# 方案简介

首先这里罗列一下当前服务器所存在的问题：

* 工厂只有一个账号，而且必须同时我司操作员进行生产，有人员流动时不安全。
* 当前生产记录生产系统不完善，存在一些bug
  + 记录上传时有一个计数参数出差，会导致工厂看到的生产数量与实际数量不一致。
  + Mac查重的机制不完善，仅对不同批次间有查重
  + 没有返工记录上传机制
* 产测工具端也存在一些问题
  + 工具不是很稳定
  + 配置项太多，全是文件，容易出错

针对以上这些问题，我们组计划对原有的产测服务器进行修改完善，并制定了详细的设方案，下面按照产测服务器相关的方案和产测工具相关的方案进行计划描述。

## 产测服务器

首先我们将服务器原来简单的一体化设计按照功能（包括要添加的新模块）分成4个部分：

* **账号管理系统：**解决一个工厂只能有一个账号的问题，同时也区分不同角色的不同权限，为其他系统的设计搭建基础。
* **订单跟踪系统：**根据参与订单的不同角色的操作需求，都对应设计有不同的操作界面，使得原本仅通过邮件进行交流的方式更加的规范化，完整的记录也能明确责任。
* **产测管理系统：**即原有的产测系统
* **品质管控系统：**根据原有的产测数据和新计划上传的数据，进行更有针对性的数据统计分析，从而提高工厂生产的品质。

## 产测工具

对于产测工具的新方案的重点是：

1. 提高多产测工具稳定性
2. 提高多通道产测的速度
3. 配合新产测服务器，使得产测过程我们更加可控

根据原来的问题及以上需求，我们设计了新的产测工具，目前以基本完成。同时也设计了一个的产测方案——flash跳转测试方案。方案从原来的下载到内存进行测试的方式，改为预先下载到flash内进行测试，可以节省下载ram的时间，并且采用命令启动测试，使得测试过程更加可控；另外新的产测工具配合新的产测服务器设计了云端直接下载测试config的接口，可以防止操作员配置错误的潜在问题，同时我们也能；最后根据新的品质管控的需求，我们设计了新的测试项目用于上传，通过统计分析新的上传数据，模组或方案设计人员可以分析出新模组设计的缺陷等。

