i++) {

if (now - postHistory[i].\_startTime > 120000) {

//清理大于两分钟未响应的请求

postHistory.splice(i, 1);

} else if (postHistory[i].\_reqUrl === reqUrl) {

//找到了就更新请求时间，并将执行环境和回调函数放入请求响应数组的调用栈；

//小于200ms判定为系统行为直接将环境和回调函数压入已有请求的调用栈，

//否则替换之前的请求

if (now - postHistory[i].\_startTime < 200) {

postHistory[i].\_startTime = now;

postHistory[i].\_handles.push({ \_callback: callback, \_this: that });

} else {

postHistory[i].\_handles = [{ \_callback: callback, \_this: that }];

}

//返回true 表示本次不发请求

return true;

}

};

//没找到与此相同的请求时将请求信息、时间、调用环境和回调函数压进请求响应数组

postHistory.push({ \_reqUrl: reqUrl, \_startTime: new Date(), \_handles: [{ \_callback: callback, \_this: that }] });

//返回false 表示本次需要发请求

return false;

}

//请求后处理函数

function afterPost(data, status) {

//获取请求响应数组

let postHistory = muphy.postHistory;

//遍历请求响应数组，找到与此请求相同的请求

for (let i = 0; i < postHistory.length; i++) {

if (postHistory[i].\_reqUrl === reqUrl) {

//找到相同的请求，遍历此请求的调用栈并调用

muphy.each(postHistory[i].\_handles, function () {

this.\_callback.call(this.\_this, data, status);

});

//调用完成删除已经处理的请求

postHistory.splice(i, 1);

break;

}

}

}

}