Global Journal of Computer Science and Technology
Software & Data Engineering
Volume 13 Issue 6 Version 1.0 Year 2013
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Object Serialization Formats and Techniques a Review

By Surbhi & Rama Chawla
Kurukshetra University DIET, India

Abstract - Serialization is a process of converting an object into a stream of data so that it can be easily transmittable over the network or can be continued in a persistent storage location. This storage location can be a physical file, database or Network Stream. This paper concludes some the work that is going on in the field of Object Serialization.

This paper presents Object Serialization Techniques that can be useful for various purposes, including object serialization Minimization which can be used to decrease the size of Serialized data.

Keywords : object serialization, compression techniques, object oriented design, performance analytics, soap.

GJCST-C Classification : D.1.5

Strictly as per the compliance and regulations of:

Object Serialization Formats and Techniques a Review

Surbhi [α] & Rama Chawla [σ]

Abstract - Serialization is a process of converting an object into a stream of data so that it can be easily transmittable over the network or can be continued in a persistent storage location. This storage location can be a physical file, database or Network Stream. This paper concludes some the work that is going on in the field of Object Serialization.

This paper presents Object Serialization Techniques that can be useful for various purposes, including object serialization Minimization which can be used to decrease the size of Serialized data.

Keywords : object serialization, compression techniques, object oriented design, performance analytics, soap.

I. Introduction

Serialization is the process of converting complex objects into stream of bytes for storage. Deserialization is its reverse process that is unpacking stream of bytes to their original form. It is also known as Pickling, the process of creating a serialized representation of object.

The following steps are necessary to do to create a serializable class:

1. Create a custom class with assigned properties.
2. Define the serialization functions.
3. Create a Controller class and instantiate our Custom class.
4. Serialize the object to a named file.
5. De-serialize the values by reading it from the file.

Object serialization has been investigated for many years in the context of many different distributed systems.

II. Most Popular Serialization Formats

There are various data serialization formats available for developers according to choose form, There are also various ways to convert complex objects to sequences of bits. It does not include markup languages used exclusively as document file formats.

- Binary Format Serialization
- XML Format Serialization
- XML-RPC Serialization[1]
- JSON Serialization[2]
- YAML[C] Serialization

The following are the basic advantages of serialization: First is to facilitate the transportation of an object through a network and secondly create a clone of an object that can be restored later on.

III. Related Work

In the paper "**Object Serialization and De-serialization Using XML**[3]" Inter operability of potentially heterogeneous databases has been an ongoing research issue for a number of years in the database community.

With the trend towards globalization of data location and data access and the consequent requirement for the coexistence of new data stores with legacy systems, the cooperation and data interchange between data repositories has become increasingly important. The emergence of the extensible Markup Language (XML) as a database independent representation for data offers a suitable mechanism for transporting data between repositories.

This paper describes a research activity within a group at CERN (called CMS) towards identifying and implementing database serialization and deserialization methods that can be used to replicate or migrate objects across the network between CERN and worldwide centers using XML to serialize the contents of multiple objects resident in object oriented databases.

The paper "**Generic Pickling and Minimization** [4]" presents generic pickling and minimization mechanisms that are provided as services similar to garbage collection. Pickling is used to externalize and internalize data. Minimization means to maximize the sharing in arbitrary data structures. The paper introduces the notion of an abstract store as a formal basis for the algorithms, and analyzes design decisions for the implementation aspects of pickling and minimization. The mechanisms presented here are fully implemented in the Alice programming system.

We presented a generic pickling and minimization mechanism. We showed how Alice, as a conservative extension of Standard ML, uses pickling in a type safe way for its component system. To build a formal base for the algorithms, we introduced abstract stores as a universal memory model. Un-pickling and pickling are based on this model, allowing us to analyze

Author α σ : Department of Computer Science & Engineering, Kurukshetra University DIET, Karnal, Haryana, India.
E-mails : er.surbhi88@gmail.com, ramachawla27@gmail.com

and evaluate our design decisions such as bottom up versus top down un-pickling and right to left versus left to right traversal. Minimization can be used to decrease the size of pickled data. However, the general mechanism presented here seems suitable for other applications such as efficient representation of runtime types. Finally, we extended the system with support for concurrency as present in Alice. The authors analyzed how pickler and minimizer must behave in such a concurrent setting.

