Оригинальная статья / Original article

https://doi.org/10.21869/2223-1536-2023-13-3-31-51

УДК 004.89:004.93

Интеллектуальная система обеспечения миграции посредством динамической десериализации данных

Р. А. Томакова[1] ✉, Д. В. Иванов[1], Н. А. Корсунский[1]

[1] Юго-Западный государственный университет
ул. 50 лет Октября, д. 94, г. Курск 305040, Российская Федерация

✉ e-mail: rtomakova@mail.ru

Резюме

Цель исследования *заключается в разработке моделей и алгоритмов системы асинхронной десериализации данных, имеющих различные форматы и структуры, для повышения эффективности определения моделей данных за счет генерации строго типизированных объектов.*

Методы. *Способ десериализации моделей из данных предполагает построчную декомпозицию строки JSON-файла с определением типов «ключ»-«значение» и с их соотнесением с моделью данных: символом, строкой, числом, булевым значением. После этого веб-контроллер проводит асинхронную генерацию класса и его объектов. Для классификации значений строк применяются классификаторы значений сериализованной строки. Для асинхронной генерации объектов используется система «контрактов» моделей и алгоритмы исполнения и конвертации данных моделей.*

Результаты. *Разработан десериализатор данных, который состоит из системы четырех контроллеров анализа моделей и алгоритма генерации значений. Простая модель десериализации одной модели позволяет сопоставить модель с заголовками таблицы реляционной базы данных для обеспечения миграции модели между системами. Генерируемые объекты представляются статическими типами данных, что обеспечивает их запись в любую систему СУБД встроенными средствами. Сложная модель представляет блок значений в виде системы различных моделей. Разработано программное обеспечение для подключения исходных и целевых баз данных, которое позволяет проводить процесс миграции данных из созданных моделей. Генерируемые значения представляются в виде полноценных объектов и могут быть использованы для создания веб-интерфейса приложений с возможностью редактировать модели данных и управлять системой моделей.*

Заключение. *Экспериментальные исследования по десериализации моделей из JSON-строки, содержащих сложные классы моделей, показали среднее значение точности определения типа данных моделей в 92% случаев, в частности при определении типов значений «символ» и «строка». Созданные модели представляются в виде таблицы данных и могут быть использованы для обеспечения миграции данных моделей.*

Ключевые слова: *десериализация; СУБД; обмен данными; веб-сервисы; информационная система; базы данных; статическая типизация.*

Конфликт интересов: *Авторы декларируют отсутствие явных и потенциальных конфликтов интересов, связанных с публикацией настоящей статьи.*

Для цитирования: Томакова Р. А., Иванов Д. В., Корсунский Н. А. Интеллектуальная система обеспечения миграции посредством динамической десериализации данных // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. 2023. Т. 13, № 3. С. 31–51. https://doi.org/10.21869/2223-1536-2023-13-3-31-51.

Поступила в редакцию 02.07.2023 *Подписана в печать 01.08.2023* *Опубликована 29.09.2023*

Intelligent System for Providing Migration Through Dynamic Data Deserialization

Rimma A. Tomakova[1] ✉, Dmitry V. Ivanov[1], Nikita A. Korsunsky[1]

[1] Southwest State University
50 Let Oktyabrya Str. 94, Kursk 305040, Russian Federation

✉ e-mail: rtomakova@mail.ru

Abstract

The purpose of research. *Timely provision of data transfer between information systems allows you to quickly exchange resources. However, applications may have different data formats and structures. Therefore, the aim of the research was to develop models, methods and algorithms for a system of asynchronous deserialization of a data string, providing an increase in the efficiency of determining data models by generating strongly typed objects.*
Methods. *The way to deserialize models from data involves line-by-line decomposition of a JSON-file line with the definition of key-value types and their correlation with the data model: character, string, number, boolean value. After that, the web controller conducts asynchronous generation of the class and its objects. To classify string values, serialized string value classifiers are used. For asynchronous generation of objects, a system of "contracts" of models and algorithms for executing and converting these models are used.*
Results. *The deserializer consists of a system of four model analysis controllers and a value generation algorithm. A simple single model deserialization model allows the model to be mapped to relational database table headers to enable model migration between systems. The generated objects are represented by static data types, which ensures that they can be written to any DBMS system using built-in tools. A complex model represents a block of values as a system of different models. Software has been developed for connecting source and target databases, which allows you to migrate data from the created models. Generated values are represented as full-fledged objects and can be used to create a web interface for applications, edit data models, and manage the model system.*
Conclusion. *Experimental studies on deserialization of models from a JSON string containing complex model classes showed an average accuracy of determining the data type of models in 92% of cases, in particular when determining the types of values "character" and "string". The created models are presented in the form of a data table and can be used to ensure the migration of these models.*

Keywords: *deserialization; DBMS; data exchange; web services; information system; databases; static typing.*

Conflict of interest: *The Authors declare the absence of obvious and potential conflicts of interest related to the publication of this article.*

For citation: Tomakova R. A., Ivanov D. V., Korsunsky N. A. Intelligent System for Providing Migration Through Dynamic Data Deserialization. *Izvestiya Yugo-Zapadnogo gosudarstvennogo universiteta. Serija: Upravlenie, vychislitel'naja tekhnika, informatika. Meditsinskoe priborostroenie = Proceedings of the Southwest State University. Series: Control, Computer Engineering, Information Science. Medical Instruments Engineering.* 2023; 13(3): 31–51. (In Russ.) https://doi.org/10.21869/2223-1536-2023-13-3-31-51.

Received 02.07.2023 *Accepted 01.08.2023* *Published 29.09.2023*

***

Введение

Проблема миграции данных позволяет решить вопрос с хранением, распределением и обработкой информации и является актуальной для большинства организаций, работающих с распределенными системами хранения информации [1]. Использование механизмов удаленной обработки информации в случае распределенных систем хранения данных требует использования различных программ, файлов разных форматов, реляционных баз данных. Подключение к таким распределенным системам с помощью различных драйверов или физическая выгрузка и перенос памяти приводят к большим издержкам во времени, безопасности и даже могут привести к повреждению данных.

В связи с этим возникает необходимость в разработке интеллектуальной системы для обработки входящих данных из формата JSON, их десериализации и интерпретации системой для последующего использования и сохранения в любой директории или базе данных, расположенных в распределенной сети и связанных между собой посредством глобальной сети. Поэтому разработка алгоритмов обеспечения миграций данных, загрузки, сохранения, чтения коллекций данных в интерпретируемые объекты посредством десериализации является актуальной задачей [2; 3].

Интерпретация текстовой строки для работы с файловой системой и объектами позволяет экономить на объеме данных и скорости их передачи, а возможность асинхронного чтения данных и их преобразования на уровне интеллектуальной системы обеспечения миграции позволит проводить операции над всеми доступными базами данных и директориями, в которых работают сотрудники предприятия. Такой подход является актуальным для многих областей, где требуется оперативная реакция на любые изменения в системе и возможность императивного обновления информации в дочерних системах через центральный узел, в качестве которого и выступает разрабатываемая система динамической десериализации [4].

Своевременное обеспечение миграции данных с формированием интерпретируемых информационной системой структур, обработкой изменений и загрузкой данных через сетевые протоколы является сложной задачей с формированием пар моделей данных и определением типов значений с помощью языка программирования со строгой типизацией. Эта задача связана с процедурами отслеживания изменений в системе и её узлах, обработки больших коллекций данных, сопоставления классов и структур и проведения обмена [5; 6].

В статье был предложен метод десериализации моделей данных, основанный на системе сопоставления контрактов словарей пар типа «ключ:значение» из строки JSON, конвертации строк в массив значений и сопоставления ключевых значений для асинхронной генерации моделей данных.

В настоящее время разработаны разнообразные методы, модели и алгоритмы для решения проблемы генерации объектов и десериализации даных в рабочие структуры [7; 8].

S. Cao, S. Di Girolamo, T. Hoefler предложили свой метод десериализации моделей данных с помощью переноса на полностью программируемые сетевые адаптеры Smart-NIC и выполнение на пути данных для каждого пакета. Это решение позволяет избежать промежуточных копий памяти, обеспечивая десериализацию «на лету». Они демонстрируем подход, разгружая Google Protocol Buffers, широко используемую платформу для сериализации / десериализации данных. Их оценка говорит о возможности ускорения процесса миграции в 4,8 раза. Затем с помощью моделирования пропускной способности микросервиса они показывают, как можно улучшить общую пропускную способность, организовав десериализацию и фактические действия приложений с помощью PsPIN [9].

A. Bartel, E. Bodden, Y. Le Traon и I. Sayar проводят исследование уязвимости метода десериализации данных напрямую из СУБД. Кроме того, они выявили, что среди исследованных библиотек 37,5% не исправлены, что оставляет гаджеты доступными для кибератак. Выявлено более 104 уязвимости CVE на платформе Java [10; 11].

