一般我们使用GET方法实现EXCEL导出，但是如果存在参数较大或者GET方式不支持的参数就需要使用POST方式导出，下面就来涨涨EXCEL导出的姿势吧。

1、创建一个POST接口请求导出数据：

|  |
| --- |
| */\*\*  \* 账号统计数据导出  \* \*/* @RequestMapping(value = "/exportByPlatform", method = RequestMethod.*POST*) public @ResponseBody Map<String, Object> exportByPlatform(@RequestBody Map<String, Object> params){  List<Integer> cycles = params.get("cycle") == null ? new ArrayList<>() : (List<Integer>)params.get("cycle");//统计周期  if(cycles.size() == 0){  cycles.add(0);  }   Map<String, Object> result = new HashMap<>();  result.put("ResultCode", "200");  result.put("ResultMsg", "Success");   Map<String, Object> query;  try{  query = initQuary(params);  }catch (Exception e){  return null;  }   List<Map<String, Object>> datas = dailyStatisticService.statisticExecute(query, 0, cycles.get(0));   Map<String, String> titles = new LinkedHashMap<>();  titles = initExcelTitle(titles);   try{  excelExportUtil.exportDataExcel(response, "消息数据平台时效统计", operateData(datas, cycles.get(0)), titles,1000);  }catch (Exception e){  result.put("ResultCode", "500");  result.put("ResultMsg", "failed!!! " + e.getCause());  }   return result; } |

2、生成EXCEL文件流，写入Response

|  |
| --- |
| */\*\*  \* 将数据写入到一个流中，供用户下载，即流形成的一个Excel文件（重写）  \*  \** ***@param*** *fileName 输出文件名  \** ***@param*** *list 存放的数据，此数据要写入到Excel表格  \** ***@param*** *titles 每列数据的标题&key  \** ***@param*** *maxRow 每个sheet中最大的数据行数，如果超过了，再新建一个sheet写入数据  \** ***@return*** *\*/* public void exportDataExcel (HttpServletResponse respose, String fileName, List<Map<String, Object>> list,  Map<String, String> titles, int maxRow) throws Exception{  //组装Excel  XSSFWorkbook wb = new XSSFWorkbook();   //计算sheet页数量  int listSize=list.size();  int sheetPage;  if(0<listSize%maxRow){  //则要建立的sheet数为  sheetPage=listSize/maxRow+1;  }else{  //若能整除，则为  sheetPage=listSize/maxRow;  sheetPage=(sheetPage==0?1:sheetPage);  }   //循环要建立的sheet的数量，为每一个sheet开始写入数据  int current = 0;//当前处理的数据index  int sheetCurrentCount = 0;//当前页签已经处理的数据量  for(int i=0;i<sheetPage;i++){  //创建sheet  XSSFSheet sheet =wb.createSheet("sheet" + i);   //第一行插入标题  XSSFRow title=sheet.createRow(0);   //标题样式  XSSFCellStyle cellStyleTitle=wb.createCellStyle();  cellStyleTitle.setFillPattern(XSSFCellStyle.*FINE\_DOTS*);  cellStyleTitle.setFillBackgroundColor(HSSFColor.ROYAL\_BLUE.*index*); //背景  cellStyleTitle.setFillForegroundColor(HSSFColor.ROYAL\_BLUE.*index*); //前景  cellStyleTitle.setWrapText(true); //自动换行  cellStyleTitle.setAlignment(HSSFCellStyle.*ALIGN\_CENTER*); // 水平居中   cellStyleTitle.setBorderBottom(HSSFCellStyle.*BORDER\_THIN*);//下边框      cellStyleTitle.setBorderLeft(HSSFCellStyle.*BORDER\_THIN*);//左边框      cellStyleTitle.setBorderTop(HSSFCellStyle.*BORDER\_THIN*);//上边框      cellStyleTitle.setBorderRight(HSSFCellStyle.*BORDER\_THIN*);//右边框    //标题字体样式  XSSFFont fontStyle=wb.createFont();  fontStyle.setBoldweight(HSSFFont.*BOLDWEIGHT\_BOLD*); //粗体  fontStyle.setFontHeightInPoints((short) 11); //字体大小   cellStyleTitle.setFont(fontStyle);   //插入标题  Set<String> keySet = titles.keySet();  Object[] keys = keySet.toArray();  /\*Arrays.sort(keys);\*/  for(int k = 0; k<keys.length; k++){  XSSFCell cell = title.createCell(k);  cell.setCellStyle(cellStyleTitle);  cell.setCellValue(titles.get(keys[k]+""));  sheet.setColumnWidth(k,(titles.get(keys[k]+"")).length() \* 512);  }   //内容样式  XSSFCellStyle cellStyleData=wb.createCellStyle();  cellStyleData.setFillBackgroundColor(HSSFColor.BLUE.*index*);  cellStyleData.setWrapText(true);   cellStyleData.setBorderBottom(HSSFCellStyle.*BORDER\_THIN*);//下边框      cellStyleData.setBorderLeft(HSSFCellStyle.*BORDER\_THIN*);//左边框     cellStyleData.setBorderRight(HSSFCellStyle.*BORDER\_THIN*);//右边框   //内容字体样式  XSSFFont fontStyleData=wb.createFont();  fontStyleData.setFontHeightInPoints((short)11);   cellStyleData.setFont(fontStyleData);   //数据插入  for(int d = current; d < list.size(); d++){  //超出每页最大量则在下一页保存  if(sheetCurrentCount == maxRow){  sheetCurrentCount = 0;  current = d;  break;  }   //装载数据  XSSFRow row = sheet.createRow(sheetCurrentCount+1);  for(int r = 0; r<keys.length; r++){  XSSFCell cell = row.createCell(r);  cell.setCellStyle(cellStyleData);  cell.setCellValue(list.get(d).get(keys[r])+"");   int width = (list.get(d).get(keys[r])+"").length() \* 256;  if(width > sheet.getColumnWidth(r)){  sheet.setColumnWidth(r,width);  }  }  sheetCurrentCount++;  }   }   //Excel文件写到输出流  OutputStream output=respose.getOutputStream();   respose.setHeader("Content-disposition", "attachment; filename="+fileName+".xlsx");  respose.setContentType("application/msexcel");   wb.write(output);   //关闭流  output.close(); } |