本文主要目的是介绍服务器的设计和更新，产测工具只进行必要的接口相关介绍，下面将根据功能设计详细描述各个部分的设计。

# 方案设计

本章将根据设计具体介绍上面所提到服务器的：账号管理、订单追踪、测试系统（完善）、品质提升等四个部分。将按照对应的4个小节进行分别介绍。

## 方案结构设计

### 软件环境

为了更好的兼容原产测系统，计划拓展的内容也将按照原有的体系进行开发：

采用B/S架构，后台使用golang+mysql，前端使用html+angularJs进行开发，具体的项目架构请参考：

<https://gitlab.espressif.cn:6688/yunzhou/factory/wikis/home>

### 硬件环境

开发环境是Linux16.04，

服务器运行环境使用Linux + nginx + mysql

浏览器建议使用Chrome或者firefox最新版浏览器

## 生产订单操作流程

再介绍账号系统和订单追踪系统之前，需要先介绍整个订单操作的流程，包括其中所参与的角色的介绍，图 2‑1所示的就是订单操作的流程图。

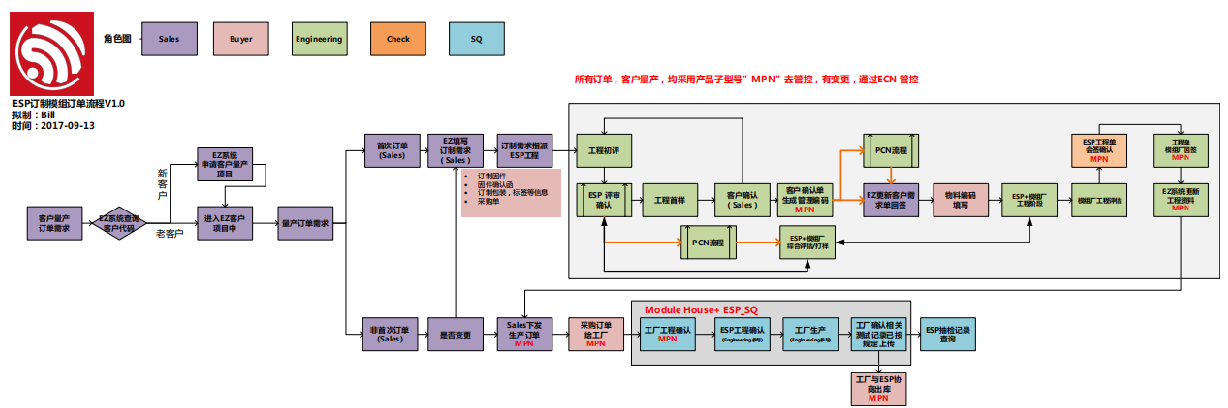


图 2‑1 订单流程图

Sales：商务

Buyer：采购

Engineer、Check：工程技术确认员

SQ：品质

这里介绍一下，一个正常生产订单完整的操作流程。首先，系统将订单分为两类

* MPN（生产管理编码）订单：用于唯一标识一个特殊需求的订单，需求包括生产测试时所涉及的全部方面，包括是否定制固件、是否定制标签等。MPN订单有sales发起，需要依次经过buyer、engineer和check人员的确认，再发送到工厂方面进行实际测试确认，最终反馈到sales，标明此MPN订单可用于生产。
* 生产订单：具体的生产需求订单，在发起新的订单前，需要有本次生产订单所需求对应的MPN订单，若没有对应需求的MPN订单就必须先发起一个MPN订单确认。sales选择一个已存在的MPN订单来发起一个生产订单，然后采购进行确认即可，不需要再通过工程人员和工厂方面的确认。

## 账号管理系统

按照订单系统中参与的角色，系统目前总共需要的角色大致有以下几种：

* 我司：
  + 管理员：拥有最大权限，用于管理整个系统，能登录全部的页面进行查看，并能管理其他账号。
  + 商务（sales）：订单的发起方，订单分为生产订单和MPN订单，拥有订单的发起和修改权限及对应的界面。
  + 采购（buyer）：订单确认方，确认商务发起的订单，和查看各个生产订单目前的状态，和确认订单进行生产的权限。
  + 工程（engineer、check）：需要根据MPN需求填写并生产具体的测试config文件，并进行对应的测试确认，在发送到工厂进行确认，最后在系统回报结果。
  + 品质（qulity）：主要是根据品质系统的信息，统计各个工厂的生产状态进行相关的操作
* 工厂：
  + 工厂管理员：确认MPN需求并对工程发出的config文件进行实测确认，并可以管理工厂名下的操作员账号。
  + 操作员：实际生产中操作产测工具的人员。

表 2‑1 账号数据库表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 描述 |
| id | int(11) | 用户唯一标识、主键 |
| account | varchar(64) | 账号，唯一标示 |
| password | varchar(64) | 密码（md5 加密） |
| name | varchar(64) | 姓名 |
| created | Datetime | 创建时间 |
| lastlogin | Datetime | 最后登录时间 |
| visibly | tinyint(1) | 删除标识 |
| groupid | int(11) | 用户组 |
| email | varchar(128) | 用户邮箱 |
| remark | varchar(256) | 用户备注 |
| factory\_sid | varchar(64) | 绑定工厂的sid作为token以兼容原来的系统（外键） |