In the paper "**Why Object Serialization is Inappropriate for Providing Persistence in Java**"[5] the author paper describes why Object Serialization is not appropriate for providing persistence in Java. With numerous code examples, Object Serialization is shown to be easy to work with initially which seduces the developer into relying on it for persistence within more complex applications.

The advanced use of object serialization requires significant work from the programmer, something that is not apparent at first. The use of object serialization together with static and transient fields and within multithreaded programs is discussed together with the "big inhale problem": the need to read in the entire object graph before processing over it can commence.

This paper has shown, with numerous supporting examples, that using Java's object serialization mechanism to provide object persistence is inappropriate. The system appears simple on the surface but there are many implications from relying on it as a persistence technology. The programmer must state the types that are candidates for persistence at compile time, whereas making this decision at runtime, on a per object basis, is more appropriate.

The serialization mechanism suffers from the **big inhale** problem where the whole graph must be read before it can be used; loading objects on demand is more efficient, reducing delay in starting an application. The serialization mechanism creates copies of objects that it writes and reads. This can break some code that makes assumptions about the hash code of an object.

μs per object	32 int		4 int, 2 null		tree(15)	
	w	r	w	r	w	r
JDK serialization	346	1410	169	596	1192	1889
UKA-serialization	35	39	19	28	201	354
improvement %	**90**	**97**	**89**	**95**	**83**	**81**
explicit marshaling	228	920	81	308	396	647
slim type encoding	19	187	16	159	72	213
internal buffering	0	159	6	19	0	330
buffer accessibility	48	65	30	49	502	291
two types of reset	16	40	17	33	21	54

Figure 1 : Object serialization for various type of objects with encodings[5]

The complexity of using object serialization within 1a distributed environment, when evolving classes and when using specialized class loaders is also discussed. The paper compares the performance of serializing and de-serializing a byte array and binary tree of the same data size to and from an NFS mounted disk and two kinds of local disk. Alternative solutions to object persistence in Java are presented at the end of the paper.

Using Experiments carried out by author to draws four conclusions:

1. The absolute amount of time to read and write a store is large;
2. Reading a store is much slower than writing a store; and if an application is likely to exhibit more reading than writing,
3. An NFS mounted disk should be used;
4. The use of JIT technology significantly increases the speed of using Java object serialization.

In the "**Object Serialization in the .NET Framework**"[6], the author describes using Serialization in .Net framework. He describes the two most important reasons are to persist the state of an object to a storage medium so an exact copy can be recreated at a later stage, and to send the object by value from one application domain to another.

It is also used by remoting to pass objects by value from one application domain to another. This paper provides an overview of the serialization used in the Microsoft .NET Framework.

The author gives Serialization Guidelines, one should consider serialization when designing new classes since a class cannot be made serializable after it has been compiled. Some questions to ask are: Do one have to send this class across application domains? Will this class ever be used with remoting? What will users do with this class? Maybe they derive a new class that needs to be serialized. When in doubt, mark the class as serializable. It is probably better to mark all classes as serializable unless:

- They will never cross an application domain. If serialization is not required and the class needs to cross an application domain, derive the class from Marshal by Ref. Object.
- The class stores special pointers that are only applicable to the current instance of the class. If a class contains unmanaged memory or file handles, for example, ensure these fields are marked as Non-Serialized or don't serialize the class at all.

"**Comparison between JSON and YAML for data serialization**"[7], this report determines and discusses the primary differences between two different serialization formats, namely YAML and JSON. A general introduction to the concepts of serialization and parsing is provided first, which also explains how they can be used to transfer and store data. This is followed by an analysis of the YAML and JSON formats, where

functionality, primary use cases, and syntax is described. In addition to this the percieved performance of implementations for both formats will also be investigated by conducting a number of tests.

