Condition データベースの開発マニュアル

— DAQ ミドルウエア用 condition データベースを作成するために —

安 芳次
千代浩司

初版：2009 年 6 月 10 日
修正版：2009 年 7 月 2 日
修正版：2009 年 7 月 3 日
修正版：2010 年 8 月 3 日

目次

1 はじめに

condition データベースは DAQ ミドルウエアコンポーネントの固有のパラメータを定義し利用されるもので、主にコンポーネント開発者・利用者によって開発・改良されるものである。一方、configuration データベースは DAQ ミドルウエアのシステム記述に利用されるもので、システム開発者・管理者によって開発・改良されるものである。condition データベースも configuration データベースも XML で記述される点は共通している。ここでは condition データベースの開発・改良に関連する情報を提供するものであり、configuration データベースについては触れない。最後の参考文献の節にソースコードの入手 URL が書いてある。

2 condition データベースの概要

condition データベースはすでに述べたように DAQ ミドルウエアコンポーネントの固有のパラメータを XML で定義し利用するものであるが、configuration データベースが直接 XML ドキュメントを DAQ オペレータによって解釈されるのとは違って、一旦 JSON と呼ばれるドキュメントに翻訳されてから、DAQ ミドルウエアコンポーネントによって翻訳されるものである。

XML ドキュメントは configuration データベースがそうであるように、基本となるものであるが、DAQ ミドルウエアコンポーネントの負荷を軽減するため、より軽量な JSO N ドキュメントの C++ 用翻訳処理系を利用することにした。実験条件が変われば、DAQ ミドルウエアコンポーネント用の XML/JSON ドキュメントは変わるかもしれない。そのため、XML → JSON 変換用コマンドを用意した。XML ドキュメントは実験条件を示すもので保存されるべきだが、JSON ドキュメントは中間ドキュメントなので保存する必要はない。

例題を挙げてその利用の仕方を示すことにする。例えばつぎのような XML の定義があるとする。XML ファイル：

```
<?xml version="1.0" encoding="utf-8" ?>
<condition>
    <daq id="0">
        <module id="0">
            <type>test0</type>
            <value>0</value>
        </module>
        <module id="1">
            <type>test1</type>
            <value>1</value>
        </module>
    </daq>
</condition>
```

上記の XML ドキュメントは次のコマンドで JSON ファイルが作られる。

```
% Xalan o condition-tmp.json condition.xml xml2json-with-attribute.xslt
```

```
% python extract.py condition-tmp.json > condition.json
```

あるいは同様なことを行うスクリプトは、DAQ-Middleware 1.0.0 がインストールされている環境では /usr/bin/condition_xml2json としてインストールされているのでこれを使用してもよい。

```
condition_xml2json condition.xml
```

生成される JSON ファイルは下記の通り (JSON ファイル) (紙幅の関係で複数行に別れていますが実際には 1 行です):

```
{"condition":{"daq":{"@id":0,"module":[{"@id":0,"type":"test0","value":0},
{"@id":1,"type":"test1","value":1}]}}}
```

生成された JSON ファイルを DAQ ミドルウエアコンポーネントが扱うために Condition クラスが用意されている。このクラスは固有のデータベースを扱う基本となるもので、通常はこのクラスを継承した新しいクラスを設計する。ただし、パラメータが少ない場合は、あえて新しいクラスを作らなくてもより簡単に利用できるので、まずは固有のクラスを設計する場合について触れ、そのあとに既存のクラスをりようするだけの場合についても触れることにする。

3 XML ファイルのパラメータから key と value のペアを作成するための基準

XML によって定義されたパラメータは、DAQ コンポーネントの中では、key と value からなるペアの集合で保持される。そのパラメータを利用するためには XML からどのように key と value に変換されるかを知る必要がある。