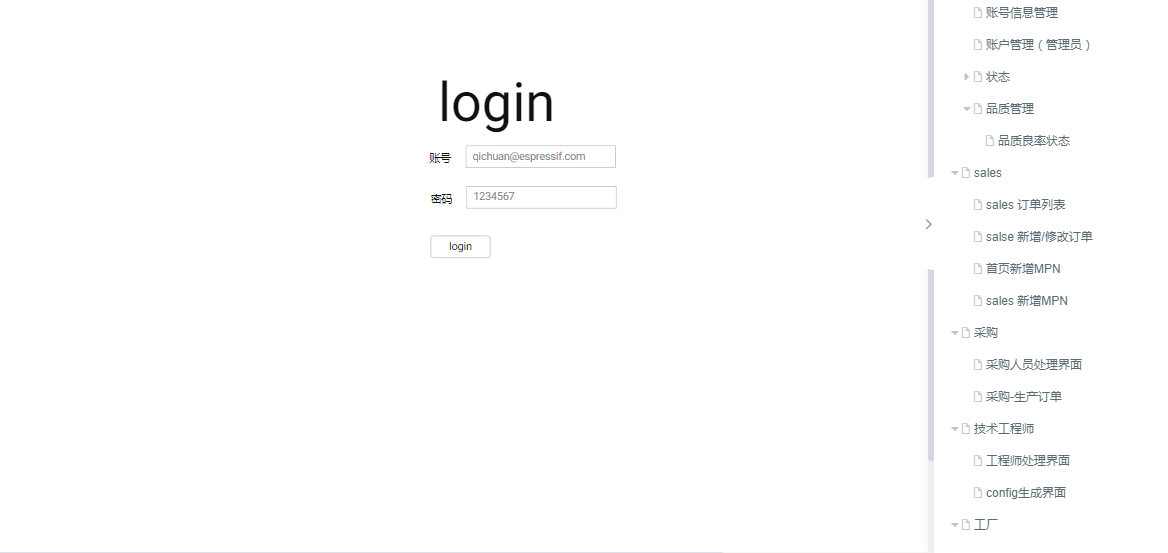


图 2‑2 网站页面结构

如图 2‑2所示，每个用户组能看见和操作的页面都不尽相同，像个人账号管理的页面都是有的，其他的页面都将根据账户的权限（分组）才能访问。

## 订单追踪系统

订单系统是本方案拓展的重点之一，有很多的角色参与其中，用户行为和操作逻辑输入比较简单，但是要保证生产不会出错，并且实现订单可追溯，还是有一定的挑战的。

### 用户行为

不同组的用户有不同的责任与权限，方案将按照用户需要进行的操作行为来设计操作页面。所以首先需要整理不同用户的不同行为：

表 2‑2 用户行为及页面列表

|  |  |  |
| --- | --- | --- |
| 用户组 | 行为 | 页面 |
| sales | 管理所有的订单 | 订单列表界面（商务） |
| 发起或修改MPN订单 | 新增MPN界面 |
| 发起或修改生产订单 | 新增生产订单界面 |
| buyer | 管理所有的订单 | 订单列表界面（采购） |
| 查看订单详情并反馈 | 订单详情界面（采购） |
|  |  |  |
| engineer | 管理所有订单 | 订单列表界面（工程人员） |
| 查看订单详情并反馈 | 订单详情界面（工程人员） |
|  | 生成产测配置文件 | Config生产界面 |
| factory | 管理所有订单 | 订单列表界面（工厂） |
| 查看订单详情并反馈 | 订单详情界面（工厂） |
|  |  |  |
| quality | 管理工厂生产品质 | 品质管理界面 |
| administrator | 管理员 | 管理员 |

### 订单数据库表

订单分为MPN和生产订单，所以数据库表也有对应的两张：

表 2‑3 MPN数据库表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 备注 |
| id | int(11) | 自增主键 |
| mpn\_sid | varchar(45) | Mpn号，唯一 |
| visibly | int(4) | 删除标识 |
| created | date | 创建时间 |
| updated | date | 最后修改时间 |
| user\_sid | varchar(20) | Mpn创建者（外键） |
| status | enum | Mpn确认状态 |
| *factory\_sid* | *varchar(45)* | *工厂sid* |
| fw\_ver | varchar(100) | 定制固件的版本号 |
| config\_sid | varchar(100) | 配置文件的名字 |
| packing | int(2) | 是否定制包装 |
| label | int(2) | 是否定制标签 |
| remark | varchar(200) | 备注信息 |

