СЕРИАЛИЗАЦИЯ СТРУКТУР ДАННЫХ ДЛЯ ХРАНЕНИЯ И ПЕРЕДАЧИ В ИНФОРМАЦИОННЫХ СИСТЕМАХ. МЕТОДЫ И СРЕДСТВА

Г. В. Погодин, Д. М. Фиго, Э. Н. Васильев

ФГУП «РФЯЦ-ВНИИЭФ», г. Саров Нижегородской обл.

Введение

В настоящее время при создании информационной системы или программного продукта логичным, а часто и обязательным является рассмотрение такой важной характеристики, как интероперабельность, т. е. способность к взаимодействию и функционированию с другими системами и продуктами без каких-либо ограничений реализации.

Для обеспечения интероперабельности необходимы эффективные средства обмена информацией между существенно отличающимися друг от друга программными системами. Многие разработчики решают вопрос такого информационного обмена, реализуя функцию создания файла выгрузки в XML-формате (в естественном или сжатом архиватором виде). Однако помимо XML-формата в настоящее время существует множество методов и средств сериализации структур данных, среди которых наиболее распространёнными являются форматы (описания будут приведены в следующем разделе):

- JSON;
- YAML;
- XML;
- INI;
- CSV;
- HDF;
- netCDF;
- GRIB.

Этот внушительный и не исчерпывающий список уже сам по себе порождает вопросы о причинах многообразия и о критериях выбора наиболее подходящего варианта. Решающим фактором для настоящего исследования стала статья британского программиста и аналитика Mehdi Gholam: «Smallest, fastest polymorphic JSON serializer» [1]. Выдержки из данной статьи представлены на рис. 1 и рис. 2, и показывают результаты тестирования шести программных модулей, соревновавшихся по скорости сериализации и десериализации тестовых структур данных, представленной в бинарном и JSON-формате.

Из приведённых выше результатов видно, что скорость десериализации для всех методов и средств, как правило, ниже скорости сериализации. В зависимости от выбранного средства, длительности этих процессов могут различаться на порядок в пределах одной тестовой структуры, что говорит о важности выбора соответствующего метода и средства, когда в программном продукте или информационной системе часто выполняются эти операции.

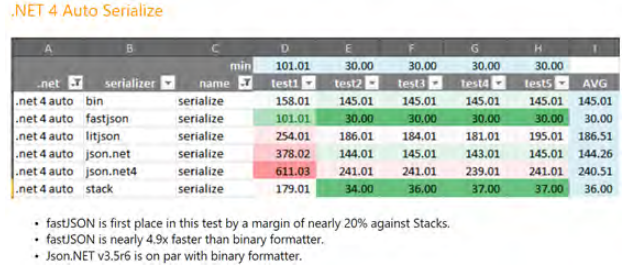

Рис. 1. Длительности выполнения сериализации для шести .Net-программ

Рис. 2. Длительности выполнения десериализации для шести .Net-программ

Но самым неожиданным в данной статье стало заявление о крайней неэффективности применения формата XML, для которого средства сериализации работают примерно в 50 раз дольше, чем самый медленное из участвовавших в сравнении. Цитата: “Ifyouareusing XML, then *don't*. It's too slow and bloated, it does deserve an honorable mention as being the first thing everyone uses, but seriously don't. It's about **50 times slower** than the slowest JSON in this list. The upside is that you can convert to and from JSON easily”.

При этом если сделать грубую оценку количества текстовых файлов с сериализованными данными, просто исходя из расширений в их названиях, то очевидно лидерство файлов, принадлежащих экосистеме XML, что видно из табл. 1.

Таблица 1

Распределение текстовых файлов с сериализованными данными

Тип файла	Количество	
*.XML	4643	5125
*.XSL	256	
*.XSD	186	
*.XSLT	40	
*.INI	3228	
*.CSV	61	
*.JSON	22	

Из всего вышесказанного не обязательно следует, что разработчикам нужно срочно переходить на более молодой и модный формат, но становится окончательно ясно что формат XML, как и его конкуренты, не является однозначно эффективным для всех технических условий применения, и его выбор для каждого конкретного случая подразумевает трезвую оценку преимуществ перед альтернативными способами сериализации и десериализации. С целью систематизировать знания для такой оценки в настоящей работе выполнен обзор методов и средств сериализации структур данных, проведено сравнение приведённых методов и определены границы их применимости.