B. Huang,Y. Tang в своей работе по использованию технологии Protobuf изучают процесс сериализации информации для хранения данных. Метод змеевидного разрыва используется для завершения интервала распределения узлов последовательности, так что рабочее состояние и состояние покоя всегда поддерживают динамический баланс. В соответствии с правилами первого уровня получают данные хранения завершенного целевого узла, анализируют грамматическую структуру и семантику целевых данных. При этом устанавливаются соответствующие соответствия, и информация хранения данных сериализуется. Чтобы проверить эффективность метода сериализации информации для хранения данных Protobuf, разработан сравнительный эксперимент. При использовании трех методов: HDVM, Redis и Protobuf – для сериализации данных JSON сравнительный анализ показывает, что HDVM имеет самое длительное время обработки, а Protobuf – самое короткое время обработки, и целостность данных не затрагивается. Данные моделирования показывают, что метод сериализации Protobuf имеет короткое время преобразования, высокое использование пространства и очевидные преимущества в правильности и целостности. Он подходит для сериализации данных JSON в случае ограниченной пропускной способности [12].

Материалы и методы

Передаваемый и загружаемый в систему пакет данных представляет из

себя коллекцию файлов, часть из которых имеет формат JSON, текстовый файл в виде массива строк. Передаваемый массив построчно и асинхронно разбирается на составные модели: классы, поля классов в виде типов данных и коллекции их значений, а затем создаётся в формате временных значений, с которыми можно работать в рамках интеллектуальной системы или связанных с ней приложений, в т. ч. для чтения и записи значений в базы данных.

Алгоритм передачи моделей данных через веб-сервис

Чтение моделей данных, получаемых информационной системой через веб-сервисы, имеет объектный формат JSON. Благодаря RESTful архитектурному стилю модели приложения, посредством HTTP-протокола реализована возможность передачи объекта в виде строки.

На данном этапе происходит процесс передачи модели данных в управляемое приложение. Используя POST-запрос HTTP-протокола происходит отправка на сервер модели данных одним из способов:

– поточным методом посредством подключения базы данных к информационной системе и выгрузки модели в формате JSON на программном уровне;

– посредством передачи файла формата JSON.

Можно выделить основные преимущества такой модели:

– использование Web-API позволяет взаимодействовать со всеми приложениями, находящимися в распределенной системы сети Интернет;

– можно использовать любые способы проведения миграций данных: консолидацию, распределение и перераспределение между информационными системами;

– позволяет работать с файловыми системами и проводить с их помощью миграции;

– позволяет проводить упрощение сложных моделей данных;

– является одним из наиболее популярных форматов передачи данных и может переводиться в объекты программ без необходимости предварительной записи в таблицы данных;

– поддерживает механизмы автоматизации миграции данных без простоя системы;

– контроллеры поддерживают асинхронные методы, что позволяет проводить миграцию данных без времени простоя.

Такой способ является наиболее эффективным при необходимости организации работы распределенной информационной системы и может быть интегрирован с технологией облачной миграции данных [13; 14].

Процесс миграции данных подразумевает сериализацию или композицию системы объектов в одну единую модель данных. Система объектов представляет из себя неупорядоченное множество пар типа «ключ: значение», взаимодействие с которыми проходит по принципу словаря [15].

Ключ – это название параметра или свойства, который передается интеллектуальной системе.

Модель данных поддерживает передачу различных типов информации. В таблице 1 представлены типы ключей, передаваемые в объекте миграции данных, и их описание.

Таблица 1. Типы ключей данных сериализуемые в модель миграции [16]

Table 1. Types of data keys serialized in the migration model [16]

Тип ключа	Описание
Число	С плавающей точкой двойной точности
Строка	Unicode, содержащей двойные кавычки и обратную косую черту
Логический тип	Истинность / Ложность
Массив	Упорядоченная последовательность однотипных значений
Значение	Все остальные типы значений (var-переменная в языках динамической типизации)
Объект	Набор пар «ключ:значение» неупорядоченного характера
Пробел	—
Nullable-тип	Обозначает пустое значение в паре «ключ:значение»

Модель передачи данных подразумевает проведение миграции данных с сопоставлением общей информационной модели.

Пример реляционной базы данных, загружаемой в информационную систему, представлен ниже (рис. 1).

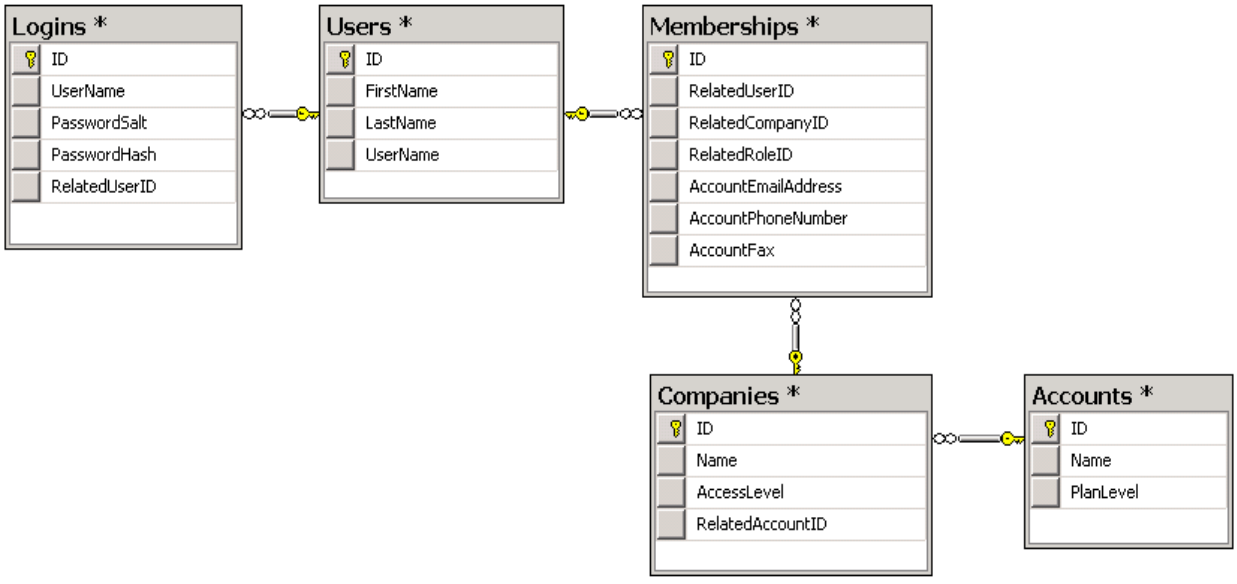


Рис. 1. Пример модели базы данных пользователей

Fig. 1. User Database Model Example

Композиция модели может происходить по нескольким сценариям: пользователь может выгрузить в формат объекта всю базу, одну отдельную или несколько таблиц, отдельные строки. Помимо этого, при желании можно также загрузить каждую отдельную таблицу в отдельный файл формата JSON с целью структуризации процесса обработки данных.

Рассмотрим пример файла с сериализованной моделью данных (рис. 2).


```
{
  "Logins":{
        "id":4,
        "UserName":"test",
        "PasswordSalt": "test1",
        "PasswordHash":1,
        "RelatedUserID":4
    },
      "Users":{
        "ID":4,
        "FirstName": "testname",
        "LastName": "testlastname",
        "UserName": "John"
      },
    "Memberships":{
        "id":4,
        "RelatedUserID": "test",
        "RelatedCompanyID": "test1",
        "RelatedRoleID":1,
        "AccointEmailAdress":4,
        "AccountPhoneNumber": "8-941-344-11-24",
        "AccountFax": null
    },
      "Companies":{
        "Name": "FX1",
        "AccessLevel": "imp",
        "RelatedAccountID": 3141
      },
    "Accounts":{
      "ID": 4,
    "Name": "tes",
      "PlanLevel":1
    }
}
```


Рис. 2. Пример загружаемой модели данных

Fig. 2. Loadable data model example

Следует отметить, что структура представленного файла состоит из массива строк, каждая из которых представляет из себя определенный тип ключа, в т. ч. тип объекта, внутри которого перечислены свойства [17].

Файл загружается в указанную администратором системы директорию. В нашем случае будет загружен один файл в главную директорию системы обеспечения миграции. После успешного сохранения файла пользователь будет извещен о результатах. На рисунке 3 изображен результат загрузки файла модели базы данных в систему.