3、JS执行请求，处理返回的数据流

|  |
| --- |
| //导出 **var** doExport = **function**(){  **var** xhr = **new** XMLHttpRequest();//创建新的XHR对象  xhr.open('post', "**${**ctx**}**report/messageStatistics/exportByAccount");//指定获取数据的方式和url地址  xhr.setRequestHeader('Content-Type', 'application/json; charset=UTF-8')  xhr.responseType = 'blob';//以blob的形式接收数据，一般文件内容比较大  xhr.onload = **function**(e) {  **var** blob = **this**.response;//Blob数据  **if** (**this**.status == 200) {  **if** (blob && blob.size > 0) {  saveAs(blob, "消息数据账号时效统计.xlsx");//处理二进制数据，让浏览器认识它  }  }  };   xhr.send(getParams()) //post请求传的参数 } |

\*注意：saveAs()方法才是重点

|  |
| --- |
| **var** saveAs = saveAs || (**function**(view) {  "use strict";  // IE <10 is explicitly unsupported  **if** (**typeof** view === "undefined" || **typeof** navigator !== "undefined" && /MSIE [1-9]\./.test(navigator.userAgent)) {  **return**;  }  **var** doc = view.document  // only get URL when necessary in case Blob.js hasn't overridden it yet  , get\_URL = **function**() {  **return** view.URL || view.webkitURL || view;  }  , save\_link = doc.createElementNS("http://www.w3.org/1999/xhtml", "a")  , can\_use\_save\_link = "download" **in** save\_link  , click = **function**(node) {  **var** event = **new** MouseEvent("click");  node.dispatchEvent(event);  }  , is\_safari = /constructor/i.test(view.HTMLElement) || view.safari  , is\_chrome\_ios =/CriOS\/[\d]+/.test(navigator.userAgent)  , setImmediate = view.setImmediate || view.setTimeout  , throw\_outside = **function**(ex) {  setImmediate(**function**() {  **throw** ex;  }, 0);  }  , force\_saveable\_type = "application/octet-stream"  // the Blob API is fundamentally broken as there is no "downloadfinished" event to subscribe to  , arbitrary\_revoke\_timeout = 1000 \* 40 // in ms  , revoke = **function**(file) {  **var** revoker = **function**() {  **if** (**typeof** file === "string") { // file is an object URL  get\_URL().revokeObjectURL(file);  } **else** { // file is a File  file.remove();  }  };  setTimeout(revoker, arbitrary\_revoke\_timeout);  }  , dispatch = **function**(filesaver, event\_types, event) {  event\_types = [].concat(event\_types);  **var** i = event\_types.length;  **while** (i--) {  **var** listener = filesaver["on" + event\_types[i]];  **if** (**typeof** listener === "function") {  **try** {  listener.call(filesaver, event || filesaver);  } **catch** (ex) {  throw\_outside(ex);  }  }  }  }  , auto\_bom = **function**(blob) {  // prepend BOM for UTF-8 XML and text/\* types (including HTML)  // note: your browser will automatically convert UTF-16 U+FEFF to EF BB BF  **if** (/^\s\*(?:text\/\S\*|application\/xml|\S\*\/\S\*\+xml)\s\*;.