Обзор методовсериализации и десериализации

JSON (англ. Java Script Object Notation) – текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается человеком. За счёт своей лаконичности по сравнению с XML, формат JSON больше подходит для сериализации сложных структур [2].

YAML (англ. Yet Another Markup Language) – «дружественный» формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода типичных структур данных многих языков программирования. Синтаксис YAML минималистичен, особенно по сравнению с XML синтаксисом. YAML в основном используется как формат для файлов конфигурации [2].

INI (англ. Initialization file) – это формат файлов конфигурации, который содержит данные настроек Подробной официальной спецификации формата не существует. Использование секций для разделения параметров способствует систематизации данных, однако для хранения настроек большого объёма с более сложной иерархической структурой лучше подходит формат XML или JSON [2].

CSV (англ. Comma-Separated Values – значения, разделённые запятыми) – это текстовый формат, предназначенный для представления табличных данных. На сегодняшний день под CSV, как правило, понимают набор значений, разделенных какими угодно разделителями. Импорт и экспорт CSV-файлов возможен во многих инженерных пакетах [2].

HDF (Hierarchical Data Format) –формат файлов, разработанный для хранения большого количества цифровой информации. Содержимое файлов HDF организовано подобно иерархической файловой системе. Метаданные хранятся в виде набора именованных атрибутов объектов [2].

GRIB (GRIdded Binary) – математический формат сжатых данных, обычно используемый в метеорологии для хранения исторических и прогнозируемых данных о погоде. GRIB заменил ADF (аэронавигационный формат данных) [2].

NetCDF (Network Common Data Form) – машинно-независимый двоичный формат файлов, являющийся открытым стандартом для обмена научными данными [3]. В основном используется в климатологии, например, при прогнозировании погоды, изучении изменения климата и геоинформационных системах. В частности, авторитетная в области океанографии организация National Oceanic and Atmospheric Administration (NOAA, США) использует этот формат в своей базе данных, доступной любому пользователю глобальной сети Интернет.

XML (англ. eXtensible Markup Language) – расширяемый язык разметки. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка [2].

Важным свойством XML, влияющим на интероперабельность, можно считать давнюю поддержку электронными таблицами Excel, позволяющими человеку работать не с разметкой, а обрабатывать инкапсулированные в ней данные спомощью штатных табличных функций общесистемного программного обеспечения. На следующей схеме (рис. 3) показано, как взаимодействуют различные файлы и операции при использовании XML в Excel. Экспорт обработанных данных из сопоставленных ячеек в XML-файл данных.

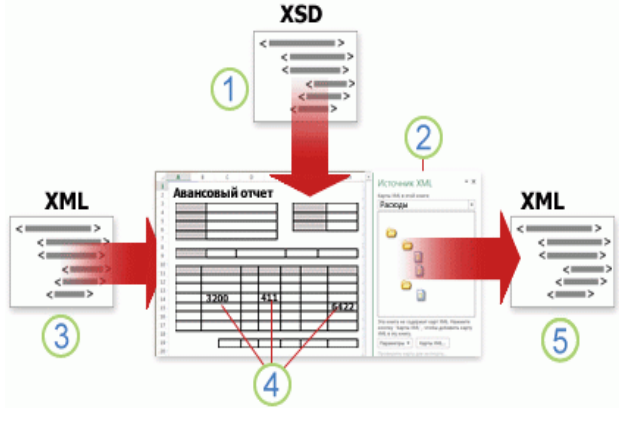

Рис. 3. Обработка XML с помощью электронных таблиц

Этот процесс можно представить в виде пяти этапов:

1) добавление в книгу файла схемы XML (XSD);

2) сопоставление элементов схемы XML с отдельными ячейками или XML-таблицами;

3) импорт XML-файла данных и привязка XML-элементов к сопоставленным ячейкам;

4) ввод данных, перемещение сопоставленных ячеек и использование функциональных возможностей Excel при сохранении структуры и определений XML.