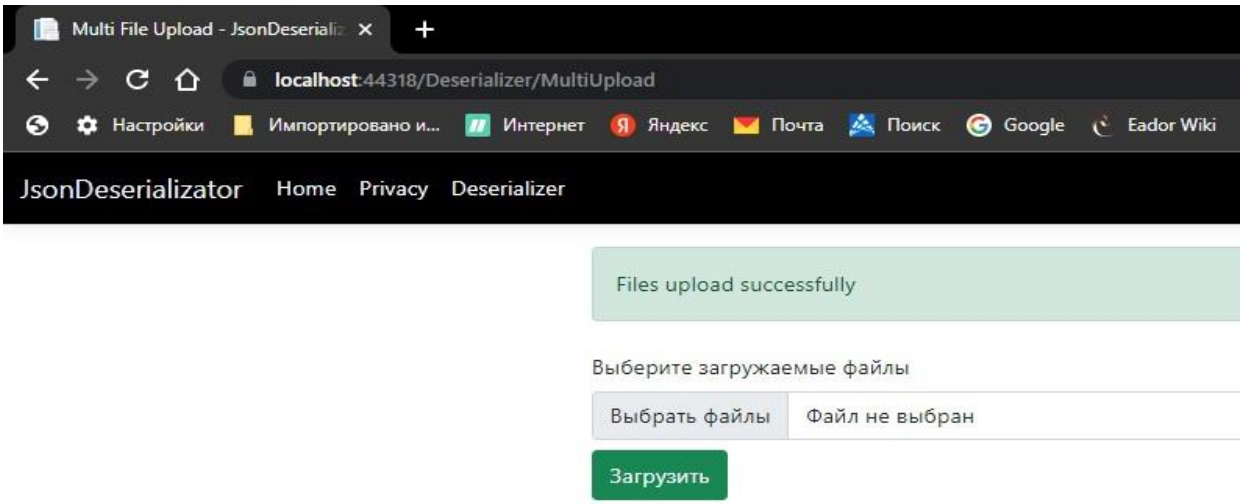


Рис. 3. Результат загрузки файла

Fig. 3. File upload result

Структурные и алгоритмические решения для динамической десериализации модели данных

Модель динамической десериализации данных представляет под собой асинхронное чтение файла и последующую интерпретацию его в систему классов и объектов. В этом процессе применяется генерация динамической структуры данных, которая затем может быть использована в различных приложениях или записана в другую базу данных. Для статического определения полей модели данных следует разработать собственную модель интерпретации: система должна создавать асинхронно и динамически класс и коллекцию его объектов программным способом [18; 19].

Анализ модели данных, представленных на рисунке 2, свидетельствует о том, что файл указанного формата может быть интерпретирован как массив, состоящий из n-строк. В таком случае, проведя декомпозицию файла с учетом того, что каждая отдельная строка является самостоятельной парой «ключ:значение», за исключением полей с типом ключа «пробел», можно создать коллекцию значений [20].

Следовательно, если организовать построчный вывод модели данных,

можно интерпретировать общую модель данных как массив. Реализуем Get-метод веб-сервиса, в котором файл формата JSON будет считываться системой. Результат построчного чтения модели данных и его открытия в браузере показан ниже (рис. 4).

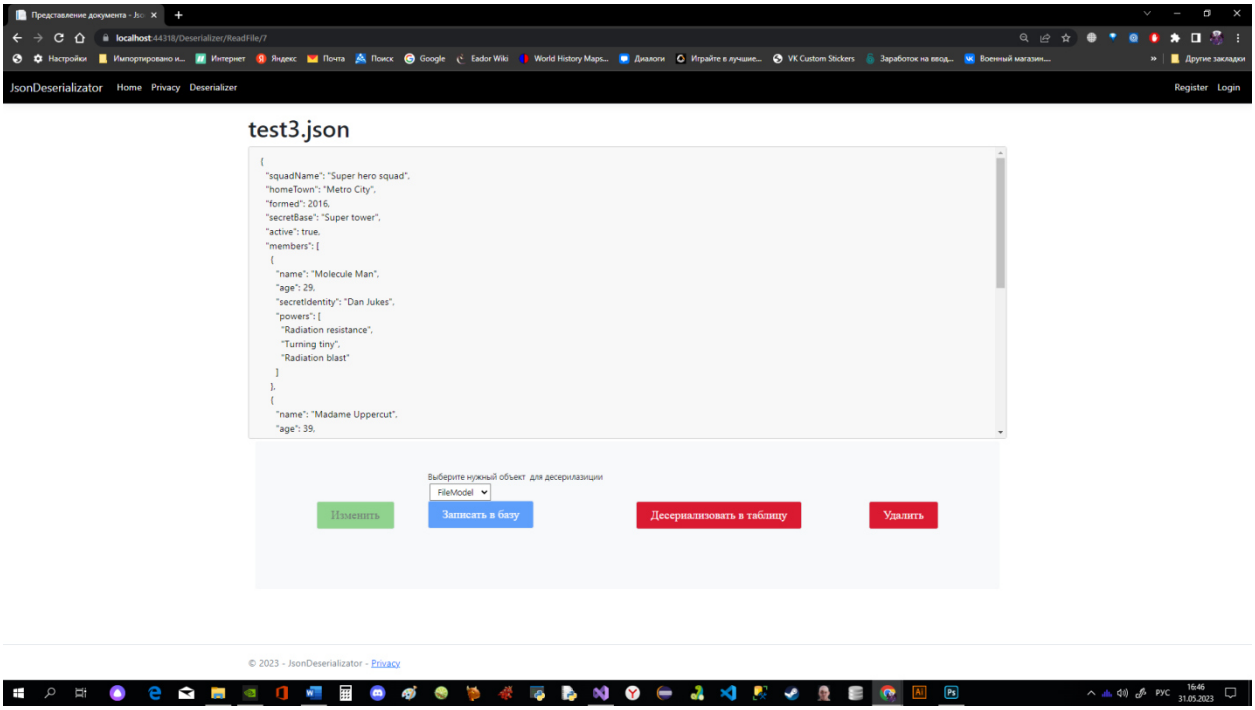


Рис. 4. Результат чтения файла информационной системой

Fig. 4. The result of reading the file by the information system

Однако каждую отдельную строку необходимо интерпретировать из строки в определенное значение выгружаемой модели данных.

Для реализации функции создания динамического объекта нужно ввести понятия существующего значения (Existing Value, EV) и несуществующего значения (Unexisting Value, UV) сериализованной строки. Причем представляемый информационной системой как динамический объект, он позволяет создавать коллекцию значений булевых типов, каждый из которых возвращает в процессе перебора данных вложенное в него значение. Условие принадлежности строки массива к существующему значению записывается в виде

$$\begin{cases} L_n = EV, \\ L_n \neq UV, \end{cases} \quad (1)$$

где UV является пустой строкой.

Например, если строка имеет тип «Пробел», то система вернёт False – значения не существует, и системой будет пропущено.

В таком случае EV является коллекцией значений, массивом с динамически определяемым размером, каждый элемент которого можно десериализовать в определенное значение.

Для десериализации существующих значений необходимо ввести понятие

функции определения значения (definition value function, DVF). Так, если проверка EV возвращает true, то происходит запись строки потока данных, определяя итоговую коллекцию контрактов при исполнении функции DVF.

Также нужно ввести понятие «исполнитель контрактов» (Contract Resolver, CR), который проводит десериализацию каждого контракта из коллекции. Так свойство (property) контракта с идентификатором порядкового номера коллекции «4» имеет тип «””active””:true,».

Нужно ввести понятие функции определения контракта (contract definition function, CDF), которая перебирает значения в коллекции исполнителей контрактов, возвращая поле динамической коллекции свойств. Динамический параметр определяет тип данных: числовой, булевый или текстовый формат. Например, если контракт имеет тип «””active””:true,», то исполнитель контрактов проводит десериализацию значения как Boolean тип данных, передает в параметр название active и его значение true.

CDF представляет из себя массив уникальных значений ключей, встречаемых системой единожды. При повторении типа параметра происходит сопоставление исполнителя контрактов, т. е. значение параметра добавляется в массив объектов, однако ключ-параметр становится общим для всей системы.

Суть процесса определения системы контрактов демонстрирует рисунок 5. Здесь массив объектов с набором одинаковых полей интерпретируется из массива строк, создавая с помощью функции CDF из коллекции исполнителя контрактов CR, которые, в свою очередь, сформированы из массива строк с помощью функции определения значения.

