

**Cmu支持订阅批量通知方案**

（仅供内部使用）

|  |  |  |
| --- | --- | --- |
| 版 本 号： | | V1.0 |
|  | | |
| 编 制： | 熊学虎 | |
| 审 核： |  | |

**修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本号 | 描述 | 作者 |
| 2015/09/07 | V1.0 | 创建 | 熊学虎 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 名词解释 2](#_Toc430096290)

[2 设计背景 2](#_Toc430096291)

[3 设计方案 2](#_Toc430096292)

[3.1 具体实现细节 4](#_Toc430096293)

[4 波及分析 7](#_Toc430096294)

[5 自测用例 7](#_Toc430096295)

[6 遗留问题 7](#_Toc430096296)

**关键词：**

订阅、通知、批量、cmu

**摘 要：**

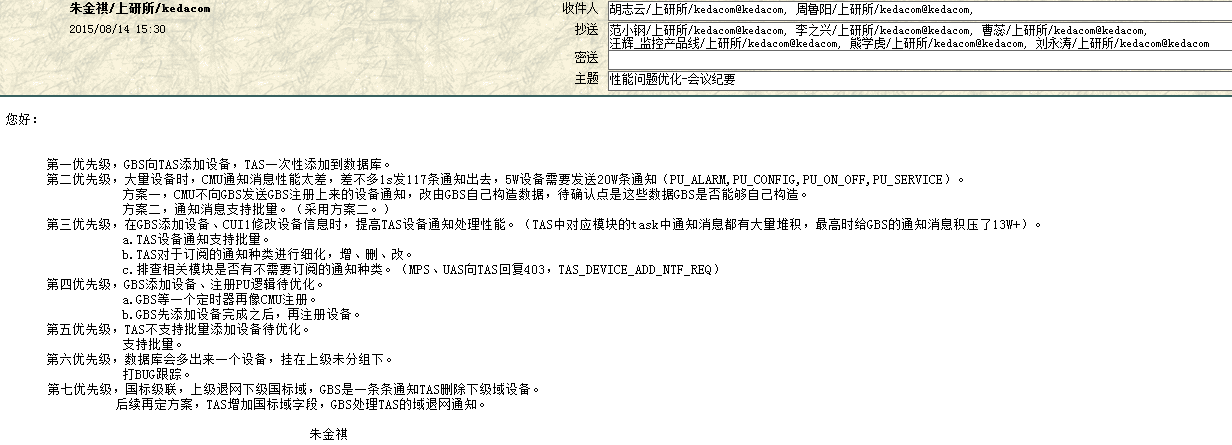
优化cmu的订阅通知，使得可以一个通知消息中能发送多个设备的通知信息。

# 名词解释

# 设计背景

目前cmu的订阅通知中，一次通知只包含了一个设备的信息，当设备量大的时候，需要发送的通知太多导致订阅方会等待很长的时间才能获取完所有设备的信息，比如接入10W设备的gbs，gbs向cmu订阅设备的各种信息（如PU\_ALARM,PU\_CONFIG,PU\_ON\_OFF,PU\_SERVICE等），初步计算，每个设备将产生约4个通知，这样订阅成功后，gbs将要收到40W的通知，这个过程将花费大量的时间，为了进一步提高通知效率，可以在一个通知中携带多个设备的信息，增大单个通知的信息量，减少通知的数量。

该需求是通过会议讨论的方式提出的，具体的会议纪要如下：



根据上述会议结论：其中第二条主要是涉及到cmu端订阅优化的，其他的几条主要是3as和gbs端改动，可能cmu端需要配合改动，故本文档主要记录第二条的优化方案。

# 设计方案

为了使本次的改动不影响到其他的模块，故将批量通知设置为订阅可选特性，默认情况下还是发送单个设备的通知，只有当订阅者设置了接收批量通知的标志后，cmu才向订阅者发送批量通知，这样诸如rcs等按需订阅的模块，就无需任何改动，只有出现了订阅通知导致性能瓶颈的模块，才需要设置这个选项。

