**Anycmd开源一个通用权限管理框架**

开源地址在<http://git.oschina.net/anycmd/anycmd>

系统中的权限管理大家都很熟悉，实现模式大同小异。研究和尝试实现权限框架的人很多，基本上把这块想明白了并实现出来就不再是初学者了。

RBAC分了好几级，每一级都在说什么这篇文章不去研究也不评价。这篇文章是按照本人的思维模式书写的。在我起初实践的时候我也试图阅读前人的文章，继承前人的知识，但是我看不懂。不是因为他们在遮遮掩掩或者故作高深，而是因为多年前作为一个初学者的我准备不够。虽然很多人认为搞计算机工作充满创造性，但是我还是认为我是搞生产的不是搞科研的，我相信在我搞生产的时候遇到的95%的困难前人都留下有解决方案，只不过我不知道罢了。理论看不懂就去实践，实践有困难就去理论，反复即可，重复的力量可以摧毁世间任何强大之物。

回归正题。

从动词开始。要控制权限首先得明白是系统要发生某种变化，“花是红的”这种陈述句是没有什么权限可以控制的。什么样的动作是潜在的需要被控制的动作呢？这个答案很简单——任何动作。但具体哪些控制是有意义的哪些控制是无意义的这要根据你的业务需求具体分析。本文不是分析具体领域的业务逻辑的而是探讨访问控制的方法的。

回顾：动词（Verb）。有动词才有控制，没动词的陈述句是无需控制的。“太阳会落”不需要控制，“林教授对学生说太阳会落”就可能需要被控制，因为‘林教授“说”’，说是动词。下图的Message就可能是含有动词的Command。

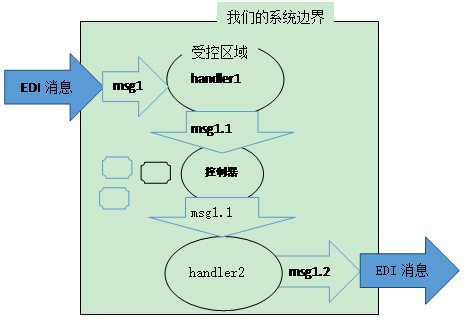


图1

很显然，我们要控制的是cmd，因为它还没有被action（行动），action后就已经成为Event（事实）了没有控制的意义了，下一个控制的机会属于下一个受控区域了。

公式：Command = Action + Input；Action = ResourceType + Verb；Event = handledAction + Output；Message集 = Command集 + Event集。

我没有好的方法来表达我的想法，就让我使用上面的公式来表达吧。这些公式肯定不严谨，比如我都没有定义加法的意义就随意的使用了它。通过这些公式我想表达的意思是Message分两种：一种是Command，它描述的是尚未发生的事情并且它为事情的发生提供了完整的输入；Event是已经发生过的事情。Command和Event分别是输入与输出。大家可以想一想是否是这样的？一条message所表达的事情无非是两种：表达某个主体想要针对某个客体做些什么但尚没有做；表达某个主体对某个客体已经做了什么。

从上图可以看出msg在我们的系统中又分为两个阶段：1进入系统后但尚未进入受控区域前；2在控区域中。

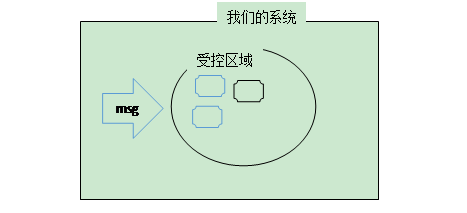


图2

说明：AC缩写的意思是Access Control。

那么msg进入我们的受控区域前会是一个比较好的访问控制策略执行点。如果AC策略设计的合理和使用的得当的话是可以做到仅在一个策略执行点就能完成所有的权限控制工作的。但很多业务系统的功能划分没能做到是完全正交的，这就使得我们无法做到仅在一个执行点来完成所有的AC事务。没关系，在那些难以覆盖到的受控区域前设立专门的执行点就可以了。比如图1中的handler1我们可以认为是展示层的控制器，因为展示层控制器是收集收入的地方所以在它前面放置AC执行点会是最佳位置，而handler2大家可以认为是应用服务或领域服务，如果将AC执行点放置在handler2前能够降低你在唯一的前端控制器前实现AC的困难的话也是可以在这里放置AC逻辑的。

访问控制有很多策略信息点。策略信息点是为AC提供信息的。比如msg的类型就是一个策略信息点，前面我们把msg分类为Command和Event两种，而如果当前收到的msg是条Event的话那就是没有控制的必要的，而只有可能引起系统状态变化的Command才是有控制意义的。比如Command声明的action和提供input也是策略信息点，如果我们的message是像http request message那样的无状态的带有完整的一次请求的信息的message的话那么message已经提供了我们的AC所需的所有策略信息点了。当然通常都是有状态的，UserSession也是我们的策略信息点。

这片文字就写到这里吧，我发现一次性难以把我想要写的内容写完，留到后续再写吧。写这片文字是为了创造一个机会介绍一下我刚刚开源的Anycmd框架。

**Anycmd是什么？**Anycmd是完全开源免费的通用权限管理框架、系统、中间件。

**Anycmd提供了哪些能力？**她会完整的支持RBAC，她会大大超出RBAC定义的能力范围，她计划提供一套完整的容易学习的AC方法论和拿来即可使用的实践框架、中间件、系统。她会比市面上现有的各种AC框架来的更加美丽大方、简洁高效。作者有美好的理想并愿意一直为她付诸努力。

**Anycmd目前构建到了什么程度？**Anycmd已经定义好了Account、Organization、Role、Group、Function、Menu、AppSystem、ResourceType、Privilege这9大AC元素。并且已经完成了这9种元素的元素集的内存管理代码。如果不准球更高级的特性的话那么她现在基本可以使用了（不建议使用）。

**更高级的特性是什么？**AC的每一个元素都应是可以随意的两两组合的，每一种组合都应是有特定的意义的。比如（function1,menu1）可以定义为如果授予主体function1权限那么系统可以自动展示出相应的menu1而无需安全管理员再进行“菜单授权”。比如（role1,role2）的组合可以定义为role1继承role2，当然（role1,role3）表示role1集成role3，你看到了这就是RBAC的层级Role特性在Anycmd中的实现。比如(organization1,group1)可以定义为把整个organization1组织结构下的用户逻辑的加入group1工作组。就举这些例子吧，请注意：Anycmd的9类AC元素对象的任意两两组合都是有意义的。如果您感兴趣的话现在可以先观察Anycmd的源码，期待您为Anycmd提供帮助确保她走在正确的道路上。

**Anycmd的权限控制到什么程度？**她从11个完全正交的维度控制到具体的业务节点的具体的主体对具体资源类型的具体资源实例施加具体的动作，她还深入到这个具体资源实例的具体属性（大家可以认为这是控制到了行列单元格），而定义在资源类型上的动作是和资源的属性正交的，把10个正交的维度作用到“实体属性”这个组织结构上。是的“实体属性”的本质也是对资源的单元划分，只不过它是资源类型节点的子节点，而资源类型的本质也是组织结构。组织结构概念是十分难以理解的，这片文字不说这个，后续我会专门说明。

好了，就到这里。Anycmd的开源地址在<http://git.oschina.net/anycmd/anycmd>

她本身就是自己的Demo，不用专门提供demo。

感谢[daxnet](http://www.cnblogs.com/daxnet)他让我学到了很多。