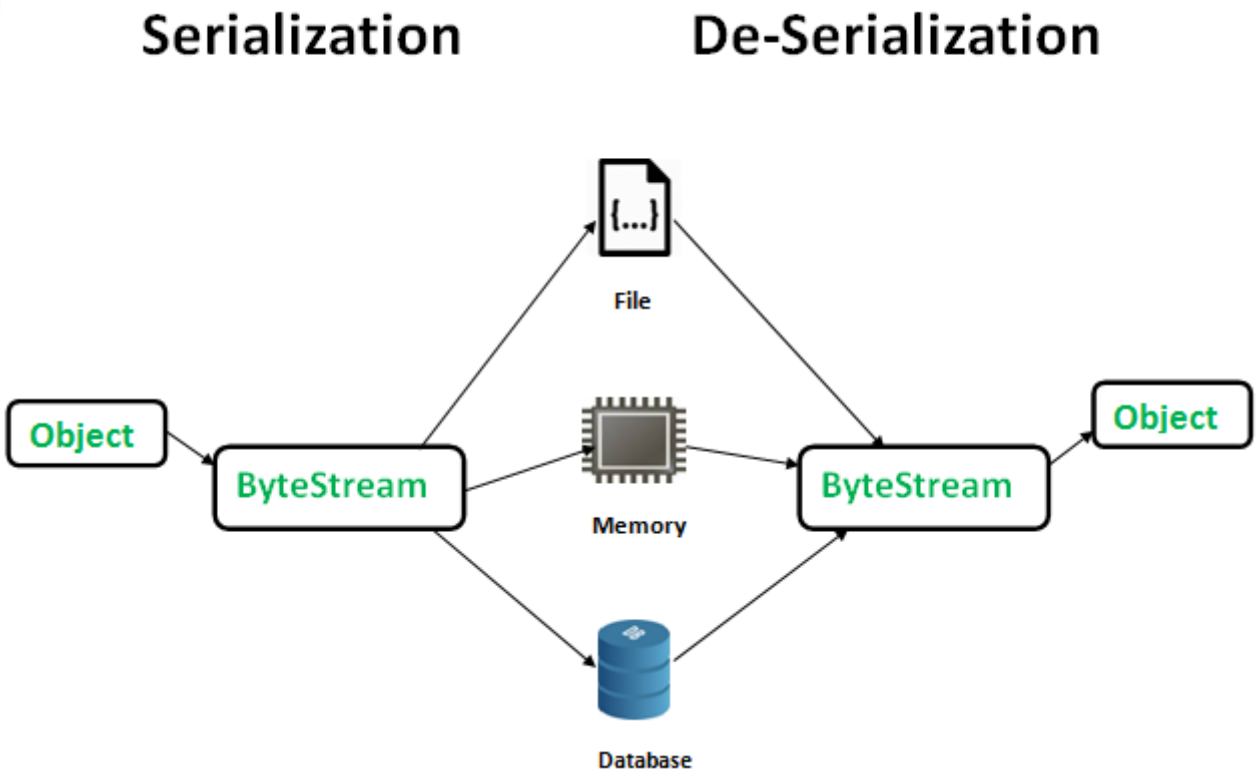
Network Programming for Windows

Serialization

jintaeks@dongseo.ac.kr
Division of Digital Contents, DSU
May 2018

Serialization

- ✓ **serialization** is the process of translating data structures or object state into a format that can be stored or transmitted (for example, across a network connection link) and reconstructed later (possibly in a different computer environment).



- ✓ serialization == marshalling
- ✓ deserialization == unmarshalling

Uses

- ✓ A method of transferring data through the wires (messaging).
- ✓ A method of storing data (in databases, on hard disk drives).
- ✓ A method of remote procedure calls, e.g., as in SOAP.
- ✓ A method for distributing objects, especially in component-based software engineering such as COM, CORBA, etc.
- ✓ A method for detecting changes in time-varying data.

Pointer swizzling

- ✓ **pointer swizzling** is the conversion of references based on name or position to direct pointer references.
  - It is typically performed during the deserialization (loading) of a relocatable object from disk.
- ✓ The reverse operation, replacing pointers with position-independent symbols or positions, is sometimes referred to as **unswizzling**, and is performed during serialization (saving).

```
struct node {
    int data;
    struct node *next;
};

struct node_saved {
    int data;
    int id_number;
   int id_number_of_next_node;
};
```

boost serialization

```
#include <boost/archive/text_oarchive.hpp>
#include <boost/archive/text_iarchive.hpp>
#include <iostream>
#include <sstream>

#pragma pack(push,1)
struct KData
{
    char m_cData;
    float m_fData;

    template <typename Archive>
    void serialize(Archive &ar, const unsigned int version)
    {
        ar & m_cData;
        ar & m_fData;
    }
};
#pragma pack(pop)
```

```
//template <typename Archive>
//void serialize(Archive &ar, KData& a, const unsigned int version)
//{
//    ar & a.m_cData;
//    ar & a.m_fData;
//}

int main()
{
    KData d;
    d.m_cData = 1;
    d.m_fData = 2.3f;

    std::stringstream ss;
    boost::archive::text_oarchive oa( ss );
    oa << d;
    std::cout << ss.str() << std::endl;

    boost::archive::text_iarchive ia( ss );
    KData d2;
    ia >> d2;
}
```

