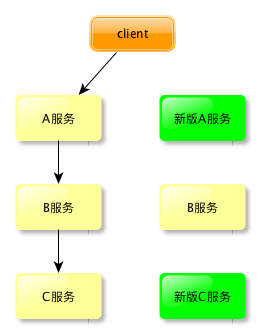
<https://blog.csdn.net/jiangeeq/article/details/80534961>

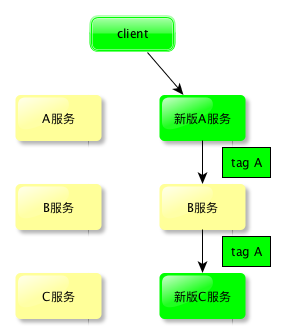
## 调用链上有多个业务服务的场景

这样一个业务场景，假设对外提供了服务Ａ给客户端访问，服务Ａ后面会调用服务Ｂ，Ｃ，Ｄ，此时需要上线一个功能，这个功能涉及到了服务Ａ，Ｃ的修改，但是服务Ｂ，Ｄ不需要变动，换言之，我们的意图是，如果一个客户端请求，走到了新的灰度服务Ａ，那么最终这个请求也应该走到这次和Ａ一起灰度的服务Ｃ上。．

这里的处理策略，可以给客户端请求进行tag打标记的方式，比如经由新版本服务Ａ处理的请求，全部打上tag A，而在服务Ｃ上，也有接入层进行转发，它转发的策略之一就是根据根据这个tag来进行转发。



请求首先走到了旧版本的服务Ａ上，该服务没有对请求打上tag，所以后续访问的都是没有配套灰度的旧版本Ｃ服务



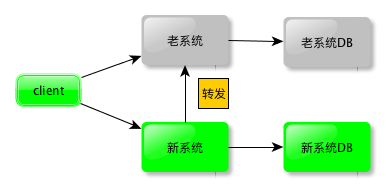
请求首先走到了新版本需要灰度的服务Ａ上，在经过该服务处理后，给请求打上了tag　Ａ，由于带上了tag，后续访问的都是配套灰度的Ｃ服务

总结：涉及到一个调用链路上某几个服务需要灰度的情况，可以通过tag的方式，将走灰度服务的请求汇集到一起来，如果一个请求走到了一个灰度路径上，就打上一个tag，这样只有有这个tag的请求才能走到这条链路上后续也需要一起灰度的服务上

## 涉及到数据的灰度服务

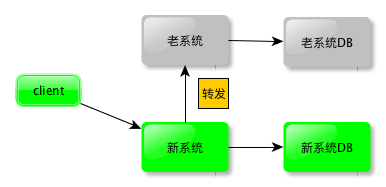
假设灰度的服务，需要使用到数据库，如果数据库前后数据不一致，需要处理的情况就比较复杂，分为以下几种情况：

### 部分灰度



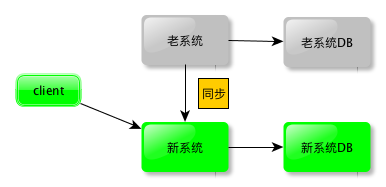
在部分灰度的情况下，有部分请求到旧系统上，另一部分请求到了新的灰度系统上．走到旧系统的请求，还是照原样处理．但是走到了新版灰度系统的请求，需要同时将请求转发给旧系统上来对应的接口上修改旧系统的数据．如果走到新系统的请求查不到该用户的数据，还需要首先同步一份来新系统上．如果是事务性的请求，以写入老系统成功来做为操作成功的标准．

### 全部灰度



在灰度系统已经全部接管了线上流量之后，为了安全起见，仍然需要对新老系统进行双写

### 灰度完成



灰度完成与前面的全量灰度状态不太一样，区别在于前面的全量灰度状态下，仍然不能肯定系统一定是没有问题的，所以需要进行新旧系统的双写来保证数据可以在老系统上进行回滚．而在灰度完成状态，此时认为这个新版本已经完全通过了验证，无需再写入旧系统了．但是此时可能存在部分在灰度期间没有上线的用户，此时需要做一次同步，从旧系统上将这部分数据同步过来．

可以看到，这三个状态下，对新旧系统是否进行双写，做了严格的区分，目的只有一个：一旦新上线的系统出现问题，可以马上撤掉灰度系统，而这期间用户的任何修改在旧系统上都是可以找到的．