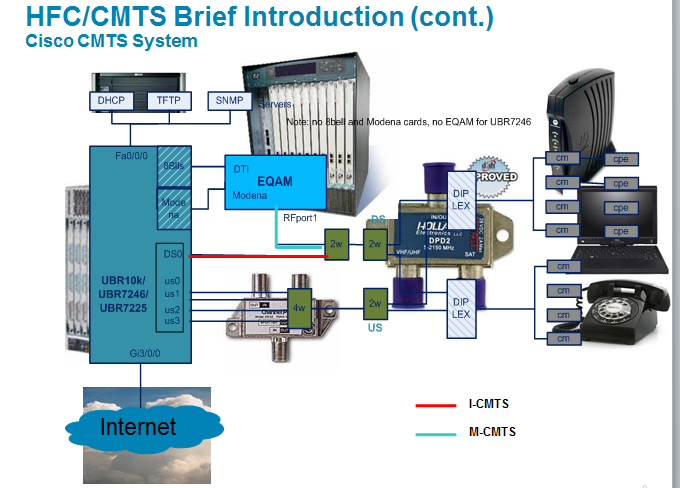
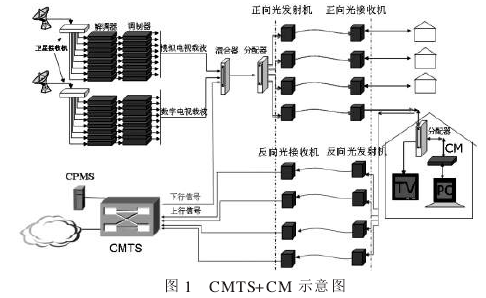
1. 思科项目介绍
2. CMTS: (**C**able **M**odem **T**ermination **S**ystem) 实质是一种接入路由器,一端接用户,一端接接入网或边缘路由器.放置在运营商机房,通过光纤出去，然后接入用户的叫HFC光纤同轴混合网。具有DHCP TFTP等功能。





其中：Downstream: CMTS to CMs. 广播的形式。所有CM可以收到，通过接收端的MAC或DOCSIS3.0后DS ID辨别目的地。

Cable Modem和普通的Modem一样，是数据信号在模拟信道上交互传输的过程，但是，普通的Modem的传输介质在用户与访问服务器之间是独立的，用户独享传输介质，而Cable Modem的传输介质是HFC网络，将数据信号调制到某个传输带宽与有线电视或者电话信号共享介质，并且Cable Modem结构也较普通的Modem复杂，无须拨号上网不占用电话线，可以随时在线连接。

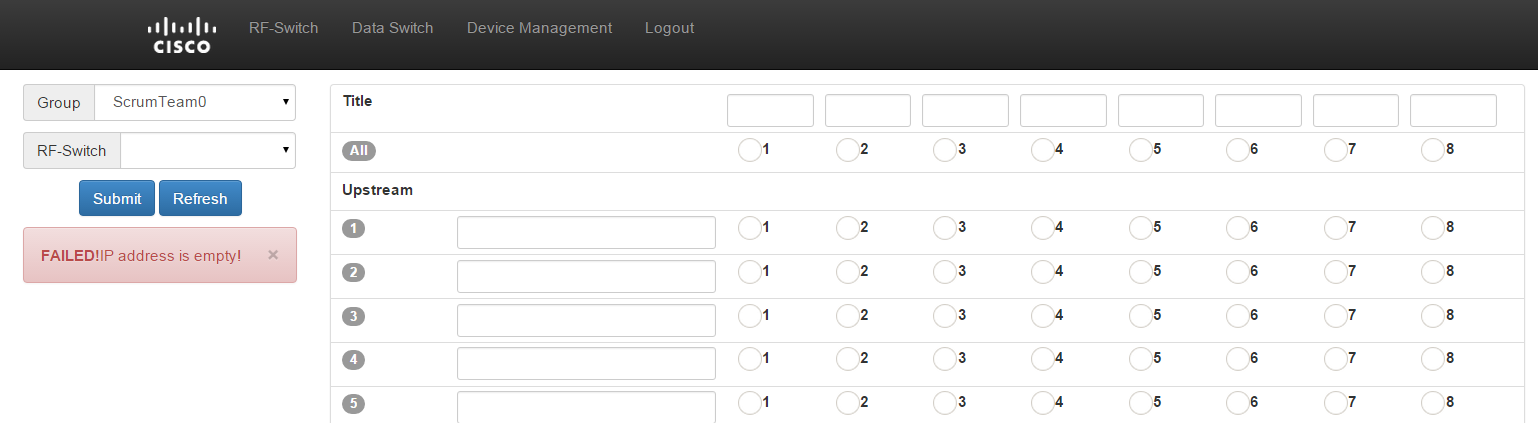
Cable Modem实现是从87MHz—860Mhz电视频道中分离出一条6Mhz用于下行数据传输采用64QAM,下行数据一般是从：5MHz—65Mhz之间的频谱上进行传送，为了有效抑制上行噪音积累，一般用qpsk，因为它比64QAM更适合噪音环境，但是速率较低。

