**数据接口设计**

# 类设计

## 底层基类

以下类，对用户透明，且私有默认构造，杜绝用户直接使用

* IOBus（总线类）

所有接口类的基类，把接口看做一条条的IO线路。

唯一的非模版基类，所有的接口功能都在此类中实现

其它模版接口类仅仅只是实现一些数据之间的转换、包含、拷贝等操作，真正要了IO就是调用一下基类方法

* KVData（key-value数据模版类)

IOBus直接子类。主要目的就是基类模版化，因为任意类型数据在Exist上都是一个key-value数据，所以取KV打头

* Container（容器模版类）

从KVData派生。所有容器接口类(Map, Vector==)的基类

* Value（数值模版类）

从KVData派生。所有数值接口类(Int8,Int16,Int32,Float==)的基类

## 接口类

为用户开放的类，用户使用这些类，就像使用普通类一样访问数据

* Key模版类

某些从其有2种类型的下标Key和int，比如Map ==,而key有可能也是int，

当key也是int类型时，[n]就没办法分辨应该取第n个元素，还是取key=n的元素,所以需要此类辅助。

使用方法:容器对象[(Exist::Key<int>)n]

再比如Delete(n),也一样Delete没办法知道n是元素位置，还是key,也需要

Delete((Exist::Key<int>)n)来告诉Delete()n是key，不是元素位置

* Map模版类

从Container派生，可嵌套KVData派生类

* Vector模版类

从Container派生，可嵌套KVData派生类

* Element元素模版类

没有基类，表示数据包含的元素，可嵌套KVData派生类

Container以及子类是包含n个Element的数据此类是为了将IO与取元素的动作分开，否则map[] = a，[]运算符首先取元素，然后无法得知是应该到Exist上写还是读，所以需要经过此类过度一下，=运算符就知道是要到Exist上去写，而不是读了

Value以及子类是只有1个Element的数据

Value其实不用包含Element，但是为了代码重用Element的IO访问逻辑，也避免IO访问逻辑需要改变时，出现多个需要修改的地方，所以将Value以及子类设计成只有1个元素的数据

* Int8整型类长度1byte

从Value派生

* 其它容器接口类、数值类型类

参照以上类定义

# 设计思想