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 备注 |
| id | int(11) | 自增主键 |
| order\_sid | varchar(45) | 订单编号（公司内部编号） |
| visibly | int(4) | 删除标识 |
| created | date | 创建时间 |
| updated | date | 最后修改时间 |
| user\_sid | varchar(20) | 订单创建者（外键） |
| count | int(11) | 订单生产数量 |
| status | enum | 订单生产状态 |
| remark | varchar(200) | 备注信息 |

### 页面设计

参考墨刀设计的界面。。。

## 产测系统完善

针对目前产测系统中的问题，计划采取的应对方案：

* 记录上传时有一个计数参数出错，会导致工厂看到的生产数量与实际数量不一致——对计数采用加锁保护，并验证效果和延迟。
* Mac查重的机制不完善，仅对不同批次间有查重——修改原有的查重逻辑判断，并验证所有情况的判断结果是否符合预期，也要验证效率。
* 没有返工记录上传机制——新增返工批次的处理流程

### 加锁计数

首先说明产生错误的原因预测：在testdata.go文件中，上传合法性逻辑判断之后要更新batch当前状态的时候，要进行2步操作：1.读取当前batch状态，2.根据判断结果更新batch状态。由于原有的程序未对这两部进行互斥加锁的处理，导致在多线程（多个请求同时上传）时有出错的可能。

根据上述原因，在testdata.go中调用dal.FindBatchBySid（获取batch当前）前，到调用batch.NextRemainMac函数之后进行加锁处理（但是实际操作中仅对此进行加锁解锁处理会导致程序死锁。。原因不明；从而采用在调用dal.FindBatchBySid前加锁，同时在函数结束后解锁——调用defer；导致的结果就是连带最后的保存testdata的操作也在加锁处理内，逻辑上不影响程序的正确性，但理论上会进一步加长加锁处理的时间影响多线程的性能）。

通过编写多线程同时上传的脚本进行对比测试：4线程同时操作，每个线程连续上传50条不重复的mac测试数据（mac连续），总共200不重复条记录。

结果显示如下：

加锁：共耗时10.77秒，200个数字没有重复（2457~2656）

未加锁：共耗时5.66秒，137个数字（2657~2793）有63条记录得到了相同的数字

根据测试结果基本可证原因如预测，虽然效率上减半，但鉴于每条记录从0.025秒到0.05秒的变化不影响正常生产，且该数据对于系统使用比较重要，故建议做此修改。

### Mac查重

#### 问题背景：

这里先介绍一下问题的背景，以及各种情况的存在。

首先一块芯片的mac理论上都是全球唯一的，这个唯一性由mac分配组织保证，我们向其购买mac地址段，再分别烧录到每块芯片的某个地方，有程序读取使用。唯一性对网络芯片及对应的网络产品而言非常重要，不仅涉及到多个协议中会使用mac作为各个实体的区分，对于目前越来越普及的物联网设备而言更为重要，mac是区分产品个体的唯一标准（很多云端方案都是这么做的）。

我司将mac地址写到芯片的efuse（一种存储结构，使用熔断烧写，故不可逆）中，由于芯片生产时的原因（目前我不知道是什么问题）出现了mac重复的情况，由于作为芯片阶段没有处理，所以我司在制作成模组之后进行判断处理。

#### 处理措施：

发现了mac重复之后，我司的处理措施是，在efuse的其他位置再烧录一个新的mac，并配合程度使用新的mac以达到唯一性的保证。所以就出现了一块芯片（或模组）存在2个mac的可能（默认mac一点存在，新的mac有可能存在或者为全0），方便区分称之为chip\_mac（默认位置的mac），和cus\_mac（新定制的mac）。处理时（重烧mac）会上传一条新的记录，带上2个mac，并作区分标识。