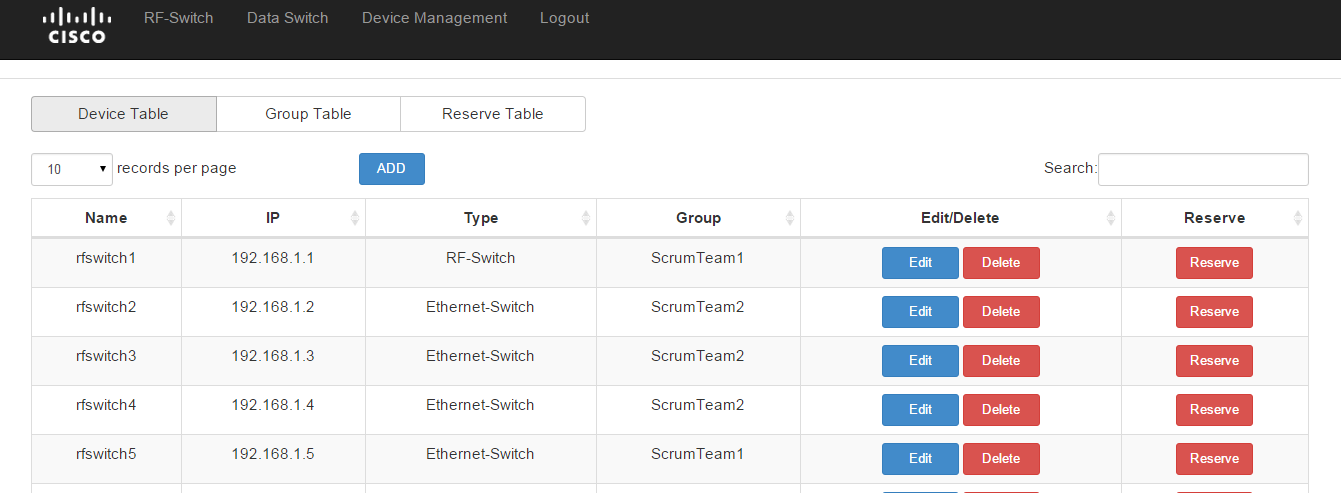
1. RF-Switch:

Rf-switch 的左边的in 去接linecard，右边的out是给CM。来是实现n + 1冗余功能。7+1冗余，应该是8个linecard，其中在cmts上已经有了8 个linecard（7个常用的一个备份的），然后当有时候运行商要更改某个linecard或者是当其中有一个linecard坏掉的时候，就讲这个转换到备份的（protected）上，同样rfsw上也要改掉。Problem1 ，怎么转换的呢？？如何去切换？？

Rf-switch:本来作用是实现N+1冗余，也就是说实现当有一个linecard坏掉的时候，就把他其中坏掉的一个切换到protected，（实际上Rf-switch仅仅起一个开关作用，in 和 out 在内部通过开关是连接在一起的）之前是 rfswitch 连接各个linecard，然后output出来后，一路ds 和一路us 混频出去就相当于一个fiber note(FN),然后FN，再经过HFC到modem，在实际应用中，整套设备是放在运营商的机房中，然后机房到用户家是通过HFC网络。但是在实验室中存在的问题是： modem资源比较紧张，很多工程师都在去用modem资源利用，如果能够实现modem可以由工程师在web上自动去切换或者预约modem，这样就比较好。所以我们充分利用Rf-switch可以实现自动切换的作用 的功能，就可以实现自动切换modem供给工程师去用。所以现在是这样modem 架去接protect,(这个地方有个问题，一个protect最多可以去接多少猫架呢？？？)然后输出是通过ds 、us 混频出去，去接fiber note。。