以下根据现有的订阅机制来分析如何实现批量通知，现有的订阅通知的特点如下：

1. 针对每一个订阅者均设置有通知队列
2. 通知队列采取每次发送一次通知，收到成功应答后发送下一个的串行机制
3. 单次通知中只有单个设备的单种类型的通知内容

根据上述的特点3，可以从以下两个方面进行优化，如下：

1. 单次通知中携带多个设备的同一类型的通知内容
2. 单次通知中携带单个设备的多种类型的通知内容
3. 单次通知中携带多个设备的多种类型的通知内容

对于第1点，订阅的初始通知比较容易实现批量，但是后续通知可能难以实现一次通知中携带多个设备的信息，因为每个设备的信息变更都是独立的，很难出现多个设备同时变更同一种数据的情况。如果采用扫描通知队列，从队列中提取不同设备的同一类型的通知，一来破坏了通知队列的队列属性（不再是队首出，队尾进），二来为了实现批量发送通知需要遍历队列，以便找到能合并到一起发送的，三来通过遍历通知队列，提取同一类型，不同设备的通知内容，可能导致设备的多种类型的通知顺序会被打乱，比如先发puconfig，然后再发onoff（之前做订阅重构的时候，发现部分模块要求先接收onoff）

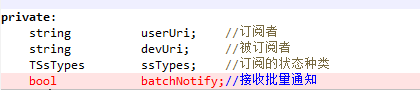
对于第2点，初始通知也比较容易实现批量，同样的后续通知可能难以实现在一次通知中携带多个类型的通知，实际的通知队列中可能是多个不同的设备的不同的类型穿插存在着，如果要在一次通知中包含同一个设备的不同类型的通知内容，也会面临第1点的问题，破坏了队列的属性。

对于第3点，要在一个通知中支持多个不同的设备的多种不同类型，首先针对多个不同的设备，可以使用vector来完成，要在一种元素中保存多种不同类型的通知，可以使用string来保存不同类型的通知序列化后的结果，然后设置标记说明具体的通知类型，接收端收到通知后根据标记来完成具体的反序列化即可。为了最大限度的能使得批量发送通知，可以在读取队列的时候完成批量组装通知包的功能，而不是在投递通知到通知队列的时候完成这个过程（可能投递的量很小不足以组成批量包，但是队列中有大量的通知存在，这种情况会降低批量组包的效率）。采用这个方案的话，无论是初始通知，还是后续的变更通知，均可以在通知压力大的时候实现批量通知。

## 具体实现细节

1. 订阅者在订阅的时候，新增字段说明是否接收批量通知，这样其他接收通知量较少的模块就可以不修改，不需要处理批量通知，增强平台的兼容性。

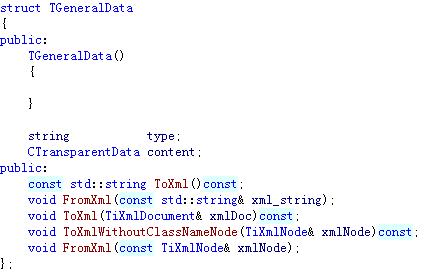
CSsReq新增字段batchNtf，表名订阅者是否需要接收批量通知，具体如下：



1. Cmu在发送通知队列中的通知时，不再是一次发送一条通知，而是根据包的大小（控制在25K以内），一次将多个通知组成一个批量通知，然后发送出去。根据这个方案可以知道，当通知队列中总是零散的存在待发送的通知时，批量通知相比之前的单个通知的提升有限，但是当队列中有大量的通知存在时，批量通知可以大幅提高单次通知中信息的有效率，而实际需要优化的，也正是通知队列中存在大量通知时这种情况，即解决通知队列的突发压力。