另外，模组还可能由于其他的原因进行返工（例如重烧固件等），处理时也会上传记录，但是会按照一个新的batch\_sid 进行。

#### 原有系统的问题：

原来系统中并没有针对mac可能出现重复的所有情况做不同的处理措施，只有一处判断当前上传记录的mac是否已经存在系统中，如果已存在则判断当前处理的批次与之前的批次是否相同，如果不同则报错，如果相同则判断为同一块模组更新数据库。这个逻辑最大的问题是，如果一个批次内的模组出现重复则系统无法甄别出来，从而导致mac重复的芯片流向市场。

#### 问题分析：

由于mac重复、复测、mac重复后返工、其他情况的返工（例如更新固件等），已经对mac重复的处理措施——定制mac的存在，针对这个问题，首先要弄清楚所有可能出现的情况，再根据不同的情况作出不同的返回结果。

当一条记录的mac第一次出现肯定是没有问题的，所以要区分当已存在mac出现时的情况。由于有测试失败的情况，但是系统不为其分配一个batch中的位置（及2.5.1小结中的计数）直到此mac首次测试成功并上传才获得在batch中的位置，所以这里的判断逻辑主要基于测试成功并上传的那些记录。

|  |  |  |
| --- | --- | --- |
| 批次类型 | 是否复测 | 是否带定制mac |
| 正常生产 | Y/N | Y/N |
| mac返工批次 | Y/N | Y |
| 其他返工批次 | Y/N | Y/N |

有上表可知，记录的情况最多有10种可能，如果知道每条记录的情况那么判断就很简单，根据来源进行不同的逻辑判断即可。但实际情况是，我们可以知道每批次的批次类型（在上传时以特色参数作区分），但是目前使用的产测方案无法区分某块芯片（或模组）某次操作是首次还是复测（虽然新的产测方案针对这个问题做了处理，来识别是否已经完成测试，但是必须要兼容旧的产测方案）所以必须在服务器上进行判断。

首先系统里应该没有任何已存在的mac记录，所以只要查到有记录，那么就只有复测或者mac重复2种可能。

如果查出记录且batch不同，那么一定是重复的mac；

如果batch相同，理论上服务器无法区分是复测还是重复，正常情况下一块模组测试通过就不会再次被测试上传记录，可以一刀切的将重复的记录判断为mac重复。但是考虑到产线工人的能力以及大强度的劳动下有可能出错，将已测试通过的模组重新测试，故也存在误判的可能。

目前我们只能想出一个过渡的机制——考虑到误重测的情况一般是：测了一块模组，但是不确定刚刚是否已经测过，所以保险起见再测一次而导致复测。所以我们加入时间缓冲判断机制，一块新模组在第一次成功记录上传之后的2分钟内（120秒），再上传记录会显示已存在记录，但是工具并不报错，但是2分钟后再上传同mac的记录都会显示mac重复。此方式确实也存在风险，即重复mac的两片模组在同一批次内，且在2分钟内同时被测试通，系统就会判断为同一块模组。

所以，要彻底解决这个矛盾的问题就必须准确的判断出是否重测，这个在服务器端是无法实现的，必须配合产测工具，由产测工具判断是否重测进而通知服务器或者就不上传直接显示通过，我们新的产测工具方案也是这么做的（测试通过后在flash指定位置写下标志位标明已经测试通过，再次测试时上电会判断标志位，若已完成测试则直接显示通过）。但是新方案的导入需要时间。