一般に XML は tag と attribute で任意に定義できるが、パラメータ記述を複雑にしないため、簡単な規則を作った。Attribute で利用できるものは、“id” のみ。tag は任意の名前を付けることができる。必ず condition tag から始まり、condition tag で終わる。key の名前は tag を attribute を “_” でつないでゆく形で決まる。たとえば、概要で出てきた XML の一部である、

```
<daq id="0">
    <module id="0">
        <type>test0</type>
```

この場合、key は daq_0_module_0_type となり、value は test0 である。value のタイプには string と digit(hex/dec) があるだけである。

また、XML が複雑になると、key の名前が長くなる。DAQ コンポーネントで扱いやすいように、prefix の値をセットアップすれば長い名前で参照せずに、たとえば、daq_0_module_0_という prefix を利用すると、DAQ コンポーネントでは “type” を find メソッドでサーチするだけで済む。

4　固有のクラスの設計

DAQ コンポーネントの中で使用する固有のパラメータが次のようなデータ構造を持っていると仮定する。つまり、string の type という変数と int の value という変数を持つ構造体が sampleParam とすると、下記のようになる。

```
struct sampleParam {
    std::string type;
    int value;
};
typedef struct sampleParam sampleParam;
```

そこで、新しいクラス ConditionSample クラスはたとえば下記のように設計する。必要となる include ファイルは

```
#include <string>
#include “Condition.h”
```

クラスは

```
class ConditionSample : public Condition {
public:
    ConditionSample();
    virtual ~ConditionSample();
    bool initialize(std::string file);
    bool getParam(std::string prefix, sampleParam* sampleParam);
private:
    Json2ConList m_json2ConList;
    conList m_conListSample;
};
```

上記のコードで現れる Json2ConList クラスと conList クラスは、前者が JSON ファイルを読み込み key と value の組からなるデータ構造（実際は std::map を利用）に格納するのに対して、後者は key と value の組からなるデータ構造を定義するものである。これらは private に定義されているように内部で使用するもので、getParam メソッドの中で利用される。従って、ConditionSample クラスの利用者は、getParam メソッドを利用してパラメータを引き出すことができる。上記の例でいえば、prefix と getParam メソッドの内で指定する key が連結されデータ構造体への key となる。上記のコードをまとめて ConditionSample.h とする。クラスの実装部分では実際のコードを見ると、下記のようになっている。この例で行くと、prefix+“type” という key でデータをサーチし、また prefix+“value” という key でサーチして、サーチに成功すれば sampleParam 構造体にデータがセットされ、失敗すれば返り値が false となる。

```
bool ConditionSample::getParam(std::string prefix, sampleParam* sampleParam) {
    setPrefix(prefix);
    std::string type;
    int value;
    if( find("type", &type)) {
        sampleParam->type = type;
        std::cout << prefix+ "type = " << type << std::endl;
    } else {
        std::cout << prefix + "type is not found" << std::endl;
        return false;
    }
    if( find("value", &value) ) {
        sampleParam->value = value;
        std::cout << prefix+ "value = " << value << std::endl;
    } else {
        std::cout << prefix + "value is not found" << std::endl;
        return false;
    }
    return true;
}
```

4.1 固有のクラスのその他の設計

initialize メソッドの説明をする。下記はそのコードである。file は JSON ドキュメントで、入力されたファイルは m_json2ConList.makeConList メソッドで内部データ構造に変換される。

```
bool ConditionSample::initialize(std::string file) {
    if (m_json2ConList.makeConList(file, &m_conListSample) == false) {
        std::cerr << "### ERROR: Failed to read the Condition file"
                  << std::endl;
        std::cerr << "### Condition file: " << file
                  << std::endl;
        return false;
    }
    init(&m_conListSample);
    return true;
}
```