\*charset\s\*=\s\*utf-8/i.test(blob.type)) {  **return new** Blob([String.fromCharCode(0xFEFF), blob], {type: blob.type});  }  **return** blob;  }  , FileSaver = **function**(blob, name, no\_auto\_bom) {  **if** (!no\_auto\_bom) {  blob = auto\_bom(blob);  }  // First try a.download, then web filesystem, then object URLs  **var** filesaver = **this** , type = blob.type  , force = type === force\_saveable\_type  , object\_url  , dispatch\_all = **function**() {  dispatch(filesaver, "writestart progress write writeend".split(" "));  }  // on any filesys errors revert to saving with object URLs  , fs\_error = **function**() {  **if** ((is\_chrome\_ios || (force && is\_safari)) && view.FileReader) {  // Safari doesn't allow downloading of blob urls  **var** reader = **new** FileReader();  reader.onloadend = **function**() {  **var** url = is\_chrome\_ios ? reader.result : reader.result.replace(/^data:[^;]\*;/, 'data:attachment/file;');  **var** popup = view.open(url, '\_blank');  **if**(!popup) view.location.href = url;  url=**undefined**; // release reference before dispatching  filesaver.readyState = filesaver.DONE;  dispatch\_all();  };  reader.readAsDataURL(blob);  filesaver.readyState = filesaver.INIT;  **return**;  }  // don't create more object URLs than needed  **if** (!object\_url) {  object\_url = get\_URL().createObjectURL(blob);  }  **if** (force) {  view.location.href = object\_url;  } **else** {  **var** opened = view.open(object\_url, "\_blank");  **if** (!opened) {  // Apple does not allow window.open, see https://developer.apple.com/library/safari/documentation/Tools/Conceptual/SafariExtensionGuide/WorkingwithWindowsandTabs/WorkingwithWindowsandTabs.html  view.location.href = object\_url;  }  }  filesaver.readyState = filesaver.DONE;  dispatch\_all();  revoke(object\_url);  }  ;  filesaver.readyState = filesaver.INIT;    **if** (can\_use\_save\_link) {  object\_url = get\_URL().createObjectURL(blob);  setImmediate(**function**() {  save\_link.href = object\_url;  save\_link.download = name;  click(save\_link);  dispatch\_all();  revoke(object\_url);  filesaver.readyState = filesaver.DONE;  }, 0);  **return**;  }    fs\_error();  }  , FS\_proto = FileSaver.prototype  , saveAs = **function**(blob, name, no\_auto\_bom) {  **return new** FileSaver(blob, name || blob.name || "download", no\_auto\_bom);  }  ;    // IE 10+ (native saveAs)  **if** (**typeof** navigator !== "undefined" && navigator.msSaveOrOpenBlob) {  **return function**(blob, name, no\_auto\_bom) {  name = name || blob.name || "download";    **if** (!no\_auto\_bom) {  blob = auto\_bom(blob);  }  **return** navigator.msSaveOrOpenBlob(blob, name);  };  }    // *todo: detect chrome extensions & packaged apps* //save\_link.target = "\_blank";    FS\_proto.abort = **function**(){};  FS\_proto.readyState = FS\_proto.INIT = 0;  FS\_proto.WRITING = 1;  FS\_proto.DONE = 2;    FS\_proto.error =  FS\_proto.onwritestart =  FS\_proto.onprogress =  FS\_proto.onwrite =  FS\_proto.onabort =  FS\_proto.onerror =  FS\_proto.onwriteend =  **null**;    **return** saveAs; }(  **typeof** self !== "undefined" && self  || **typeof** window !== "undefined" && window  || **this** )); |

4、Req：

a) 导出的数据没有按照插入MAP的表头顺序显示

---> 原 因：HashMap是散列映射, 插入数据不记录插入顺序

---> 解决方案：使用LinkHashMap || TreeHashMap

b) 接口获取无法同时获取参数和Request对象

---> 解决方案：使用公共Response， 接口只获取参数

|  |
| --- |
| protected HttpServletResponse response;  */\*\*  \* 获取Response  \* \*/* @ModelAttribute public void setReqAndRes(HttpServletResponse response){  this.response = response; } |