**序列化结构v0.9**

**研发三部**

模块名： <weedong/core/seq/seq.h>

依赖: <weedong/core/bstream/bstream.h>

简介(详细请看<weedong/core/ seq/seq.h > )：

提供一套宏定义对数据结构进行定义，并提供如下功能:

1. 支持序列化结构继承
2. 支持动态字符串( C++[ std::string ], AS3[ String ] )
3. 支持动态数组 ( C++[ std::vector ], AS3[ Vector. ] )
4. 支持动态字典 ( C++[ std::map ], AS3[ Object ] )
5. 提供 read write 接口对二进制数据流(<weedong/core/bstream/bstream.h>)进行操作
6. 通过与客户端中间件 Seq 的配合，使用服务器客户端对数据流进行统一接口处理

结构/类:

class CSeq; //序列化结构基类

{

void read( wd::CStream &stream );

void write( wd::CStream &stream );

void clear(void);

}

支持类型:

支持的基本类型有如下:

int8 uint8 int16 uint16 int32 uint32 float double

支持的对象类型如下:

string : 通过std::string 作为string 类型的实现方式

object : 只支持作为CSeq 的派生类

支持的容器类型有如下:

array : 通过std::vector 作为array 类型的实现方式

map : 通过std::map 作为map 类型的实现方式, key 统一为std::string

容器的元素类型接受范为是: [ 所有基本类型, 所有对象类型]

示例：

#include <weedong/core/seq/seq.h>

#include <weedong/core/seq/seq\_log.h>

#include <weedong/core/bstream/bstream.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

定义CSampleBase

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CSampleBase : public wd::CSeq

{

public:

uint16 id; //Id号

std::string name; //名称

CSampleBase()

{

id = 65535; //Id号

name = "示例基类描述"; //名称

}

void write( wd::CStream &stream ){ loop( stream, CSeq::eWrite ); }

void read( wd::CStream &stream ){ loop( stream, CSeq::eRead ); }

void loop( wd::CStream &stream, CSeq::ELoopType type )

{

wd::CSeq::loop( stream, type );

TFVarTypeProcess( id, type, stream );

TFVarTypeProcess( name, type, stream );

}

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

定义CSample, 继承CSampleBase

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

class CSample : public CSampleBase

{

public:

int8 var1; //int8变量

uint8 var2; //uint8变量

int16 var3; //int16变量

uint16 var4; //uint16变量

int32 var5; //int32变量

uint32 var6; //uint32变量

float var7; //float变量

std::string var8; //字符串变量

CSampleBase var9; //对象变量

std::vector< std::string > var10; //对象数组变量

std::map< std::string, CSampleBase > var11; //对象词典变量

CSample()

{

var1 = -1; //int8变量

var2 = 1; //uint8变量

var3 = -2; //int16变量

var4 = 2; //uint16变量

var5 = -3; //int32变量

var6 = 3; //uint32变量

var7 = -3.1415f; //float变量

var8 = "研发三部"; //字符串变量

}

void write( wd::CStream &stream ){ loop( stream, CSeq::eWrite ); }

void read( wd::CStream &stream ){ loop( stream, CSeq::eRead ); }

void loop( wd::CStream &stream, CSeq::ELoopType type )

{

CSampleBase::loop( stream, type );

TFVarTypeProcess( var1, type, stream );

TFVarTypeProcess( var2, type, stream );

TFVarTypeProcess( var3, type, stream );

TFVarTypeProcess( var4, type, stream );

TFVarTypeProcess( var5, type, stream );

TFVarTypeProcess( var6, type, stream );

TFVarTypeProcess( var7, type, stream );

TFVarTypeProcess( var8, type, stream );

TFVarTypeProcess( var9, type, stream );

TFVarTypeProcess( var10, type, stream );

TFVarTypeProcess( var11, type, stream );

}

};

int main(int argc, char\* argv[])

{

CSample Sample;

CSample copySample;

wd::CStream stream; //二进制流

std::vector<uint8> binaryBuffer; //二进制Buffer

//对数组进行操作

Sample.var10.push\_back( "研发一部" );

Sample.var10.push\_back( "研发二部" );

Sample.var10[1] = "研发三部";

//对词典进行操作

Sample.var11[ "研发四部" ] = CSampleBase();

//输出到二进制流

Sample >> stream;

//输入到序列化结构

stream.position(0); //重置指针

copySample << stream; //相当于 copySample = Sample;

//输出到二进制buff

stream.position(0); //重置指针

stream >> binaryBuffer;

return 0;

}