多租户的设计方案

一.需求分析

当我们的应用被不同的组织或者机构使用时，并且不同的组织和机构的访问数据是不可见时，此时我们需要针对不同的机构做数据隔离。多租户就是解决数据隔离的一种比较常用的方式。它是从应用上面做软隔离，所有的机构使用的是同一个数据库，通过租户的标识做数据隔离。

二. 常用技术方案

数据隔离常用的技术方案大概分为如下三种

1.独立数据库

数据完全是隔离的，不同的机构使用的是独立的数据库。但是消耗的资源是最大的，相当于每一个机构都需要一套完整的独立的应用。这种方式在私有话部署中比较常见，且可以做定制化的需求。

2.共享数据库，Schema隔离

这种方式是不同的机构使用相同的数据库不同的schema，可以理解比如存在机构A和机构B，那么在mysql中，可以创建数据库healthcare_A，healthcare_B两个库，两个不同的机构使用这两个不同的库。这种方式其实也是很少使用，这种方式不适合大型的微服务项目，原因有如下两点

（1）大型的微服务项目我们会针对业务拆分成多个服务，每个服务对应对应不同的schema（在mysql中），比如我们healthcare_account（表示账号）、healthcare_device（表示设备）、healthcare_map（表示地图导航）等等，如果在mysql中，我们使用这种方式做隔离，就需要每个微服务都创建很多个类似的schema，会导致数据库越来越大不便于维护

（2）使用这种方式我们在应用中需要根据不同的机构切换数据源，这可能会影响性能和复杂度，当然我们可以使用一些中间件来处理切换数据源的问题，但是也会给提高系统的复杂度

3.共享数据库，共享schema

这种方式在数据库层面数据是完全没有隔离的，但是几乎每一个表（共享表除外）中都会包含一个机构的标识字段（tenant_id）,在应用层通过这个tenant_id做数据隔离，比如（select * from account where tenant_id = ubt）这样我们查询出来的数据都是ubt这个机构的账号数据。这种数据隔离的方式成本最低，实现方式最容易，但是也存在如下问题

（1）租户信息如何从前端传递到后端服务

（2）微服务之间调用如何传递租户信息

（3）如何对开发人员更加友好，否则容易造成数据泄漏

（4）某些表其实是全局表，不需要做租户隔离，如何处理

（5）某些表中的某些个别的sql不需要做隔离，如何处理

（6）关联查询时如何保证处理

（7）应用代码的其他地方需要获取租户信息，如何处理

（8）其他协议如何传递租户信息

总结上面问题，其实就是如何容易的保证数据的隔离性

三.落地技术细节

我们使用的是第三种方式，这一节主要是讲述我们在落地的过程如何处理上述问题的技术实现细节

1.租户信息如何从前端传递到后端服务

（1）根据域名获取，每个机构对应不同的三级域名，比如：机构新凯（xinkai.ubtrobot.com），港大（gangda.ubtrobot.com）

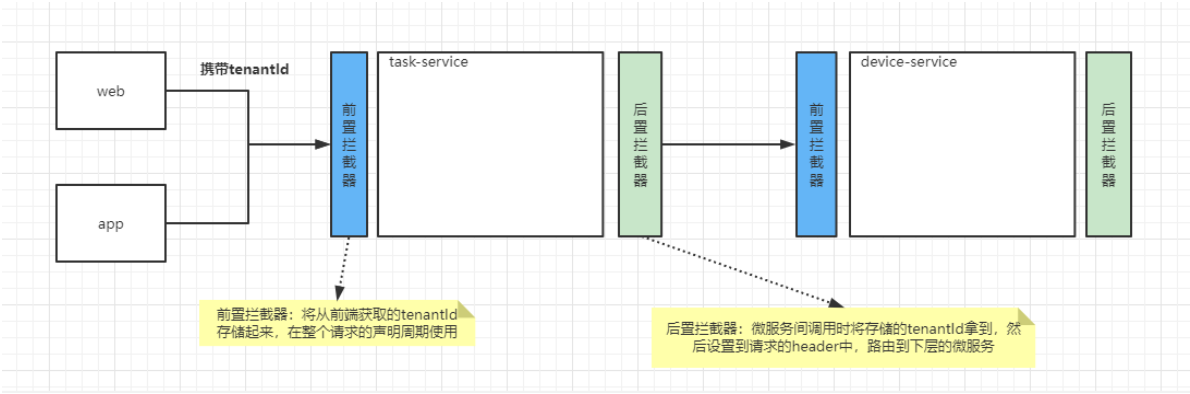
（2）前端将租户信息放到请求的header参数中，比如：当用户登录后，选择机构后，就可以获取到机构信息，此时可以将租户信息放到header中的x-tenant-header，后端直接从请求的header中获取tenantId