Сочетание простого формального синтаксиса, удобства для человека, расширяемости, а также базирование на кодировках Юникод для представления содержания документов привело к широкому использованию как собственно XML, так и множества производных специализированных языков (расширений) на базе XML в самых разнообразных программных средствах. Такими расширениями, очень важными для поддержки жизненного цикла изделий, являются PLM XML [4] и STEP XML [5].

PLM XML представляет собой протокол совместной работы или обмена информацией между различными источниками данных (PLM-системами, хранилищами данных и файлами деталей) и разнообразным набором приложений (визуализация / макет, CAD, производство, PDM и т. д.).

Схематично состав данных, содержащихся в файлах PLM XML, представлен на рис. 4.

Рис. 4. Состав набора данных объекта PLMXML

Схема данных одного из исследованных файлов формате PLM XML представлена на рис. 5.

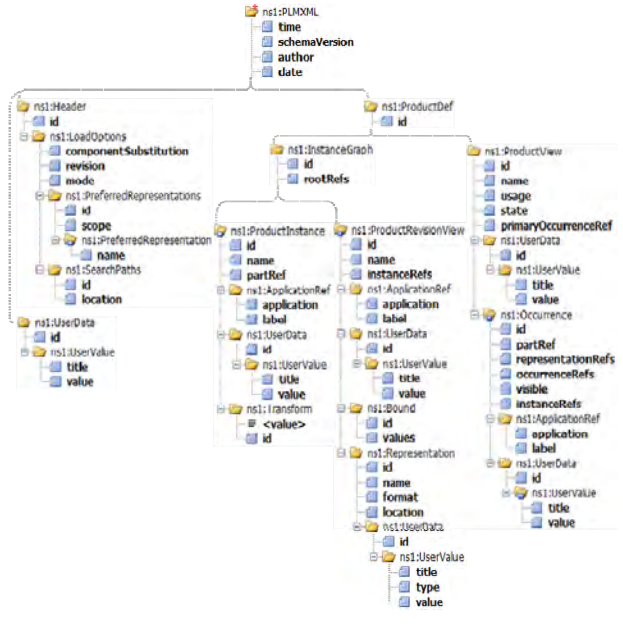

Рис. 5. Схема данных в формате PLM XML

Данный файл занимает 14 КБ дискового пространства и несёт в себе информацию о сборочной единице, состоящей из 9 деталей, в нём данные о взаимном расположении ссылки на их упрощённую геометрию. Сама же информация об упрощённой геометрии хранится в бинарных файлах JT-описания, в совокупности занимающих 364 КБ.

Будучи считанным вместе с дочерними JT-описаниями из файлов на диске, десериализованных CAD-системой, результат представляет собой 3D-модель, которая в пользовательском графическом окне просмотра будет выглядеть, как показано на рис. 6.

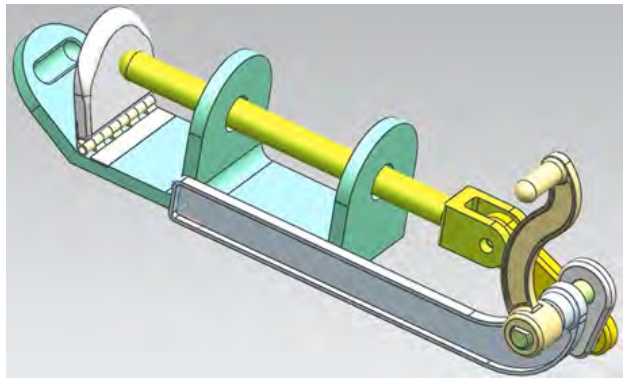

Рис. 6. 3D-модель в CAD-системе после обработки PLMXMLфайла

Ещё одним из важных расширений языка XML, для ВНИИЭФ и всех предприятий-пользователей системы ЛОЦМАН: PLMявляется язык описания конфигураций баз данных ЛОЦМАН: PLM, применяемый в файлах с расширением cpd_cfg.

Сравнение методов сериализации JSON и XML, YAML

В качестве основных технологий для сравнения были выбраны JSON, XML и YAML. Данный выбор обусловлен распространенностью и динамикой развития форматов обмена данными.

Для сравнения различных методов сериализации-десериализации JSON и XML, YAML были выбраны следующие критерии оценки:

1. Удобочитаемость кода.
2. Скорость сериализации-десериализации.
3. Объем файла в различных форматах.

