参考：<http://blog.csdn.net/fuckyouone/article/details/9446499>

这是对avro官方文档的直译：

avro是一个数据序列化系统

**提供了：**

* 丰富的数据结构
* 压缩的，快速的，二进制的数据格式
* 一种文件格式，用于存储持久化数据
* 一个RPC系统
* 和动态语言的简单交互。并不需要为数据文件读写产生代码，也不需要使用或实现RPC协议（也就是说不和RPC集成）。代码生成是一种优化方式，但是只对于静态语言有意

**schemas：**

      所谓schemas，就是一种描述文件，能够说清楚你的对象的描述文件。可能有你对象里面有什么，   他们之间的逻辑关系，数据的统计信息。比如说对人的描述包括身高、体重、三维、性别等，只要他的格式确定，就可以叫做schema。

（me：其实就是将简单的pojo类的信息以另外一种格式描述了出来,生成好文件后我们可以在此基础上进行修改，修改某些方法的实现）

      Avro也是依赖于schema的（为了跨语言，也为了持久化，总要有二进制数据解析的指导）。当读取Avro数据的时候需要使用到schema，   当写数据的时候也需要有schema的指导。这样就允许了每个数据都没有前缀开销（因为可以用schema来理解数据），这就会使得序列化很快而且数据会比较小。

      当把Avro数据存储到文件的时候，会把他的schema数据一起存储，这样数据就可以被接下来的任意程序使用了。如果程序希望用其他的schema来读取数据，是很简单的，因为你可以比较两个schema来看能不能读取数据

      当Avro用于RPC的时候，客户端和服务端可以在连接握手的阶段进行schema交换。（这种方式是可以被优化的，因为对于很多schema来说，是没有schema的交换的）由于两边都有对方完整的schema存在了，所以可以比较出同名域、缺少的域、多余的域。

     avro的schema是用json定义的。这样有json包的语言（如java的jackson）就会比较容易实现了。

**和其他序列化系统的比较：**

    avro提供了功能上和Thrift、Protocol Buffers等相同的特性。但也有自己的不同点：

* 动态类型：Avro不需要生成代码，数据总是和schema放在一起的，使得不用代码生成或静态类型就可以做所有必要的处理这种特性使得使用简单语言构造一般的通用系统变得很方便
* 没有加标签的数据：因为在读取数据的时候，是有schema在的，所以很少的类型的信息需要加载进入数据中，并且序列化数据更小
* 没有手动的序列化ID：当一个schema变化后，新旧schema在处理数据时就会都有，这样两者之间的不同就可以直接通过符号来处理，也就是属性名称

//================20171021