（3）放到请求参数中，跟普通的参数一样

我们最终根据产品的需求，以及申请域名的不便性，选择的是第二种方式

2.微服务之间调用如何传递租户信息

由于我们的系统是微服务设计，微服务之间的通信是通过http的请求方式，我们需要解决在微服务调用中，租户信息传递的问题。比如，当我们前端调用task微服务时，此时前端将租户信息传递到task微服务，当task微服务需要调用device服务时，需要将租户信息传递到device微服务，否则device服务在执行sql操作时会造成数据泄漏。我们的处理方式是在微服务调用时使用拦截器如下图所示



通过前置拦截器和后置拦截器，可以实现微服务间调用时传递租户信息

3.如何对开发人员更加友好

当我们在使用sql过滤数据时，我们前面说过了在sql后面加上tenantId=xinkai这种方式过滤数据，可以实现数据的隔离。但是如果我们在应用中通过硬编码的方式这样追加条件会带来如下问题

（1）工作量过于繁重，整个微服务的接口数量可能有大几百或者上千的接口，每一个接口都需要手动带上这个参数会非常的麻烦

（2）但凡由于开发人员疏忽，忘记了带上这个条件，就会造成数据泄漏，这是非常危险的

（3）人员的交替，每个新入职的新人加入这个项目时，都需要同样的方式，这对他们也非常不友好

我们当然希望有一种自动追加的这个tenantId这个条件方法，这样我们在应用层就不需要过渡关注这个问题。好在我们使用的mybatis-plus提供这样的插件，该插件也是一个类似拦截器（其实是代理），在执行sql语句之前，自动在插件的逻辑中追加这个条件。比如当我们编写（select * from device where device_name="CR"，经过插件处理后，自动会帮我们加上租户的过滤条件，select * from device where device_name="CR" and tenant_id="xinkai"）。如此一来开发者就不需要关注每个sql语句了。

4.某些全局表其实是不需要通过租户信息过滤

一些全局表不需要使用租户信息过滤，因为全局表，是针对所有的租户都需要使用的表，比如机器人类型表。每个租户都是这些机器人类型，我们不需要针对每个租户冗余一份机器人类型的表（除非后面有不同的机构有不同的机器人类型）。针对这种情况，我们对mybatis-plus的插件做了优化，通过配置参数的形式，设置全局表（ignoreTables），当插件发现查询的表示ignoreTables里面，那么就不会自动添加tenantId条件

5.非全局表的部分sql不需要租户信息过滤

其实这种情况不常见，这种方式就需要半手动处理了，因为只有开发人员自己知道哪些特殊情况下不需要租户信息过滤。但是mybatis-plus也拓展了这种简单的处理方法。就是在我们需要过滤的sql操作的接口上面加上**@InterceptorIgnore(tenantLine = "true")**这个注解，插件在执行sql时如果发现该方法上面有这个注解时，就不会自动添加tenantId信息。比如我们的应用启动时，将设备缓存到内存中，这个时候，就不需要过滤租户信息，因为查询的是全量信息，此时这个功能就使用到了。

6.关联查询

关联查询可以使用上面的5的方式处理（5的方式因为是半手动，所以任何情况都能使用这个方式），mybatis-plus也支持别名，只需要在关联查询时给表设置别名，则会自动正确追加tenantId信息，否则就会报错。

7.如何在任何地方获取租户信息

我们需要将租户信息保存在整个请求的声明周期内都能访问到的地方，因为我们在任何地方都有可能会执行sql语句。我们将租户信息保存在线程上下文中，这样我们就可以在没有切换线程的时候的任何地方都能获取到该租户信息。但是当我们使用线程池或者使用子线程时，会导致租户信息失效，这个问题后面会在单独的设计文档中详细说明解决方案，因为很多地方都是用到了类似的功能

8.其他协议如果传递租户信息

前面说到，我们前端调用http的方式将租户信息放到header中传递到服务端，微服务之前的调用通过自定义拦截器路由租户信息，以及获取保存租户信息。但是我们的请求入口除了http请求的方式，还有mqtt协议的请求方式，这个我们应该如何保证数据的安全性呢？我们的处理方式是当机器人通过mqtt协议向平台发起请求时，我们拿到机器人的唯一标识（deviceNo），然后通过这个唯一标识获取这个机器人当前属于哪个机构，然后将机构信息设置在线程上下文中，之后的微服务调用逻辑和上面类似