Удобочитаемость кода

Критерий инспектируемости (удобочитаемости) является достаточно субъективным, но существенным при ручной проверке или правке содержимого файла, содержащего конфиденциальную информацию.

В табл. 2 приведены примеры одних и тех же данных, представленных в различных форматах.

Таблица 2

Инспектируемость кода JSON и XML, YAML

JSON	XML	YAML

"embedded2":
{
"strel1": "el1",
"strel2": "el2",
"strel4": "el4",
"strel5": "el5",
"strel6": "el6",
"strel7": "el7",
"intel1": 1,
"intel2": 2,
"intel3": 3,
"list1": [1, 2, 3, 4, 5],
"list2":["1","2", "3","4","5","6"],
"map1": {"3":3,"2": 2,"1": 1},
"map2": {"1": "1", "2": "2", "3": "3" }
}
	<root>
	<embedded2>
	<intel1>1</intel1>
	<intel2>2</intel2>
	<intel3>3</intel3>
	<list1>1</list1>
	<list1>2</list1>
	<list1>3</list1>
	<list1>4</list1>
	<list1>5</list1>
	<list2>1</list2>
	<list2>2</list2>
	<list2>3</list2>
	<list2>4</list2>
	<list2>5</list2>
	<list2>6</list2>
	<map1>
	<1>1</1>
	<2>2</2>
	<3>3</3>
	</map1>
	<map2>
	<1>1</1>
	<2>2</2>
	<3>3</3>
	</map2>
	<strel1>el1</strel1>
	<strel2>el2</strel2>
	<strel4>el4</strel4>
	<strel5>el5</strel5>
	<strel6>el6</strel6>
	<strel7>el7</strel7>
	</embedded2>
	</root>
		embedded2:
		intel1: 1
		intel2: 2
		intel3: 3
		list1:
		- 1
		- 2
		- 3
		- 4
		- 5
		list2:
		- '1'
		- '2'
		- '3'
		- '4'
		- '5'
		- '6'
		map1: '3': 3 '2': 2 '1': 1
		map2: 1: '1' 2: '2' 3: '3'
		strel1: el1
		strel2: el2
		strel4: el4
		strel5: el5
		strel6: el6
		strel7: el7

Скорость сериализации-десериализации и объем файла в разных форматах

Для определения скорости сериализации-десериализации был взят файл конфигурации базы данных PDM системы в формате XML, его схема представлена на рис. 7.

Для последующей работы тестовый файл был переведен в форматы JSON и YAML с помощью общедоступной утилиты XMLSpy. После этого объекты из полученных файлов поочерёдно сериализовывались и десериализовывались различными методами на одном и том же персональном компьютере с параметрами:

– процессор: Intel i7 с тактовой частотой 3,4 ГГц;

– оперативная память: 8 ГБ.

В качестве парсера JSON использовался Fast-JSON, для парсинга XML применялся Microsoft XML Parser [6]. Результаты измерений приведены в табл. 3.

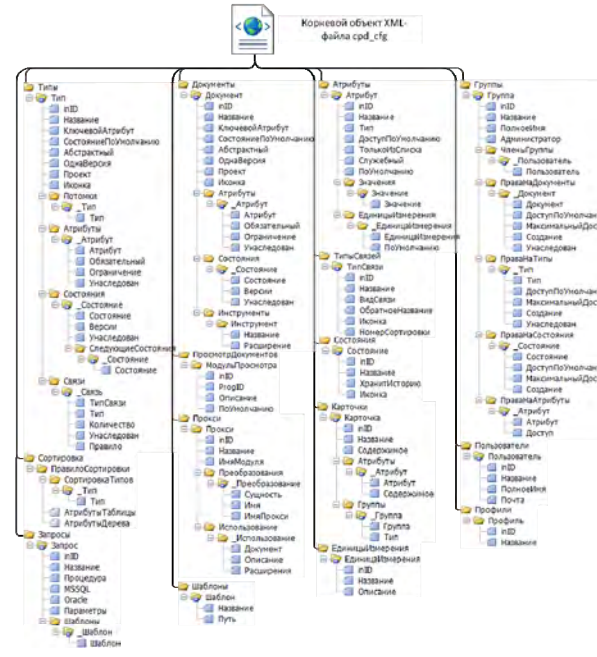

Рис. 7. Схема файла XML-конфигурации базы данных PDM системы

Таблица 3

Сравнение форматов сериализации-десериализации

Метод и средства сериализации-десериализации	Время операции сериализации, миллисекунды	Время операции десериализации, миллисекунды	Объем файла, байт
JSON (FastJSON)	279	644	556
XML (Microsoft XML Parser)	773	461	886
YAML	689	572	443