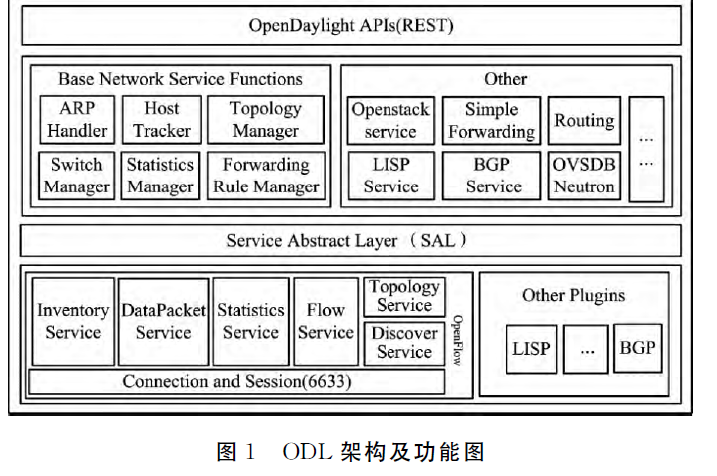
1. 关于工程的框架实现：
2. Sdn

软件定义网络，把数据层和控制层分开，分为3层，最底层：数据转发平面，提供物理连接网络并根据转发规则转发数据，中间层是控制平面，是一个软件层，负责提供网络基础服务，提升网络的可编程性。上层：应用层负责网络的管理、优化和使用。

1. OpendayLight

处在SDN的控制平面，具有拓扑管理，交换机管理，路径转发，主机管理、网络资源切片管理这5哥基础功能。支持Openflow,Lisp,BGP等协议，多个控制器之间可以采用集群的工作方式。ODL采用基于osgi的karaf作为实现架构，使用maven管理代码，允许在控制器运行时进行能功能模块的安装、删除和更新。下图是ODL架构图：

ＯＤＬ在设计上，自底向上分为４层：南向协议层、服务抽象层、控制层，以及北向接口层。框架视图如图１所示。南向协议层包含Ｏｐｅｎｆｌｏｗ、ＬＩＳＰ、ＧＢＰ等协议插件，负责与实际的网络设备进行通信、收集设备信息、监听相应端口。服务抽象层SAL：屏蔽了不同南向协议插件之间的差异，负责路由和适配的工作，使得南向协议层呈现统一的北向接口控制层的调用。控制层：负责交换机管理等拓扑结构管理等。北向接口层为网络应用程序提供访问的接口，ODL采用REST API



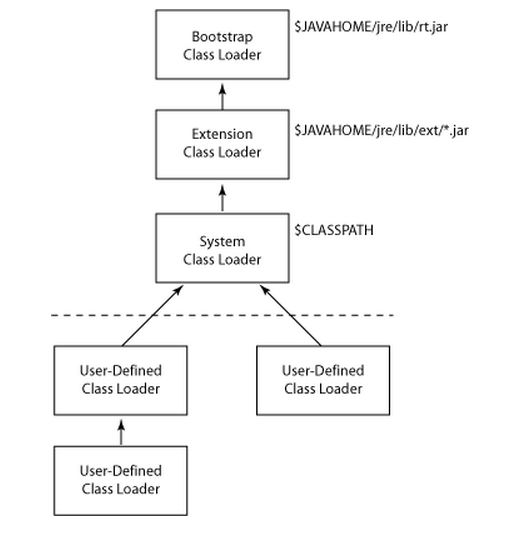
1. OSGI及Karaf

* OSGI <http://www.sdnlab.com/odlcommunity/question/62> <http://blog.csdn.net/liubag/article/details/22825127>

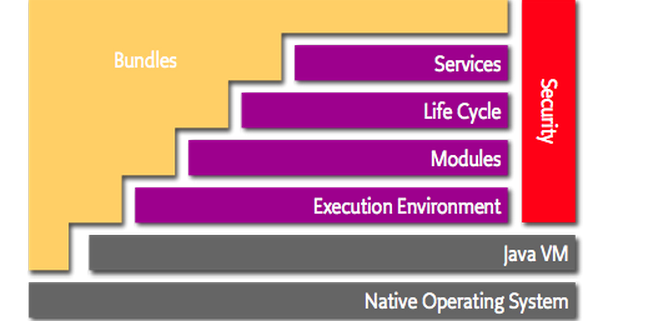
是什么导致了OSGI的产生和发展？？

Java的类加载器(Classloader)是一种分层结构，如下图所示，分为引导类加载器(Bootstrap Class Loader)，扩展类加载器(Extension Class Loader)，系统类加载器(System Class Loader)以及用户定义的类加载器(User-defined Class Loader)。引导类加载器在JVM时负责加载rt.jar里面的类，扩展类加载器负责加载在扩展目录下的jar文件中的类，系统类加载器则在Classpath上面搜索类加载器，用户定义的类加载器则从用户指定的路径（比如一个网络URI）加载类。在该类加载体系中，一个类加载器总是先去上层类加载器加载类，一层一层迭代，当无法找到需要的类时在自己加载。