変換された内部データ構造は conList クラスで処理されるようになる。

4.2 固有のクラスを使ったサンプルプログラム

下記のようなサンプルプログラムを作ってみた。Sample.h は

```
#include “ConditionSample.h”
```

Sample.cpp は

```
#include <iostream>
#include “Sample.h”
int main() {
    ConditionSample myCondition;
    try {
        if (!myCondition.initialize("condition.json")) {
            std::cerr << "initialization error" << std::endl;
            return 0;
        }
        sampleParam sampleParam;
        sampleParams sampleParams;
        if (myCondition.getParam("daq_0_module_0_", &sampleParam))
            sampleParams.push_back(sampleParam);
        else
            throw "error for getting daq_0_module_0_";
        if (myCondition.getParam("daq_0_module_1_", &sampleParam))
             sampleParams.push_back(sampleParam);
        else
             throw "error for getting daq_0_module_1_";
    }
    catch (const char* str) {
        std::cerr << str << std::endl;
    }
    catch (...) {
        std::cerr << "some error..." << std::endl;
    }
}
```

5　固有のクラスを作らない利用法

固有のクラスを使わないで直接 Condition.h を参照して condition データベースを利用する方法を示す。ConditionSample で利用した方法をそのままクラスではなくメインプログラムで書いてしまおうということである。Sample プログラムとの違いは Json2ConList クラス、conList クラス、Condition クラスが直接見えていることで、パラメータの数が少ないうちは下記のようなプログラムでもシンプルに書けるのでよいかもしれない。しかし、パラメータが多い場合はオブジェクトを作ってそれを利用するという方法が優れている。

```
#include <iostream>
#include <string>
#include "Condition.h"

int main(int argc, char** argv) {
    Json2ConList m_json2ConList;
    conList m_conList;
    Condition m_condition;
    std::string file = argv[1];
    if (m_json2ConList.makeConList(file, &m_conList) == false) {
        std::cerr << "### ERROR: Failed to read the Condition file"
                  << std::endl;
        std::cerr << "### Condition file: " << "condition.json"
                  << std::endl;
    }
    m_condition.init(&m_conList);
    std::string prefix = argv[2];
    m_condition.setPrefix(prefix);
    std::string type;
    int value;
    if (m_condition.find("type", &type)) {
        std::cout << prefix+ "type = " << type << std::endl;
    } else {
        std::cout << prefix + "type is not found" << std::endl;
        return false;
    }
    if( m_condition.find("value", &value) ) {
        std::cout << prefix+ "value = " << value << std::endl;
    } else {
        std::cout << prefix + "value is not found" << std::endl;
        return false;
    }
}
```

6 サンプルプログラムの Makefile とその実行

Sample プログラムや Simple プログラムをコンパイルするには関連するファイルを必要とする。JSON ファイルを解析するために JSON spirit と呼ばれるツールを使っている。JSON spirit 関連ファイルには json_spirit.h および関連 include ファイルと libJsonSpirit.so ライブラリが必要である。また、JSON を condition の List に変換・格納するためのファイル群として、json2conlist.h(class Json2ConList) や Condition.h がある。これらのファイルは DAQ-Middleware 1.0.0 がインストールされている環境ではインクルードファイルについては `/usr/include/daqmw/`、ライブラリファイルについては `/usr/lib/daqmw/` にインストールされている。Makefile は下記のとおりである。小数の扱いに boost_regex を使っている点に注意を要する。

```
CC       = g++
PROG     = Sample
CXXFLAGS  = -g -O0 -Wall
CPPFLAGS += -I/usr/include/daqmw
LDLIBS   += -L/usr/lib/daqmw -lJsonSpirit -lboost_regex
```

```
all: $(PROG)

OBJS += ConditionSample.o
OBJS += Sample.o

$(PROG): $(OBJS)

$(PROG).o: $(PROG).cpp $(PROG).h
ConditionSample.o: ConditionSample.cpp ConditionSample.h Condition.h

clean:
    rm -f *.o ${PROG}
```

その上で、プログラムを実行すると、

```
./Sample
ConditionSample created
key: daq_0_module_0_type
daq_0_module_0_type = test0
key: daq_0_module_0_value
daq_0_module_0_value = 0
key: daq_0_module_1_type
daq_0_module_1_type = test1
key: daq_0_module_1_value
daq_0_module_1_value = 1
ConditionSample deleted
```