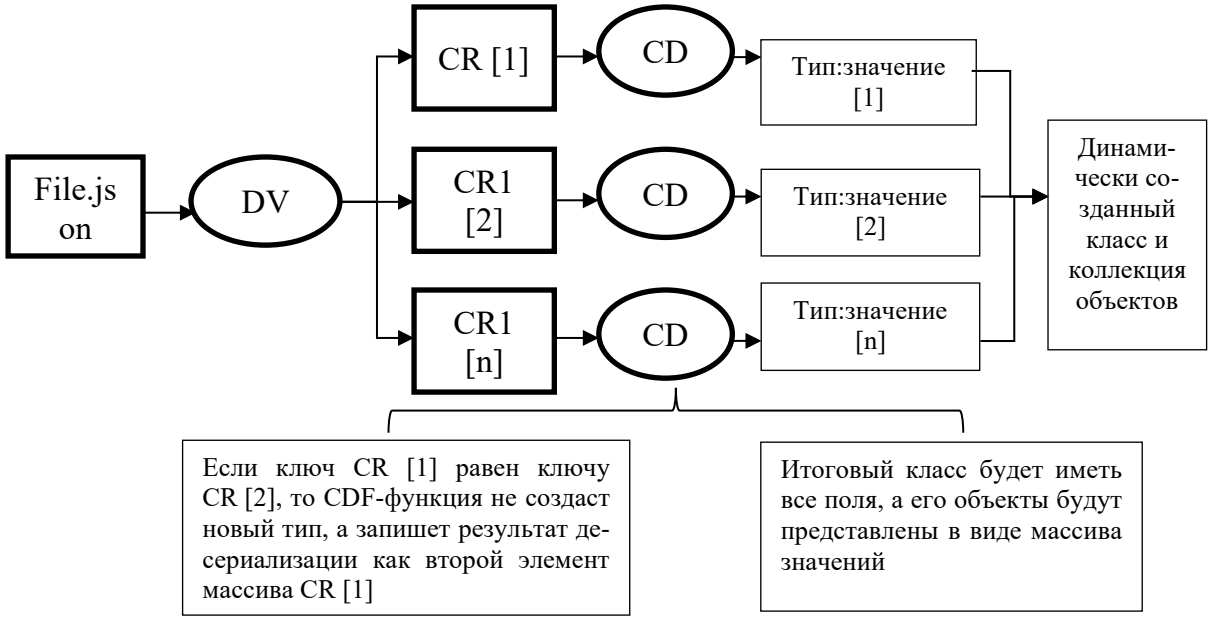


Рис. 5. Концептуальная модель процесса определения системы контрактов функций DVF, CDF

Fig. 5. Conceptual model of the process of defining a system of contracts for functions DVF, CDF

Помимо этого, следует ввести понятие функции автоконвертера параметров (Parameters AutoConverters Function, PACF). PACF принимает в качестве параметра выходное значение функции определения контракта CDF, после чего генерирует итоговый файл в статический класс с набором свойств из исполнителя контрактов CR, храня значения: идентификатор, название, значение.

Каждый объект внутри передаваемого файла модели данных представляет из себя отдельный сегмент, определенный в фигурных кавычках. На рисунке 6 показана концептуальная модель сегментации файла модели.

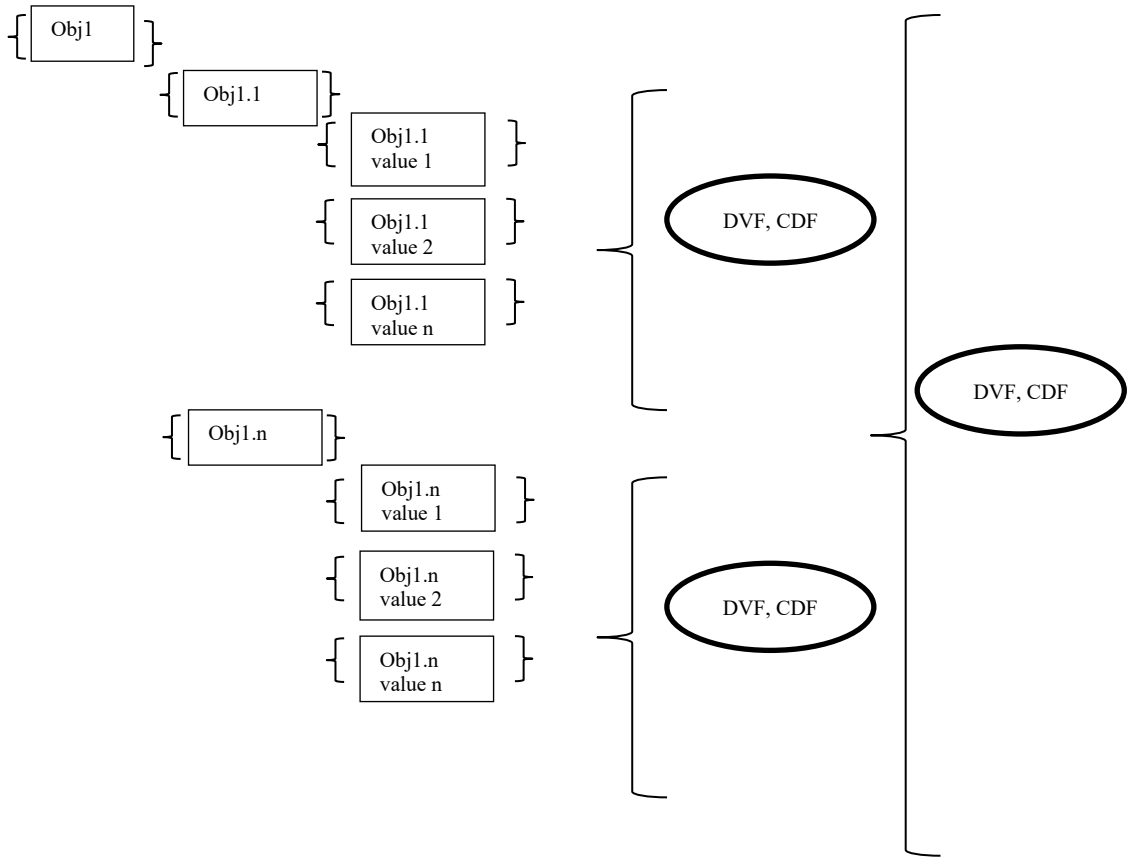


Рис. 6. Концептуальная модель сегментации файла модели

Fig. 6. Conceptual model of model file segmentation

Разработанная информационно-вычислительная система создает динамическую структуру данных, которая представляет из себя конвертированную коллекцию CR исполнителей контрактов, генерирующую с помощью функции автоконвертера параметров свойства возвращаемого динамического объекта, который интерпретируется системой как объект.

Значения этого объекта могут как быть записаны в сопоставляемые таблицы данных (при этом система определения данных позволяет пользователю выбрать, какие значения следует прове-

сти, а какие – удалить) или быть использованными в сторонней программе, подключенной к распределенной системе. Например, если наше приложение принимает данные из исходной СУБД, проводит сериализацию данных и передает их на обработку как объект в целевые системы.

Пусть, например, наше веб-приложение получило файл формата .json, тогда система должна вернуть динамически созданные классы и объекты.

Поскольку структура является динамической и создается при обращении к контроллеру веб-приложения с использованием Get-метода, данные в разрабатываемом методе находятся только в оперативной памяти. Информация о сгенерированной структуре будет удалена при переходе на новую страницу, если проведение миграции данных на текущий момент не требуется. Помимо этого, данный механизм генерации динамического объекта и их записи позволяют передавать параметры в любую существующую СУБД, к которой веб-приложение имеет доступ.

Временная сложность расчета алгоритма десериализации данных DVF функции записывается в виде

$$T_n = O(CR \cdot EV), \qquad (2)$$

где T_n – время в мс; O – число объектов внутри модели; CR – число исполнителей контрактов; EV – число существующих значений.

В случае, если все значения в файле имеют данные, то функция является квадратической типа:

$$T_n = O(k^2), \qquad (3)$$

поскольку в таком случае $CR = EV$.

Временная сложность вывода данных t в динамический объект классов равняется

$$t = O(m), \qquad (4)$$

где m – PACF автоконвертера данных, следовательно, функция является линейной.

Временная сложность заполнения переменных tv генерируемых полей классов значениями равняется

$$tv = O(td), \qquad (5)$$

где t – количество полей классов; d – размер массива данных, передаваемых в конкретное значение, являясь результатом перемножения двух линейных алгоритмов.