Using the combined background information and results from the tests, conclusions regarding the main differences between the two are then determined and discussed.

As has been concluded, it is clearly very easy to read thanks to the required usage of whitespace and the ability to skip surrounding quotes for strings. YAML also has the advantage of allowing comments in the document. Users can easily read and manipulate the output, which is one of the reasons as to why it's often used for configuration files.

This enables the straightforward definition of strongly-typed objects that match serialized structures, for example existing XML formats. Inheritable translation scopes group sets of object serialization binding definitions, and enable inheritance. The present system supports (compressed) XML for serialization, while future work will develop alternate translation schemes, such as type-length-value and JSON.

Execution times in seconds for Seriali-zation

Method	Simple	Complex
JSON.generate	0.1550s	0.5830s
JSON.pretty_generate	0.1470s	0.6060s
YAML.dump	2.4531s	3.4732s

Execution times in seconds for De-Serialization

Method	Simple	Complex
JSON.parse	0.0440s	0.0790s
YAML.load	0.2750s	0.3360s

Table 2 : JSON VS. YAML Serialization performance [7]

The execution times measured for the serialization/deserialization process shows their results, similar to the serialization process, which can be seen in table 2. Both implementations are much faster at generating data structures from a serialized string than doing the opposite. YAML is also slower.

IV. Conclusion

The primary design goals for Serialization, to provide a simple and effective data exchange, but also being easy to generate and load. It is widely used and is used natively available in the most common modern programming. Object Serialization as presented here is especially well suited for functional programming languages, where the closure semantics and the ability to serialize code is essential. Also a minimization technique helps reduce Serialization sizes considerably.

V. Future Scope

To implement means by which Serialization and Deserialization of Objects can be done using modern formats XML and JSON after adding Compression or Encryption or possibly both to the Object Streams.

In future I also want to see how the Performance of Object Serialization is affected in a Normal CLR Binary VS. Native JIT compiled Binary. The perceived performance of XML or JSON can be determined from a custom benchmarking. A complex data set will also be used to test performance of the implementations for documents with deeper hierarchies.

Another direction for future work may be to modify the semantics of the serialization algorithm to improve performance.

References Références Referencias

1. Mark Allman. 2003. An evaluation of XML-RPC. SIGMETRICS Perform. Eval. Rev. 30, 4 (March 2003), 2-11.
2. Audie Sumaray and S. Kami Makki. 2012. A comparison of data serialization formats for optimal efficiency on a mobile platform. In Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication. ACM, New York, NY, USA, Article 48.
3. N. Bhatti & W. Hassan, et al, "Object Serialization and Deserialization Using XML", Tata McGraw-Hill, ADVANCES IN DATA MANAGEMENT, Vol 1, 2000.
4. G. Imre, et al, "A Novel Cost Model of XML Serialization", Science Direct, Electronic Notes in Theoretical Computer Science, vol. 261, Department of Automation and Applied Informatics, Budapest University of Technology and Economics, Budapest, Hungary, 2010.
5. Guido Tack, Leif Kornstaedt, et al, " Generic Pickling and Minimization", Science Direct, Electronic Notes in Theoretical Computer Science, Programming Systems Lab Saarland University, Saarbr¨ucken, Germany, 2006.
6. Huw Evans, "Why Object Serialization is Inappropriate for Providing Persistence in Java", Department of Computing Science, The University of Glasgow, Glasgow, G12 8RZ, UK.
7. MALIN ERIKSSON, VICTOR HALLBERG, "Comparison between JSON and YAML for data serialization", the School of Computer Science and Engineering Royal Institute of Technology, 2011.
8. Piet Obermeyer and Jonathan Hawkins, "Object Serialization in the .NET Framework", Microsoft Corporation, 2001.
9. Lukasz Opyrchal and Atul Prakash, "Efficient Object Serialization in Java", Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122, USA.