Simple のプログラムであるが、下記のようにして使う。

```
% ./Simple condition.json daq_0_module_0_
key: daq_0_module_0_type
daq_0_module_0_type = test0
key: daq_0_module_0_value
daq_0_module_0_value = 0
```

7 資料

7.1 Condition.h のソースコード

```
#ifndef CONDITION_H
#define CONDITION_H

#include <ctype.h>
#include "json2conlist.h"

class Condition
{
    public:
        Condition() {;}
        ~Condition() {;}

    public:
        void init(conList* cList) {
            m_cList = cList;
            m_prefix = "";
        }

        void setPrefix(std::string prefix) {
            m_prefix = prefix;
        }

        bool find(std::string key, void* value) {
            std::cerr << "key: " << m_prefix << key << std::endl;
            conIt it = m_cList->find(m_prefix+key);
            if (it == m_cList->end()) {
//                cerr << "not find !" << endl;
                return false;
            }

            std::string second = it->second;
            type t = check(second);
            switch (t) {
            case type_digit:
            case type_xdigit:
                char *e;
                *(unsigned int*)value = (unsigned int)strtoul(second.c_str(), &e, 0);
                break;
            default:    // type_string
                *(std::string *)value = second.c_str();
                break;
            }
            return true;
        }

    protected:
        void printInt(std::string name, unsigned int value) {
          std::cout << name << ":" << value << std::endl;
        }

        void printString(std::string name, std::string value) {
          std::cout << name << ":" << value << std::endl;
        }
```

```
54
55      private:
56          enum type {
57              type_digit,
58              type_xdigit,
59              type_string
60          };
61
62          bool isDigit(std::string str) {
63              for (int i = 0; i < (int)str.size(); ++i) {
64                  if (isdigit(str[i]) == false) {
65                      return false;
66                  }
67              }
68              return true;
69          }
70
71          bool isXdigit(std::string str) {
72              if (str[0] != '0') {
73                  return false;
74              }
75              if (str[1] != 'x' && str[1] != 'X') {
76                  return false;
77              }
78              for (int i = 2; i < (int)str.size(); ++i) {
79                  if (isxdigit(str[i]) == false) {
80                      return false;
81                  }
82              }
83              return true;
84          }
85
86          type check(std::string str) {
87              if (isDigit(str) == true) {
88  //                std::cout << "digit !" << std::endl;
89                  return type_digit;
90              }
91              if (isXdigit(str) == true) {
92  //                std::cout << "xdigit !" << std::endl;
93                  return type_xdigit;
94              }
95  //            std::cout << "string !" << std::endl;
96              return type_string;
97          }
98
99      private:
100         conList* m_cList;
101         std::string m_prefix;
102 };
103 #endif
```

7.2 ConditionSample.h のソースコード

```
#ifndef CONDITION_SAMPLE_H
#define CONDITION_SAMPLE_H

#include <string>
#include "Condition.h"


struct sampleParam
{
    std::string type;
    int value;
};

typedef struct sampleParam sampleParam;

typedef vector< sampleParam > sampleParams;

class ConditionSample : public Condition
{
public:
    ConditionSample();
    virtual ~ConditionSample();

    bool initialize(std::string file);
    bool getParam(std::string prefix, sampleParam* sampleParam);
    bool getParams(std::string prefix, sampleParams* sampleParams);

private:
    Json2ConList m_json2ConList;
    conList m_conListSample;
};

#endif
```

8 参考文献

1. World Wide Web Consortium, Extensible Markup Language (XML), http://www.w3.org/XML/
2. Apache Software Foundation, Xalan, http://xalan.apache.org/
3. Crockford D, The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627 (Introducing JSON, http://www.json.org/)
4. JSON, http://www.json.org/json-ja.html
5. JSON Spirit, http://www.codeproject.com/KB/recipes/JSON_Spirit.aspx
6. Keita Kitamura, xml2json.xsl
7. ソースコード http://daqmw.kek.jp/src/Condition-for-DAQ-Middleware-1.0.0.tar.gz