Следовательно, алгоритм всей десериализации данных O может быть представлен в виде общей линейно-квадратической функции типа

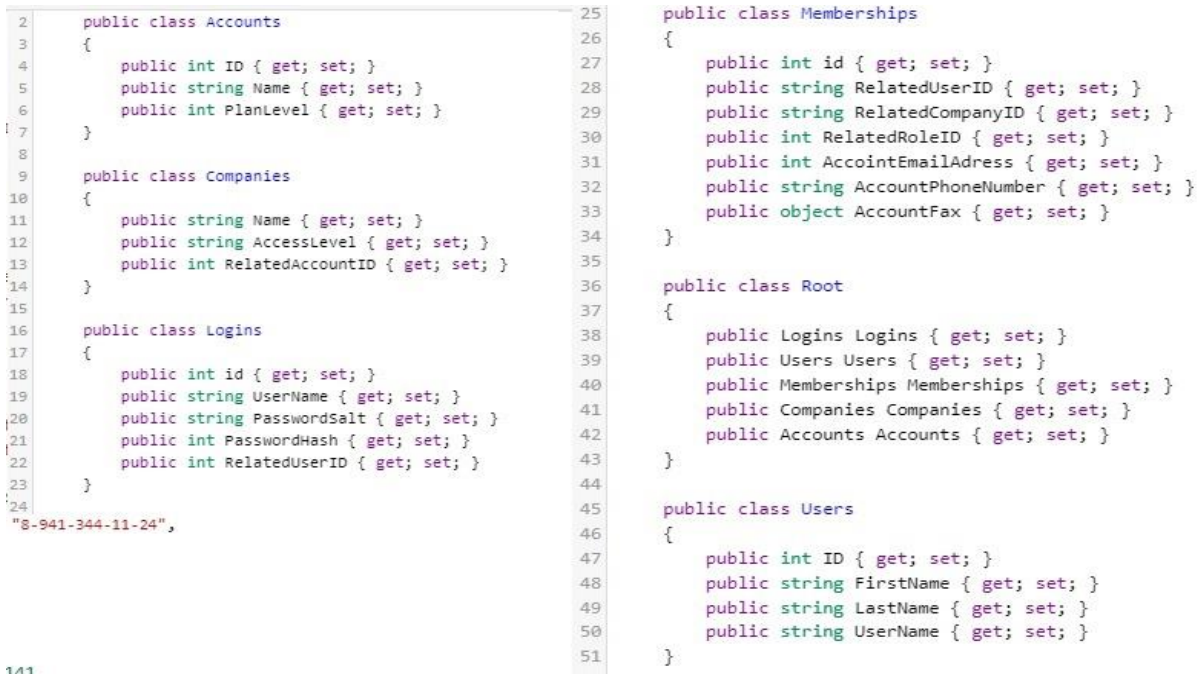
$$O = O_{kf} + O_m + O_{td}, \qquad (6)$$

или

$$O = T_n + t + tv. \qquad (7)$$

Рассмотрим на примере. На рисунке 7 изображен системный результат десериализации модели данных из рисунка 2 в виде процедурно генерируемой системы классов.

В свою очередь, DVF-функция создает систему классов, которые могут иметь коллекцию значений в виде объектов. На рисунке 8 изображены генерируемые CDF-функцией объекты, наследуемые от классов из рисунка 7.



```
    public class Accounts
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public int PlanLevel { get; set; }
    }

    public class Companies
    {
        public string Name { get; set; }
        public string AccessLevel { get; set; }
        public int RelatedAccountID { get; set; }
    }

    public class Logins
    {
        public int id { get; set; }
        public string UserName { get; set; }
        public string PasswordSalt { get; set; }
        public int PasswordHash { get; set; }
        public int RelatedUserID { get; set; }
    }

"8-941-344-11-24",
```

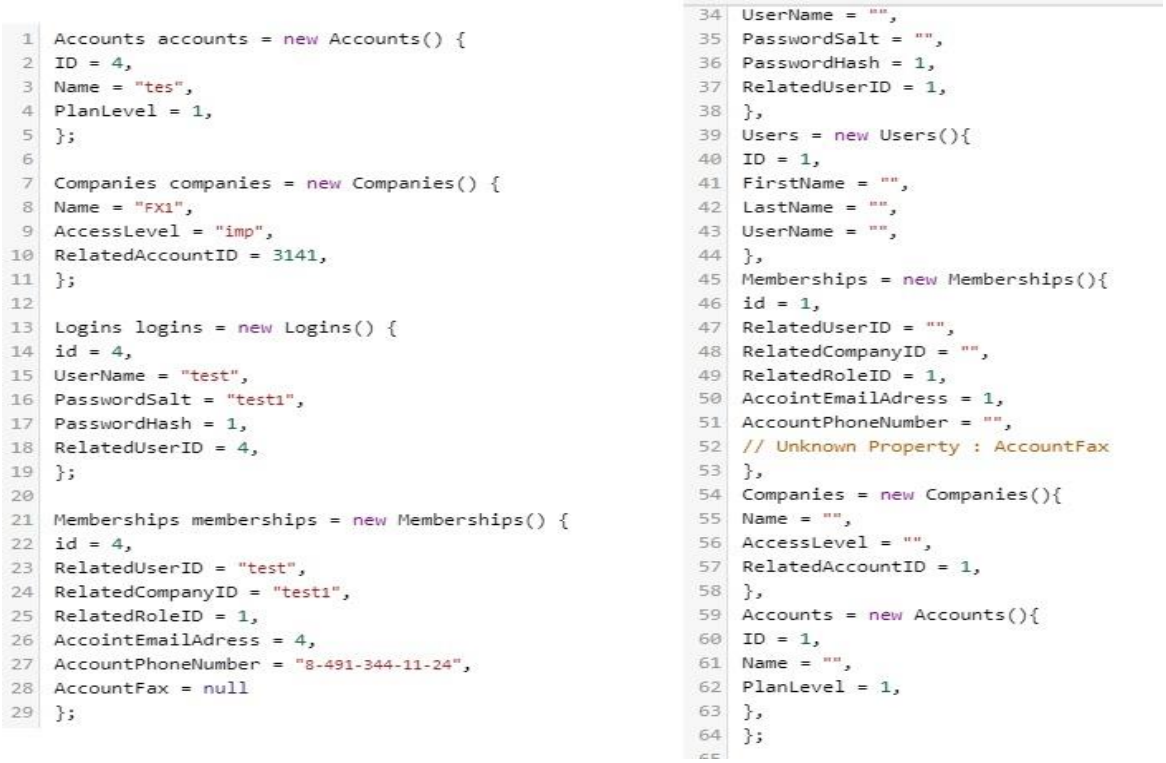
```
public class Memberships
{
    public int id { get; set; }
    public string RelatedUserID { get; set; }
    public string RelatedCompanyID { get; set; }
    public int RelatedRoleID { get; set; }
    public int AccointEmailAdress { get; set; }
    public string AccountPhoneNumber { get; set; }
    public object AccountFax { get; set; }
}

public class Root
{
    public Logins Logins { get; set; }
    public Users Users { get; set; }
    public Memberships Memberships { get; set; }
    public Companies Companies { get; set; }
    public Accounts Accounts { get; set; }
}

public class Users
{
    public int ID { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string UserName { get; set; }
}
```

Рис. 7. Результат работы системы в виде генерируемого значения классов

Fig. 7. The result of the system operation in the form of a generated class value



```
Accounts accounts = new Accounts() {
ID = 4,
Name = "tes",
PlanLevel = 1,
};

Companies companies = new Companies() {
Name = "FX1",
AccessLevel = "imp",
RelatedAccountID = 3141,
};

Logins logins = new Logins() {
id = 4,
UserName = "test",
PasswordSalt = "test1",
PasswordHash = 1,
RelatedUserID = 4,
};

Memberships memberships = new Memberships() {
id = 4,
RelatedUserID = "test",
RelatedCompanyID = "test1",
RelatedRoleID = 1,
AccointEmailAdress = 4,
AccountPhoneNumber = "8-491-344-11-24",
AccountFax = null
};
```

```
UserName = "",
PasswordSalt = "",
PasswordHash = 1,
RelatedUserID = 1,
},
Users = new Users(){
ID = 1,
FirstName = "",
LastName = "",
UserName = "",
},
Memberships = new Memberships(){
id = 1,
RelatedUserID = "",
RelatedCompanyID = "",
RelatedRoleID = 1,
AccointEmailAdress = 1,
AccountPhoneNumber = "",
// Unknown Property : AccountFax
},
Companies = new Companies(){
Name = "",
AccessLevel = "",
RelatedAccountID = 1,
},
Accounts = new Accounts(){
ID = 1,
Name = "",
PlanLevel = 1,
},
};
```

Рис. 8. Результат работы системы в виде генерируемых коллекций объектов

Fig. 8. The result of the system operation in the form of generated collections of objects

На рисунке 9 представлена схема модифицированного алгоритма десериализации данных с помощью алгоритма генерации динамических объектов.