但是考虑到本来mac重复的可能性也不高，所以整体风险非常小，并且基本不存在批量重复的可能性，对于市场的风险属于可控。

基于此，我们设计了如下的mac查重判断方案：

**正常的生产/mac返工批次**：

首先我们知道只要芯片存在定制的mac，那么不管是什么原因，程序都会以定制的mac作为芯片的mac，这很重要，有助于简化判断，同时这样也就不需要区分正常批次和mac返工批次。

1. 检查mac有效性并选择主要判断mac——cus\_mac优先（全0表示没有）
2. 根据选择的mac查询是否已存在记录（只选择一个mac查，查询最新的一条），若不存在则此记录为新纪录根据测试结果进行操作写入数据库；若存在则进入下一步判断。
3. 做mac检查判断如果已经存在cus\_mac的记录，且chip\_mac不相同则判断为cus\_mac重复（当然就是mac重复，不区分具体哪个mac重复）。
4. 搜索是否存在已经测试通过的记录，若存在则判断，当前时间与查到记录保存时间的间隔，若大于120秒，则返回mac重复，若小于120或不存在记录则进行下一步判断。
5. 判断当前记录与原记录的batch是否相同，若不同则返回批次错误，否则即判断为重测记录，进而更新旧记录、写入新纪录并标为最新，更新batch信息，为记录排号。

P.s．对于新的产测方案而言，区别的地方在于第4步时，如果读到已经存在成功的记录就直接返回mac重复错误，不进行120秒的判断操作，或者将120秒改成0即可。

P.s.2.由于调试和实际生产前测试以及兼容更早版本的产测工具的需求，mac重复判断作为一个可选项，根据产测工具的上传时的参数决定是否进行查重。

### 返工处理

2.5.2节已经说明，mac返工的订单不需要再做特殊的处理，需要做的是其他类型返工的处理。由于已经确定了是返工且不是mac返工，所以可以说这批的mac是已经做过校验的，不需要再进行mac查重（否则一定是重复的），系统在上传时也特别指定了一个特殊字段来标识返工类型，所以根据标识我们就可以跳过mac查重的判断。

待补。。。

## 品质管理系统

### 原系统的问题

品质管理系统的主要目标是，对产测时上传的测试结果数据进行分析和统计，形成批次良率的统计、展示等，从而使得我司的品质管理人员可以通过这些数据对工厂的生产状态进行管理，及时发现不合理的地方加以改善，以提高生产的良率、效率的等。

另外，对于某些客户的特殊需求，如导出某批次的所有产测记录，或者某批次的所有mac的列表，或者我司自己进行查看也属于，品质管理的范围。

其次，原来的系统只是收集每次产测的结果（即成功或失败），所以只能进行类似良率的统计。如果能上传一些测试的关键参数，如rf性能相关的参数，再进行对应的统计分析，那么我们的芯片设计或者模组设计人员就也能通过数据来分析，次批次是否存在问题，或者模组设计上就存在问题，能更有针对性的提高我司产品的质量和生产。

但是，目前的产测方案的数据相对来说不够“浓缩”，有很多数据，同时数据格式也不够规范，无法通过一个两个数据就能表明某项性能的好坏。这导致数据的存储和统计不方便，即使统计出来也很难看出问题所在。所以品质系统优化的首要任务是，设计一个新的或者更加规范的方案，能够产生更加规范的数据，或者将一些数据进行合并产生更方便理解的结果。

### 新的产测方案

对此，我们设计新的产测方案——flash跳转测试方案。新方案的目标有2个，第一是使用flash内的产测程序就不用下载到ram再进行测试，能节省一些时间；另一方面新方案使用新的产测固件程序，针对2.6.1提到的问题对了相应的优化，把之前很多会产生多组结果的数据进行合并成一个结果数值，也表明了最重要的对于产测分析最有效的数据进行标明，再配合新的产测工具，可以实现这些附加数据的上传。

对此服务器也需要进行对应的业务扩展升级。

1. 首先是要进行数据库的升级，需要新增一张用于存储这些测试数据的表。
2. 原有的统计界面也需要根据这些数据进行对应的展示。