Packet serialization

```
#include <boost/serialization/vector.hpp>
#include <boost/serialization/string.hpp>
#include <boost/serialization/version.hpp>
#include <boost/archive/text_oarchive.hpp>
#include <boost/archive/text_iarchive.hpp>
#include <boost/shared_ptr.hpp>
#include <iostream>
#include <sstream>


enum EPacketType
{
    ECLGS_VERIFY_ACCOUNT_REQ,
    ECLGS_LOGIN,
};
```

```
struct KPacketVerifyAccount
{
    std::string         m_login;
    int                 m_id;
};


template <typename Archive>
void serialize( Archive& ar, KPacketVerifyAccount& a, const unsigned int version )
{
    ar & a.m_login;
    ar & a.m_id;
}
```

```
struct KPacketLogin
{
    std::string         m_login;
    std::string         m_password;
    int                 m_id;
    int                 m_age;
};

template <typename Archive>
void serialize( Archive& ar, KPacketLogin& a, const unsigned int version )
{
    ar & a.m_login;
    ar & a.m_password;
    ar & a.m_id;
    ar & a.m_age;
}
```

```
#ifndef IN
#define IN
#endif

#ifndef OUT
#define OUT
#endif


/// @see     https://stackoverflow.com/questions/8815164/c-wrapping-vectorchar-with-istream
template<typename CharT, typename TraitsT = std::char_traits<CharT> >
class vectorwrapbuf : public std::basic_streambuf<CharT, TraitsT>
{
public:
    vectorwrapbuf( IN std::vector<CharT>& vec )
    {
        setg( vec.data(), vec.data(), vec.data() + vec.size() );
    }
};//class vectorwrapbuf
```

```
class KPacket;
typedef boost::shared_ptr<KPacket>   KPacketPtr;
class KPacket
{
public:
    template <class T>
    void SetData( unsigned int nSenderUID, unsigned short usPacketId, const T& data );

    unsigned int         m_nSenderUid;
    unsigned short       m_usPacketId;
    std::vector<char>    m_buffer;
};//class KPacket

template <typename Archive>
void serialize( Archive& ar, KPacket& a, const unsigned int version )
{
    ar & a.m_nSenderUid;
    ar & a.m_usPacketId;
    ar & a.m_buffer;
}//serialize()
```

```
template <class T>
void KPacket::SetData( unsigned int nSenderUID, unsigned short usPacketId, const T& data_ )
{
    m_nSenderUid = nSenderUID;
    m_usPacketId = usPacketId;

    std::stringstream    ss;
    boost::archive::text_oarchive oa{ ss };
    oa << data_;

    std::string& str = ss.str();
    m_buffer.reserve( str.size() );
    m_buffer.assign( str.begin(), str.end() );
}//KPacket::SetData()
```

```
template <typename T>
void BufferToPacket( IN std::vector<char>& buffer, OUT T& data )
{
    // alternative (slow) implementation. jintaeks on 2017-08-24_20-08
    //std::stringstream ss;
    //std::copy(buffer.begin(), buffer.end(), std::ostream_iterator<char>(ss));
    //boost::archive::text_iarchive ia{ ss };
    //ia >> data;

    vectorwrapbuf<char> databuf( buffer );
    std::istream is( &databuf );
    boost::archive::text_iarchive ia{ is };
    ia >> data;
}//BufferToPacket()


template <typename T>
void BufferToPacket( IN std::stringstream& ss_, OUT T& packet_ )
{
    boost::archive::text_iarchive ia{ ss_ };
    ia >> packet_;
}//BufferToPacket()
```

```
template<typename T>
void PacketToBuffer( IN T& packet_, OUT std::vector<char>& buffer_ )
{
    std::stringstream    ss;
    boost::archive::text_oarchive oa{ ss };
    oa << packet_;

    // set [out] parameter
    std::string& str = ss.str();
    buffer_.reserve( str.size() );
    buffer_.assign( str.begin(), str.end() );
}//PacketToBuffer()


template<typename T>
void PacketToBuffer( IN T& packet_, OUT std::stringstream& ss_ )
{
    boost::archive::text_oarchive oa{ ss_ };
    oa << packet_;
}//PacketToBuffer()
```

```
void main()
{
    KPacket      packets[2];

    KPacketVerifyAccount     verifyAccount;
    {
        verifyAccount.m_login = "jintaeks";
        verifyAccount.m_id = 48;
        packets[0].SetData(0, ECLGS_VERIFY_ACCOUNT_REQ, verifyAccount);
    }


    KPacketLogin         login;
    {
        login.m_login = "jintaeks₩0hello";
        login.m_password = "hello world";
        login.m_id = 99;
        login.m_age = 48;
        packets[1].SetData(0, ECLGS_LOGIN, login);
    }
```

```
// write to archive
    //
    std::stringstream ss;
    PacketToBuffer(IN packets[1], OUT ss);

    // read from archive
    //
    KPacket      packetNew;
    BufferToPacket(IN ss, OUT packetNew);

    if (packetNew.m_usPacketId == ECLGS_VERIFY_ACCOUNT_REQ)
    {
        KPacketVerifyAccount     verifyNew;
        BufferToPacket( IN packetNew.m_buffer, OUT verifyNew );
    }
    else if (packetNew.m_usPacketId == ECLGS_LOGIN)
    {
        KPacketLogin     loginNew;
        BufferToPacket( IN packetNew.m_buffer, OUT loginNew );
    }//if.. else if..
}//main()
```

MY **BRIGHT** FUTURE

DSU Dongseo University 동서대학교

DSU Dongseo Univ 동서대학교