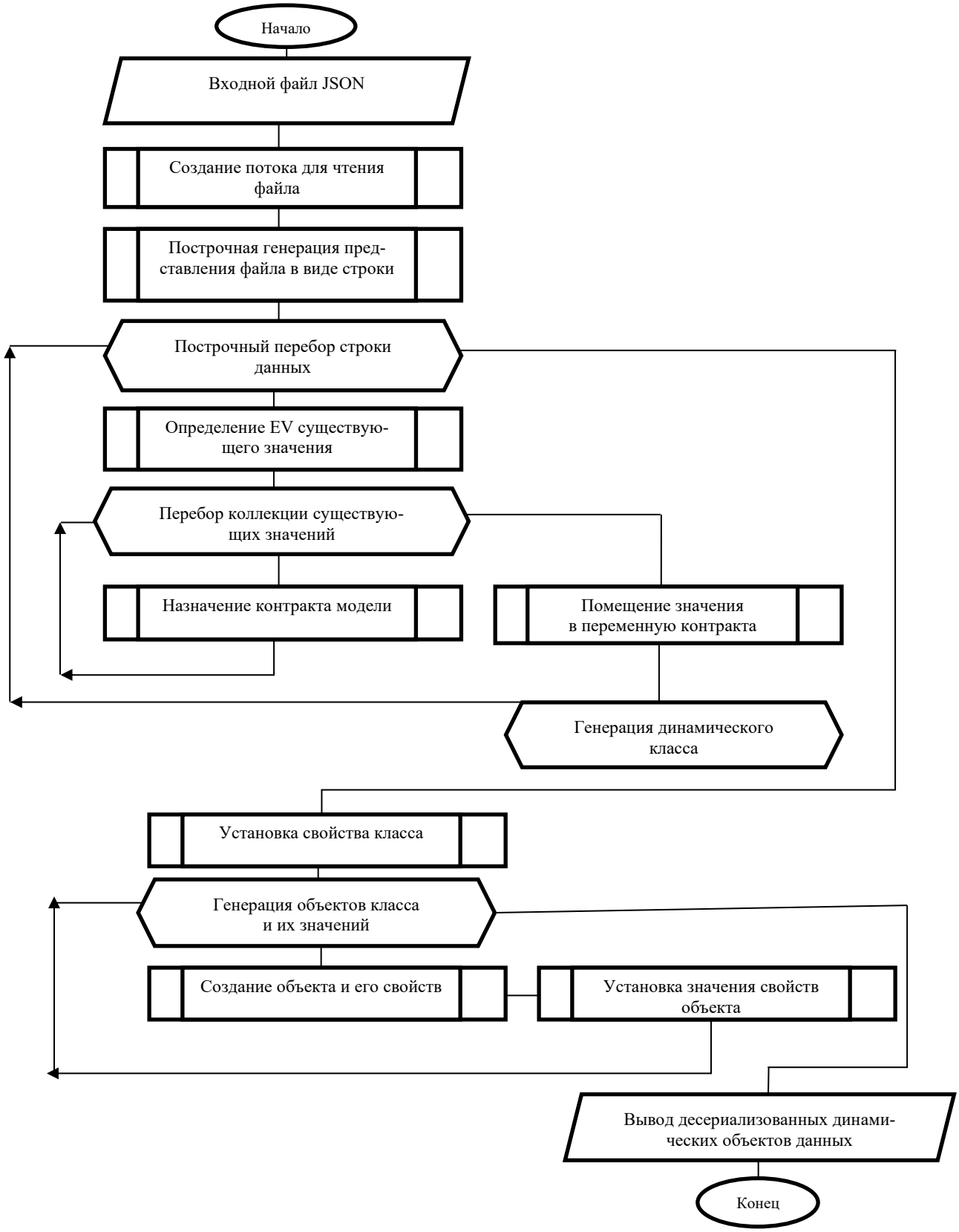


Рис. 9. Схема модифицированного алгоритма десериализации данных на основе генерации динамических объект

Fig. 9. Scheme of the modified data deserialization algorithm based on the generation of dynamic objects

После выполнения процесса десериализации пакетов данных создается система объектов и классов, которая может быть интерпретирована системой сразу (например, визуально быть представлена в виде таблицы или системы объектов), а также быть записана в любую реляционную базу данных.

Результаты и их обсуждение

Для оптимизации работы десериализатора моделей данных было разработано веб-приложение обеспечения миграции данных. На рисунке 10 изображено интерфейсное окно главного меню программного обеспечения. Приложение поддерживает аутентификацию и авторизацию пользователей.

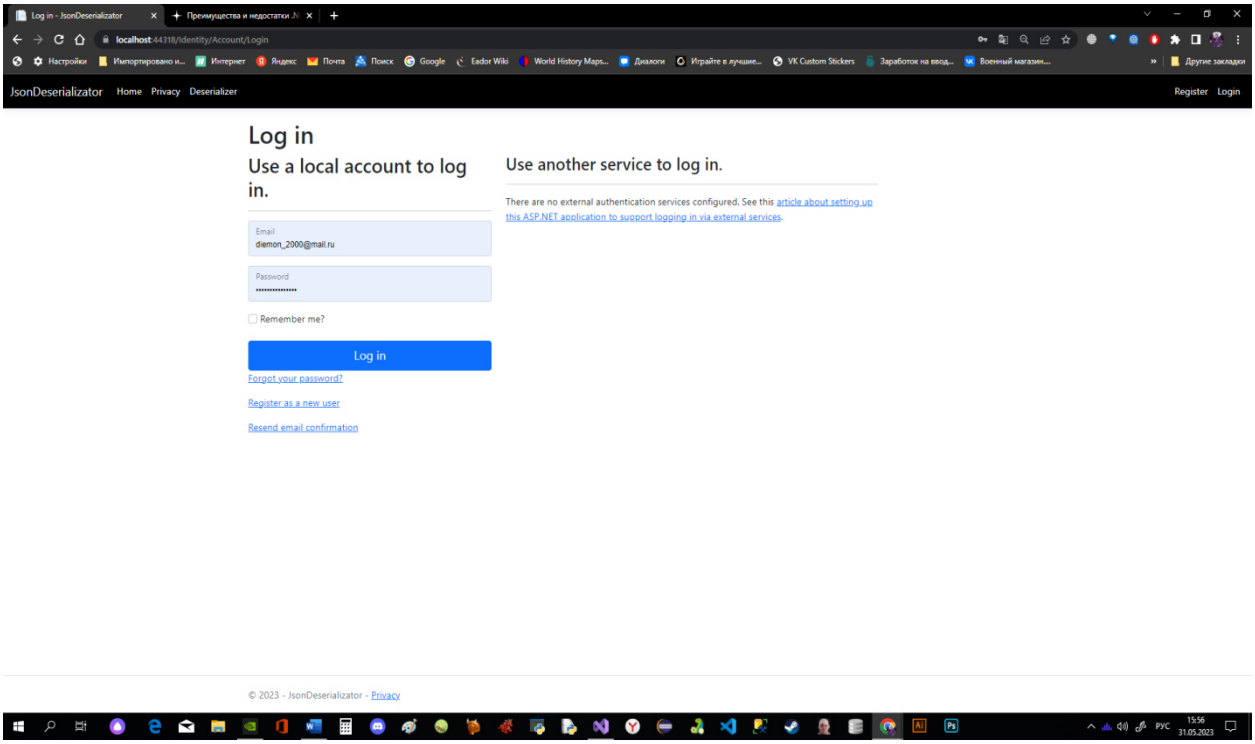


Рис. 10. Интерфейсное окно главного меню программного обеспечения

Fig. 10. Interface window of the main menu of the software

Главное меню включает систему ссылок, с помощью контроллеров реализована маршрутизация по сайту с помощью MVC-модели. Модуль Home отправляет пользователя в главное меню сайта.

Модуль Deserializer предоставляет пользователю доступ к директории, указанной администратором или выбранной им из предоставленных. На рисунке 11 изображено окно представления директории сервера.

На рисунке 3 был показан процесс загрузки файлов. В этот раздел можно перейти из меню раздела Deserialize.

Открытие файла работает по модели адресации с системой id. Переходя по нужной ссылке, происходит асинхронное открытие документа с возможностью его чтения и изменения.