这个方案的重点在于如何将多个设备的多种类型的通知放到单次通知中。首先，为了能支持多个设备，必然要使用vector或者set等容器来保存单个设备的各种类型的通知；为了能以一种数据形式来保存单个设备的不同类型的通知，可以将不同类型的通知先序列化为xml，然后以string的方式保存，为了方便接收端不用解析string就能获取通知的类型，最好还能将通知的类型一并发送出去。通知队列为了能保存各种类型的通知消息，通知消息都是以xml形式的string来保存的，所以将通知队列中的多个通知合并为批量通知，可以通过累加多个通知的长度来作为批量通知的长度，而不用每次都将批量通知序列化来获取批量通知的长度，加快拆包的速度。

2.1设计的用于保存单个设备的不同类型的通知的通用数据类型如下：



type字段说明数据的类型，这样接收端可以根据这个类型来产生对应的对象，而不用先从具体的消息中解析event字段后再来产生对应的对象

content字段是具体的通知类型，是序列化以后的结果。这里使用的是CTransparentData而不是string，是因为content本身就是xml内容，使用CTransparentData的话，在解析批量通知的时候，可以不用解析content中的内容，后续在解析具体的通知内容的时候，再解析实际内容。如下代码以供解析批量通知的参考：



2.2 cmu在发送通知时，如果对方在订阅时指定支持批量通知，则会优先发送批量通知。即当通知队列中的通知数量大于1，并且订阅者支持接收批量通知的话，则cmu会将读取通知队列中的多条通知，以组成一条批量通知。如果订阅者支持批量通知，但是通知队列中的通知数量小于等于1的话，则以普通的单个通知的形式发送，以加快订阅者处理通知的速度（如果批量通知中只包含一条记录的话，理论上处理普通通知会比批量通知要快，因为批量通知在处理时，需要分两部分反序列化）

3 跨域订阅时，cmu和cmu之间使用批量通知，因此cmu端既要实现批量通知，同时还要实现批量通知的接收和处理，其他模块只需要处理批量通知即可，实现上可以参考cmu是如何处理批量通知的。

# 波及分析

受影响的主要是：

1. cmu和cmu之间的跨域订阅，订阅请求中的batchNotify设置为true，故而能接收批量通知，加速跨域订阅通知的效率
2. 部分模块如cui和rcs和cmu之间的订阅，由于这些模块这次没有做改动，故而要求它们收到的通知还是普通通知，不能出现批量通知，否则无法处理
3. Gbs和cmu之间的订阅，gbs支持批量通知，要求cmu在发送通知时，尽可能的发送批量通知。

# 自测用例

1. 单级平台，登录cu，查看设备状态，要求设备状态准确无误
2. 在1的基础上，下线部分设备，要求cu上设备状态准确无误
3. 组成科达级联的上下级，上下级均有设备，查看上下级设备的状态，要求所有设备的状态准确无误，抓包检查下级给上级发送的通知，可能出现批量通知
4. 环境同3，杀掉下级的cmu，要求在cu上下级的所有设备均下线，抓包检查下级给上级发送的通知，可能出现批量通知
5. 以上测试用例中，注意多使用cmu的调试命令sslist和dsslist，观察通知队列的状态等参数

# 遗留问题

虽然支持批量通知后，能在一定程度上提高通知的效率，但是仍然可能面临订阅者通知队列持续增长的问题，当通知队列中缓存的消息太多的时候，cmu使用的内存将持续增长，可能最后被系统oom，对于这个问题，暂时不做解决，以下方案可供参考：

1. 通知队列中缓存的消息达到上限时，可以考虑丢弃部分通知。显然，丢弃部分通知会导致订阅者无法获取到正确的状态，此方法意义不大。
2. 当通知队列中的消息达到上限时，由cmu端发起和订阅者之间的断链，这样订阅也随之拆掉，由订阅者重新订阅。这样可以一定程度解决问题，但是同样治标不治本，伴随而来的问题是可能导致频繁的断链。