Таблица 4

Плюсы и минусы различных реализаций JSON

Реализация JSON	Плюс	Минус
JSON.Net	– Возможность может обрабатывать наборы данных в качестве выходных данных	– Большой размер файла; – Медленный по сравнению с остальными реализациями JSON;
LitJSON	– Малый размер dll; – Относительно быстро	– Не обрабатывает наборы данных в исходном коде – Для полноценной работы необходимо наличие установленного .NET 3.5
Microsoft Json Serializer v1.7	– Является встроенным во многие среды разработки; – Может сериализовать основные полиморфные объекты	– Не может десериализовывать полиморфные объекты; – Некорректная работа с наборами данных; – Не работает с такими типами как словари, хеш-таблицы.
FastJSON	– Малый размер при компиляции; – Быстрый; – Обрабатывает полиморфные наборы объектов	– Для полноценной работы необходимо наличие установленного Silverlight 4.0

Из табл. 3 видно, что JSON сериализуется существенно быстрее XML и YAML, но пятидесятикратное преимущество FastJSON не подтвердилось, а на десериализации предложенной структуры данных он даже показал худший результат.

Есть предположение, что результаты обусловлены индивидуальными особенностями тестовой структуры и тем, что файл в формате JSON считывается посимвольно и сразу сохраняется в объект, таким образом, объект формируется за один проход по файлу. Этап обработки XML и YAML данных делится на три стадии. Сначала строится дерево объектов, далее идет преобразование дерева объектов, после чего уже конвертируется в нужные структуры данных. Разница времен десериализации обусловлена эффективностью алгоритма сжатия данных. При обработке небольших объемов данных, JSON будет более компактным, однако при работе с более сложными структурами эффективность сжатия XML будет выше, чем у JSON или YAML.

Плюсы и минусы различных реализаций JSON

В связи с распространённостью нотации JSON существует большое количество различных реализаций данного формата. Для сравнения, некоторые плюсы и минусы различных реализаций приведены в табл. 4.

Выводы

Рассмотренная проблема может оказаться ближе и глубже, чем кажется на первый взгляд: данные технологии используются в программном обеспечении повсеместно, и выбор метода сериализации-десериализации может существенно влиять на время отклика информационной системы на запрос пользователя, и этот выбор зависит от конкретной задачи.

XML эффективен там, где доля разнотипных символьных данных в общем потоке велика, доля разметки мала, а возможности трансляции и синтаксической проверки размеченного текста востребова-

ны. В остальных случаях программная обработка XML может оказаться неоправданно затратной, и в качестве нотации для хранения и передачи информации, полезно рассмотреть средства, изначально ориентированные на данные, такие как INI, YAML, JSON.

В настоящей работе, собрана лишь начальная информация для комплексного и доказательного рассмотрения вопроса эффективности различных методов и средств сериализации-десериализации в конкретных ситуациях применения. Можно заключить, что подняты проблемы и предложены пути повышения производительности программ на участках сериализации данных. Полученные результаты помогут оптимизировать механизмы хранения и передачи сложных структур данных при разработке различных компонентов и систем внедряемых и разрабатываемых в настоящий момент специалистами ВНИИЭФ.

Литература

1. https://www.codeproject.com/Articles/159450/fastJSON
2. http://ru.wikipedia.org
3. The NetCDF Users Guide. Data Model, Programming Interfaces, and Format for Self-Describing, Portable Data.NetCDF version 4.0. 2008.
4. https://www.plm.automation.siemens.com/ru_ru/products/open/plmxml.
5. ISO 10303-28
6. https://msdn.microsoft.com/ru-ru/library/bb412179(v = vs.110).aspx.