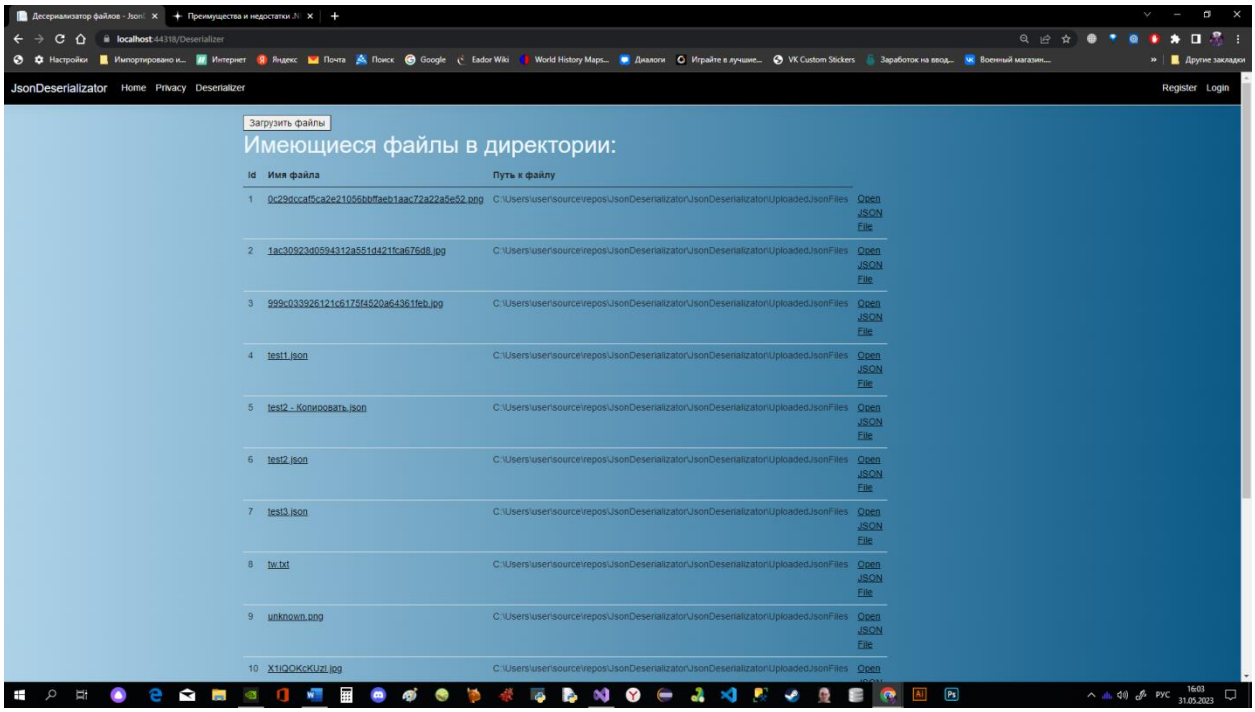


Рис. 11. Окно раздела Deserializer

Fig. 11. Deserializer section window

При нажатии на кнопку «Десериализовать в таблицу» со стороны сервера происходит применение механизма разбора строки в однотипные объекты на языке C#, представленные на клиенте. Кнопка «Записать в базу» становится активной и позволяет сохранить данные, сопоставленные с моделью (список которых загружается из базы данных). На рисунке 12 изображен результат десериализации и создания объекта из файла test3.json.

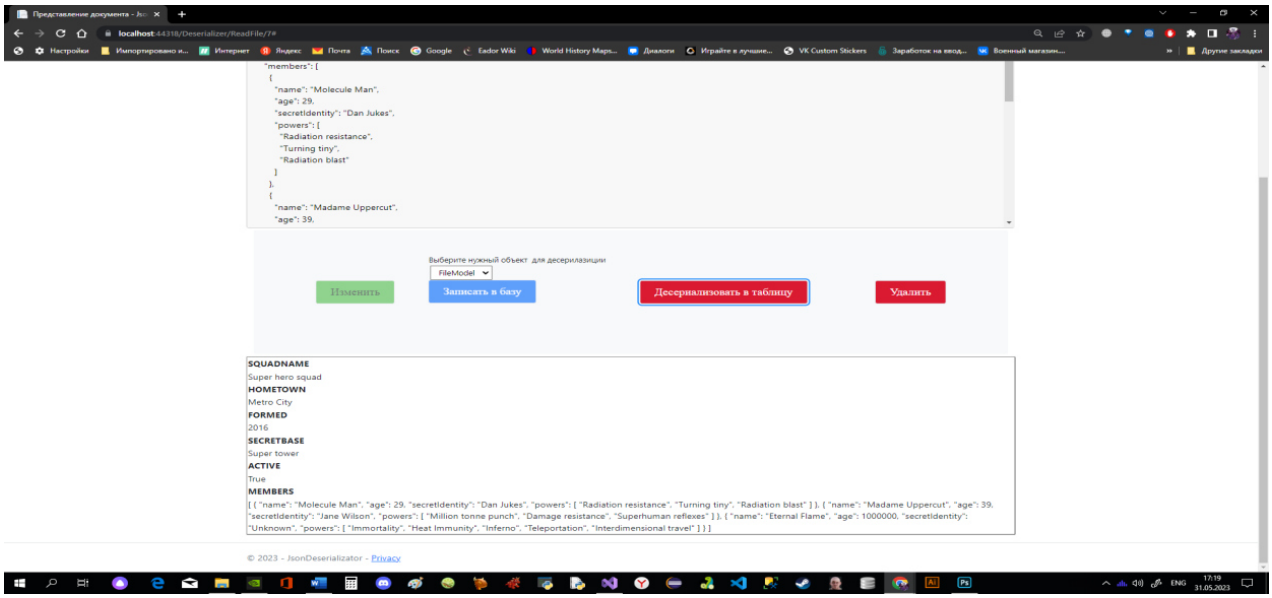


Рис.12. Результат десериализации файла test3.json

Fig. 12. The result of the deserialization of the file test3.json

Модель данных может быть загружена в базу данных при условии сопоставления всех полей объекта. В случае, если таблицы на текущий момент не существует, система с помощью Entity Framework создает новую таблицу и заполняет её значениями.

Выводы

1. Разработан метод динамической десериализации моделей данных. Десериализатор состоит из системы четырех контроллеров анализа моделей и алгоритма генерации значений. Простая модель десериализации одной модели позволяет сопоставить модель с заголовками таблицы реляционной базы данных для обеспечения миграции модели между системами. Генерируемые объекты представляются статическими типами данных, что обеспечивает их запись в любую систему СУБД встроенными средствами. Сложная модель представляет блок значений в виде системы различных моделей. Разработано программное обеспечение для подключения исходных и целевых баз данных, которое позволяет проводить миграции данных из созданных моделей. Генерируемые значения представляются в виде полноценных объектов и могут быть использованы для создания веб-интерфейса приложений, редактировать модели данных и управлять системой моделей.

С помощью разработанных алгоритмов функции автоконвертации PACF и её методов DVF, CDF происходит асинхронная генерация словарей с парами «ключ-значение», сопоставление ключей и валидация загружаемой модели данных.

2. Установлены критерии качества в виде скорости десериализации модели данных и точности определения модели. Экспериментальные исследования по десериализации моделей из JSON-строки, содержащих сложные классы моделей, показали среднее значение точности определения типа данных моделей в 92% случаев, в частности, при определения типов значений «символ» и «строка». Созданные модели представляются в виде таблицы данных и могут быть использованы для обеспечения миграции данных моделей.

Список литературы

1. Метод классификации рентгенограмм на основе использования глобальной информации об их структуре / Р. А. Томакова, М. В. Томаков, И. В. Дураков, В. В. Жилин // Биомедицинская радиоэлектроника. 2016. № 9. С. 45–51.

2. Нейронные сети с макрослоями для классификации и прогнозирования патологий сетчатки глаза / Н. А. Кореневский, Р. А. Томакова, С. П. Серегин, А. Ф. Рыбочкин // Медицинская техника. 2013. № 4 (280). С. 16–18.

3. Нейросетевые модели сегментации ангиограмм глазного дна на основе анализа RGB-кодов пикселей / А. Н. Брежнева, С. А. Борисовский, Р. А. Томакова, С. А. Филист // Системный анализ и управление в биомедицинских системах. 2010. Т. 9, № 1. С. 72–76.

4. Многослойные морфологические операторы для сегментации сложноструктурируемых растровых полутоновых изображений / С. А. Филист, А. Р. Дабагов, Р. А. Томакова, И. А. Малютина // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. 2019. Т. 9, № 3 (32). С. 44–63.

5. Tomakova R. A., Filist S. A., Pykhtin A. I. Automatic Fluorography Segmentation Method Based on Histogram of Brightness Submission in sliding Window // International Journal of Pharmacy and Technology. 2017. Vol. 9, N 1. P. 28220–28228.

6. Томакова Р. А., Шевцов А. Н. Способ построения телекоммуникационной сети передачи данных с борта воздушного судна на наземный диспетчерский пункт // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. 2020. Т. 10, № 1. С. 157–173.

7. Shiyi Cao, Salvatore Di Girolamo, Torsten Hoefler. Accelerating Data Serialization / Deserialization Protocols with In-Network Compute // Workshop on Exascale MPI (ExaMPI) Dallas, TX, USA: EEE, 2023. P. 12–19.

8. Sayar Imen, Bartel Alexandre, Bodden Eric, Le Traon Yves. An In-depth Study of Java Deserialization Remote-Code Execution Exploits and Vulnerabilities // ACM Transactions on Software Engineering and Methodology. 2023. N 1-31. https://doi.org/10.1145/3554732.

9. Juan Antonio Mora-Castillo. Object serialization/deserialization and data transmission with JSON // Revista Tecnología en Marcha. 2016. Vol. 29, is. 1. P. 118–125. https://doi.org/10.18845/tm.v29il.2544.

10. Huang B., Tang Y. Research on optimization of real-time efficient storage algorithm in data information serialization // PLoS One. 2021. N 16(12). P. e0260697. https://doi.org/10.1371/journal.pone.0260697.

11. Software Architecture: 15th European Conference, ECSA 2021 / ed. by S. Biffl, E. Navarro, W. Löwe, M. Sirjani, R. Mirandola, D. Weyns. Sweden: Virtual Event, 2012. 339 p.