在这种类加载机制中，存在以下几个问题：1）类版本冲突：当类路径上存在同一个类的不同版本时，如果类加载器找到一个版本，则不再搜索加载下一个版本；2）无法确定jar之间的依赖关系：现有的JAR标准中缺乏对与Jar文件之间依赖关系的定义支持，因此只有在运行时间无法找到所需的类时，才会打出java.lang.ClassNotFoundException，但这通常不能有效帮助开发人员解决问题；3）信息隐藏：如果一个jar在类路径上并且被加载，那么所有该jar中的公共类（public class）都会被加载，无法避免某些类被隐藏从而不被加载。尽管在J2EE中改进了类加载机制，可以支持以war或者ear应用为单元进行加载，但是这些问题还是没有被很好地解决，并且热部署效果让人忧心。



      OSGi就是为了克服这些问题而生，却又不局限与这些问题，对Java开发人员而言的确是好多顿大餐。OSGi是一个动态的Java模块（Module）系统，它规定了如何定义一个Module以及这些模块之间如何交互。每个OSGi的Java模块被称为一个bundle。每个bundle都有自己的类路径，可以精确规定哪些Java包和类可以被导出，需要导入哪些其它bundle的哪些类和包，并从而指明bundle之间的依赖关系。另外bundle可以被在运行时间安装，更新，卸载并且不影响整个应用。通过这种方式，分层的类加载机制变成了网状的类加载机制。在应用程序启动之前，OSGi就可以检测出来是否所有的依赖关系被满足，并在不满足时精确报出是哪些依赖关系没被满足。



OSGi service platform是一个开放并且提供统一接口标准的体系框架，基于这个体系框架，服务提供商，程序开发人员，软件提供商，服务网管运营商，设备提供商能够协调地联合起来开发，部署以及管理向用户提供的各种服务。

灰色部分，是OSGI的基础，可以在任意一个本地操作系统中安装有Java虚拟器上可以运行OSGI.

Execution Environment：是CVM + CDC + FP + PP组成的JVM配置。

Module：这一方面功能将主要负责bundle的安装部署，更新和卸载，以及bundle在设备的物理存储（如果有的话）。在这个层次，每个bundle都是独立的，它的安装，升级和卸载完全不依赖任何其他bundle，这点framework提供了强大的隔离性。

Life Cycle专门负责对bundle的解析（比如关联两个有相互依赖关系的bundle），启动（相当于运行应用程序）和停止（相当于停止应用程序）。这个层次中，bundle间的逻辑关系被创建起来，这些关系能否成功的创建，将会直接影响bundle的成功解析和启动。Service Registry：可以认为是一个数据库，bundle启动后，可以向这个数据库注册它动态提供的服务。只要bundle不被停止，且bundle不主动撤销注册的服务，这个服务将一直保存在这个数据库中供其它bundle来查询和使用。而Services就是由bundle运行时提供的具体服务对象，这些服务对象的存在，使得framework具有极其动态的特征，并为framework运行时提供灵活强大的功能。  
Jar文件是bundle的唯一格式，也就是说，我们要运行bundle，必须把代码打成jar文件。而jar文件可以带有manifest文件，这个文件对bundle是不可缺少的。

* Karaf

Karaf是Apache旗下的一个开源项目.Karaf同时也是一个基于OSGi的运行环境,Karaf提供了一个轻量级的OSGi容器,可以用于部署各种组件。OSGi是OpenDaylight的旧版本，就在氢版本中，他加载每一个模块，如OpenFlow plugin，GUI，controller等；  
Karaf用在Helium氦版本中，他可以加载你需要的模块。Karaf工具是基于OSGi的OpenDaylight特性容器，用于实现OpenDaylight各功能组件的热插拔

