## JDBC核心文档

## JDBC设计目的

鉴于现用的mdsplus有诸多不便，未来也需要更加快速、方便，支持多种格式的科学型数据库，利用集群技术，实现JDBC。

## JDBC整体结构

### JDBC框架

CoreService操作mongodb存储Experiment和Signal相关的信息，Cassandra主要负责payload的信息。



### JDBC操作关系

CoreService和StorageEngine是底层操作，CoreApi对CoreService和StorageEngine进行了封装。FixedWaveDataTypePlugin实现了IDataTypePlugIn接口；PathQueryPlguIn和IdQueryPlugIn实现IQueryPlguIn接口。这些实现的接口被添加到CoreApi里，Web即可操作。



### JDBC接口设计

存储引擎方面：

IStorageEngine：最底层接口，定义了数据库的初始化和清空，删除数据，拷贝数据等最基本的操作。

IMatrixStorageEngineInterface：中间层接口，定义了矩阵型StorageEngine的操作形式，未来有别的StorageEngine形式，比如图形、文字型等，可在这里定义。

信号处理方面：包括信号操作的接口和信号基本信息的接口。

信号操作的接口：

IDataTypePlugIn：定义了插件的名字，创建信号，读写信号几个操作。

string Name; int ScoreDatatype(string dataType);

Dictionary<string, object> GetInformation<T>(Signal signal);

Task<object> GetDataAsync<T>(Signal signal, string fragment, string format);

Task<ICursor> GetCursorAsync<T>(Signal signal, string dataFragment);

Task PutDataAsync<T>(Signal signal, string fragment, object data);

Signal CreateSignal(string datatype, string name, string iniString);

IQueryPlugIn：定义了插件的名字，查询条件评分，查找entity操作。

string Name; int ScoreQuery(string query);

Task<IEnumerable<JDBCEntity>> FindJdbcEntityAsync(string query);

信号基本信息接口：

ITypedSignal:定义了信号的基本操作，通过IDataTypePlugIn来执行读写数据。

插件接口：

最后实现的都是通过插件接口来进行数据操作。

FixedWaveDataTypePlugin:定义了CoreService和IMatrixStorageEngineInterface，然后实现IDataTypePlugIn接口里的方法，即可对数据进行操作。

同理查询插件IdQueryPlugIn和PathQueryPlugIn。



## JDBC存储引擎

2个的区别主要是Cassandra每次都会增加一个新的Payload，Mongodb可以选择增加新Payload还是在最后面的Payload上继续写。

查找Cursor时，Cassandra和Mongodb都是一个维度一个维度的存储下来，不同的是，Mongodb一个维度就是一个Cursor，最后查找完，有几个维度就有几个Cursor，这些Cursor放在一个List里，展现给用户唯一一个Cursor；Cassandra只有分页的概念，一个维度的所有Payload存下来，如果没有超出页的大小，比如1000，有几个维度就有几页，如果超出了页的大小，比如某一维度有1500个Payload，就需要2页。

### Cassandra Storage Engine

主要由四个函数构成：

* public async Task<ICursor<T>> GetCursorAsync<T>(Guid signalId, List<long> start, List<long> count, List<long> decimationFactor = null)

例子：

var cursor = storageEngine.GetCursorAsync<double>(sig11.Id, new List<long> { 2, 1, 3 }, new List<long> { 2, 2, 3 }, new List<long> { 2, 2, 2 }).Result;

List<double> result = cursor.Read(3).Result.ToList();

double[] assertresult = { 213, 215, 217 };

通过GetCursorAsync得到4个维度的所有payload,然后通过decimationFactor的最后一个数即为Factor来算间隔点，通过cursor里的Read函数实现，如果一个维度有10个payload,会遍历所有10个payload,比较Start和End，利用Factor来读取数据。

下面是2种添加数据的方法，后面一个指定开始位置和结束位置。

* public async Task AppendSampleAsync<T>(Guid signalId, List<long> dim, List<T> samples, bool createNewSEPayload = false)
* public async Task AppendSampleAsync<T>(Guid signalId, List<long> dim, List<T> samples, long start, long end, bool createNewSEPayload = false)

### MongoDB Storage Engine

* public async Task<ICursor<T>> GetCursorAsync<T>(Guid signalId, List<long> start, List<long> count, List<long> decimationFactor = null)
* public async Task AppendSampleAsync<T>(Guid signalId, List<long> dim, List<T> samples, bool createNewSEPayload = false)

## JDBC-Web端

Global.asax启动