12. Борисовский С. А., Брежнева А. Н., Томакова Р. А. Нейросетевые модели с иерархическим пространством информативных признаков для сегментации плохоструктурированных изображений // Биомедицинская радиоэлектроника. 2010. № 2. С. 49–53.

13. Фримен Э., Робсон Э., Сьерра К. Паттерны проектирования. СПб.: Питер, 2011. С. 203–229.

14. Мартин Р. Принципы, шаблоны и методы гибкой разработки на C#. СПб.: Питер, 2019. С. 39–49.

15. Varanasi B., Bartkov M. Spring REST. Building Java Microservices and Cloud Applications. Apress Berkeley, CA, 2022. 243 p.

16. Sarshfield S. Data Migration Best Practices: Strategies for Successful Data Migration Between Applications. Kindle, 2018. 169 p.

17. Ньюмен С. Создание микросервисов. СПб.: Питер, 2016. 304 с.

18. Software Architecture for Big Data and the Cloud / I. Mistrík, R. Bahsoon, N. Ali, M. Heisel, B. Maxim. Elsevier Science, 2017.

19. Morris J. Data Migration Handbook: Practical Advice for Data Migration Projects. London: BCS The Chartered Institute for IT, 2020. P. 231–244.

20. Дулан Д. Миграция данных: практическое руководство по эффективному переносу данных. Wembledon, London: BCS Learning & Development Ltd, 2020. P. 301–312.

References

1. Tomakova R. A., Tomakov M. V., Durakov I. V., Zhilin V. V. Metod klassifikacii rentgenogramm na osnove ispol'zovaniya global'noj informacii ob ih strukture [Method of classification of radiographs based on the use of global information about their structure]. *Biomedicinskaya radioelektronika = Biomedical Radioelectronics*, 2016, no. 9, pp. 45–51.

2. Korenevsky N. A., Tomakova R. A., Seregin S. P., Rybochkin A. F. Nejronnye seti s makrosloyami dlya klassifikacii i prognozirovaniya patologij setchatki glaza [Neural networks with macro layers for classification and prediction of retinal pathologies]. *Medicinskaya tekhnika = Medical Equipment*, 2013, no. 4 (280), pp. 16–18.

3. Brezhneva A. N., Borisovsky S. A., Tomakova R. A., Filist S. A. Nejrosetevye modeli segmentacii angiogramm glaznogo dna na osnove analiza RGB-kodov pikselej [Neural network models of segmentation of fundus angiograms based on the analysis of RGB pixel codes]. *Sistemnyj analiz i upravlenie v biomedicinskih sistemah = System Analysis and Management in Biomedical Systems*, 2010, vol. 9, no. 1, pp. 72–76.

4. Filist S. A., Dabagov A. R., Tomakova R. A., Malyutina I. A. Mnogoslojnye morfologicheskie operatory dlya segmentacii slozhnostrukturiruemyh rastrovyh polutonovyh izobrazhenij [Multilayer morphological operators for segmentation of complex structured raster halftone images]. *Izvestiya Yugo-Zapadnogo gosudarstvennogo universiteta. Seriya: Upravlenie, vychislitel'naya tekhnika, informatika. Medicinskoe priborostroenie = Proceedings of the Southwest State University. Series: Control, Computer Engineering, Information Science. Medical Instruments Engineering*, 2019, vol. 9, no. 3 (32), pp. 44–63.

5. Tomakova R. A., Filist S. A., Pykhtin A. I. Automatic Fluorography Segmentation Method Based on Histogram of Brightness Submission in sliding Window. *International Journal of Pharmacy and Technology*, 2017, vol. 9, no 1, pp. 28220–28228.

6. Tomakova R. A., Shevtsov A. N. Sposob postroeniya telekommunikacionnoj seti peredachi dannyh s borta vozdushnogo sudna na nazemnyj dispetcherskij punkt [Method of building a telecommunication data transmission network from an aircraft to a ground control

room]. *Izvestiya Yugo-Zapadnogo gosudarstvennogo universiteta. Seriya: Upravlenie, vychislitel'naya tekhnika, informatika. Medicinskoe priborostroenie = Proceedings of the Southwest State University. Series: Control, Computer Engineering, Information Science. Medical Instruments Engineering*, 2020, vol. 10, no. 1, pp. 157–173.

7. Shiyi Cao, Salvatore Di Girolamo, Torsten Hoefler. Accelerating Data Serialization / Deserialization Protocols with In-Network Compute. Workshop on Exascale MPI (ExaMPI). Dallas, TX, USA, EEE Publ., 2023, pp. 12–19.

8. Sayar Imen, Bartel Alexandre, Bodden Eric, Le Traon Yves. An In-depth Study of Java Deserialization Remote-Code Execution Exploits and Vulnerabilities. *ACM Transactions on Software Engineering and Methodology*, 2023, no. 1-31. https://doi.org/10.1145/3554732

9. Juan Antonio Mora-Castillo. Object serialization/deserialization and data transmission with JSON. *Revista Tecnología en Marcha*, 2016, vol. 29, is. 1, pp. 118–125. https://doi.org/10.18845/tm.v29il.2544

10. Huang B., Tang Y. Research on optimization of real-time efficient storage algorithm in data information serialization. *PLoS One*, 2021, no. 16(12), p. e0260697. https://doi.org/10.1371/journal.pone.0260697

11. Software Architecture: 15th European Conference, ECSA 2021; ed. by S. Biffl, E. Navarro, W. Löwe, M. Sirjani, R. Mirandola, D. Weyns. Sweden, Virtual Event Publ., 2012. 339 p.

12. Borisovsky S. A., Brezhneva A. N., Tomakova R. A. Nejrosetevye modeli s ierarhicheskim prostranstvom informativnyh priznakov dlya segmentacii plohostrukturirovannyh izobrazhenij [Neural network models with a hierarchical space of informative features for segmentation of poorly structured images]. *Biomedicinskaya radioelektronika = Biomedical Radioelectronics*, 2010, no. 2, pp. 49–53.

13. Freeman E., Robson E., Sierra K. Patterny proektirovaniya [Design patterns]. St. Petersburg, Peter Publ., 2011, pp. 203–229.

14. Martin R. Principy, shablony i metody gibkoj razrabotki na C # [Principles, patterns and methods of agile development in C#]. St. Petersburg, Peter Publ., 2019, pp. 39–49.

15. Varanasi B., Bartkov M. Spring REST. Building Java Microservices and Cloud Applications. Apress Berkeley, CA, 2022. 243 p.

16. Sarshfield S. Data Migration Best Practices: Strategies for Successful Data Migration Between Applications. Kindle, 2018. 169 p.

17. Newman S. Sozdanie mikroservisov [Creation of microservices]. St. Petersburg, Peter Publ., 2016. 304 p.

18. Mistrík I., Bahsoon R., Ali N., Heisel M., Maxim B. Software Architecture for Big Data and the Cloud. Elsevier Science, 2017.

19. Morris J. Data Migration Handbook: Practical Advice for Data Migration Projects. London, BCS The Chartered Institute for IT Publ., 2020, pp. 231–244.

20. Doolan D. Migraciya dannyh: prakticheskoe rukovodstvo po effektivnomu perenosu dannyh [Data Migration: A Practical Guide to Effective Data Migration]. Wimbledon, London, BCS Learning & Development Ltd Publ., 2020, pp. 301–312.

Информация об авторах / Information about the Authors

Томакова Римма Александровна,
доктор технических наук, профессор,
профессор кафедры программной инженерии,
Юго-Западный государственный университет,
г. Курск, Российская Федерация,
e-mail: rtomakova@mail.ru,
Researcher ID: O-6164-2015,
ORCID: 0000-0003-152-4714

Rimma A. Tomakova, Dr. of Sci. (Engineering),
Professor, Professor of the Department
of Software Engineering, Southwest State
University, Kursk, Russian Federation,
e-mail: rtomakova@mail.ru,
Researcher ID: O-6164-2015,
ORCID: 0000-0003-152-4714

Иванов Дмитрий Вадимович, магистрант,
Юго-Западный государственный университет,
г. Курск, Российская Федерация,
e-mail: asmadisel@yandex.ru,
ORCID: 0009-0004-5581-1641

Dmitry I. Vadimovich, Undergraduate,
Southwest State University,
Kursk, Russian Federation,
e-mail: asmadisel@yandex.ru,
ORCID: 0009-0004-5581-1641

Корсунский Никита Александрович,
аспирант, Юго-Западный государственный
университет, г. Курск, Российская Федерация,
e-mail: cor.nick2013@yandex.ru,
ORCID: 0009-0005-4606-5517

Nikita A. Korsunsky, Post-Graduate Student,
Southwest State University,
Kursk, Russian Federation,
e-mail: cor.nick2013@yandex.ru,
ORCID: 0009-0005-4606-5517