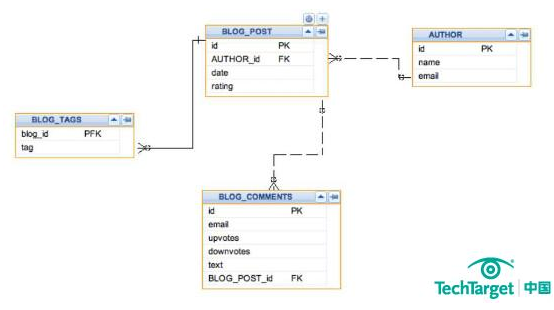
1. 我所做的工作部分。

* Mongodb

Mongodb效率高的原因：首先是内存映射机制，数据不是持久化到存储设备中的，而是暂时存储在内存中，这就提高了在IO上效率以及操作系统对存储介质之间的性能损耗。（毕竟内存读取最快）其次，NoSQL并不是不使用sql，只是不使用关系。没有关系的存在，就表示每个数据都好比是拥有一个单独的存储空间，然后一个聚集索引来指向。搜索性能一定会提高的。

Insert方面MongoDB比SQL Server快十倍左右，Update快20倍左右，Select快2倍左右。mongodb存储量很大可以达到P级别，10TB足够用数据表和数据集，对于存储的大多是文字类的消息是可以用no-sql用类似json的Bson格式存储数据，如果要是存储一片博客的话，包括内容和评论那么可能nosql只在一个document里就存储了，而sql则可能要存储在几个变中。预定义结构VS.动态结构.sql需要预定义好这样的数据结构，后期如果再修改会很麻烦，而nosql是可以动态数据存储。结构化查询和非结构化查询。关系型表中主键的概念对应非关系存储中的文档Id。SQL数据库使用预定义优化方式(比如列索引定义)。

例如：存取一篇博客这里我们举一个例子，博客文章。大多数人在编程的时候会使用“类/对象”的架构来进行建模，但使用关系型数据库来存储博客数据就会遇到麻烦，因为每一个输入的元素会分散到多个表当中。如下图所示：

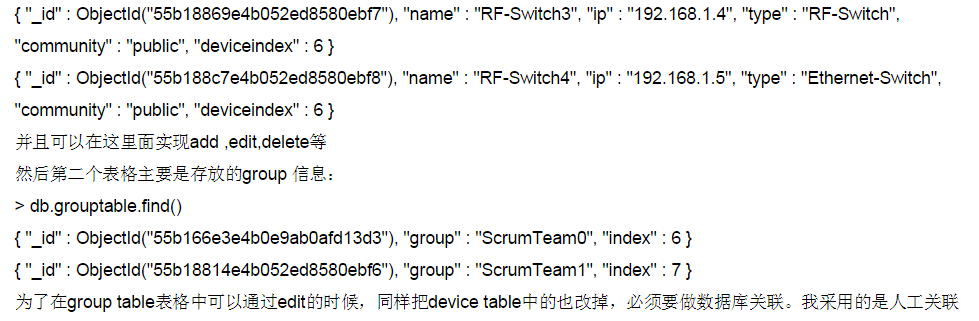


用MongoDB，你的博客文章就能够存储在一个单一的集合当中，每一个元素都很明显。

主要实现的功能是：增删改查以及按照时间在指定的时间内后台数据会自动删除等。

难点以及自己认为比较难实现的地方：就是数据库管理和自动删除的功能

数据库关联的实现：两种方法人工关联和Dbref因为想到利用之前的代码，就想到用人工关联的方式

情景：，device table和group table，device table里面存储的是设备的name,ip,type,group 等，group table里面存储的是group的id 

实现思路：根据device的index去查grouptable中对应的index的group 名字是什么,有个for循环遍历，先将所有的数据得到放到list里面，然后得到每一个DBobject对象，然后把{ "deviceid" : { "$oid" : "55b01e08e4b04009dcbca85c"}变成{ "\_id" : "55b01e08e4b04009dcbca85c" }这种类型，然后去得到index,再用这个index去查对应的name,再put到原来的list里面。实现关联。在edit group的时候只是名字改变，但是index不变。

后台数据库自动删除： 用来Mongodb的ttl(文档的存活时间)。可以设置在数据插入后的一定秒数后过期，也可以设置文档在确定的时钟时间过期（Date）.

Capped collection:可以规定一个colleciton的最大的存储空间和最大document数量。

延伸—行存储和列存储：

* SNMP实现切换：
* Web ui以及LDAP
* Servlet以及java web问到的知识
* Load-balance部分根据device的index去查grouptable中对应的index的group 名字是什么，所有有个for循环遍历，先将所有的数据得到放到list里面，然后得到每一个DBobject对象，然后把{ "deviceid" : { "$oid" : "55b01e08e4b04009dcbca85c"}变成{ "\_id" : "55b01e08e4b04009dcbca85c" }这种类型，然后去得到index,再用这个index去查对应的name,再put到原来的list里面