МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
Тернопільський національний економічний університет
Факультет комп'ютерних інформаційних технологій
Кафедра комп'ютерної інженерії

Бодров Олександр Миколайович

Модель текстових структур даних для Web-додатків / Text data structures model for Web applications

спеціальність:123 – Комп'ютерна інженерія
освітньо-професійна програма – Комп'ютерна інженерія

Випускна кваліфікаційна робота

Виконав студент групи КІм-21
О. М. Бодров
_____

Науковий керівник:
д.т.н., доцент, І. Р. Паздрій
_____

ТЕРНОПІЛЬ - 2019

РЕЗЮМЕ

Випускна кваліфікаційна робота на тему «Модель текстових структур даних для Web-додатків» зі спеціальності 123 «Комп’ютерна інженерія» написана обсягом 88 сторінок і містить 9 ілюстрацій, 2 додатки та 65 джерел за переліком посилань.

Метою роботи є розробка та проектування абстрактного операційного формату даних eJSON, а також короткої документації та специфікації основних принципів та особливостей формату даних.

Методи дослідження включають методи: теорію алгоритмів і систем, методи аналізу текстових даних, методи об’єктно-орієнтованого програмування, принципи та підходи до проектування текстових структур та форматів.

Розроблено абстрактний операційний формат даних eJSON та специфікацію, яка описує правила, принципи та положення формату.

Здійснено демонстраційну реалізацію розробленого формату для опису конфігураційних файлів для веб-додатку на основі платформи ASP.NET Core.

Можливими напрямками подальших досліджень є продовження робіт по аналізу структур даних та існуючих стандартів для розробки нових та,або покращення розробленого формату eJSON.

КЛЮЧОВІ СЛОВА: ФОРМАТ, ДАНІ, СТУРКТУРА, СПЕЦИФІКАЦІЯ.

RESUME

The Master's thesis on Design and Design of an «Text data structures model for Web applications», specializing in 123 Computer Engineering, has a total of 88 pages and contains 9 illustrations, 2 additions and 65 reference sources.

The purpose of the work is to develop and design an abstract eJSON operational data format, as well as brief documentation and specification of the basic principles and features of the data format.

Research methods include methods: theory of algorithms and systems, methods of textual data analysis, methods of object-oriented programming, principles and approaches to designing text structures and formats.

An abstract eJSON operational data format and specification describing the format's rules, principles, and regulations have been developed.

A demonstration implementation of the developed format for describing the configuration files for a web application based on the ASP.NET Core platform has been implemented.

Possible areas for further research are the continuation of work on the analysis of data structures and existing standards for the development of new and, or improvements to, the developed eJSON format.

KEYWORDS: FORMAT, DATA, STRUCTURE, SPECIFICATION.

ЗМІСТ

ВСТУП

Актуальність теми. В наш час правильно побудований та розроблений веб-додаток може створити сильний поштовх для розвитку бізнесу, стартапу чи власної ідеї.

Веб-додаток – клієнт-серверний додаток, в якому клієнт взаємодіє з веб-сервером за допомогою браузера. Логіка веб-додатки розподілена між сервером і клієнтом, зберігання даних здійснюється, переважно, на сервері, обмін інформацією відбувається по мережі. Одним з переваг такого підходу є той факт, що клієнти не залежать від конкретної операційної системи користувача, тому веб-додатки є крос-платформеними службами.

Веб-додаток здатен реалізувати складні обрахунки та процеси, а також диктувати власні правила обробки запитів та посилань. Для ефективного обміну даними між компонентами веб-додатку необхідно реалізувати структуру даних.

На даний момент існує безліч текстових форматів та структур даних, більшість з них має описану специфікацію та стандарт. Проте кожний з них має власні особливості та принципи. Тому обирати структуру необхідно під конкретні умови та задачу.

Мета роботи. Метою роботи є розробка та документація формату даних. Розроблюваний формат дозволить більш ефективно організувати передачу даних між компонентами системи. Для цього розроблено набір розширень, інструментів та принципів.

Об’єкт дослідження – процеси обміну даними між компонентами веб-системи в умовах функціонування в глобальній мережі на основі підходу «клієнт-сервер» або хмарних обчисленнях. Також досліджуються підходи до пост-транспортної серіалізації та де-серіалізації. Також досліджуються підходи до ефективного збереження структур даних.

Предмет дослідження – структури та формати даних, ціль яких – опис форматування та/або безпосередньо самих даних. Також їх зберігання та передача.

Методи дослідження. Семантичний аналіз принципів текстових структур даних, їх особливостей та методів їх обробки. Експериментальний аналіз процесу передачі даних за допомогою структур JSON, YAML, XML, MIME та інших.

Наукова новизна отриманих результатів.

Удосконалена структура «eJSON», побудована на основі формату даних «JSON» представляє собою абстрактний рівень, який існує поверх «JSON». eJSON компілюється в «чистий» JSON. Для опису та структуризації даних eJSON надає потужний набір інструментів, параметрів та принципів. Використання eJSON дозволяє описувати складні структури даних ефективно та зрозуміло для людей. Будь який документ типу JSON може бути трансльований в eJSON і навпаки.

Практичне значення одержаних результатів. Завдяки реалізованим механізмам та принципам, eJSON дозволяє зберігати та передавати дані більш ефективно, ніж при використанні JSON. За допомогою інструментів eJSON будь який документ JSON може бути трансльований в eJSON, що дозволить економити пам’ять. Також завдяки меншим об’ємам вихідних даних eJSON здатен підвищити ефективність передачі даних між компонентами веб-системи, що підвищує їх швидкість роботи.

Публікації та апробація результатів. Отримані результати апробовані в межах II науково-практичної конференції молодих вчених і студентів «Інтелектуальні комп’ютерні системи та мережі Тернопільського національного економічного університету та опубліковано дві тези доповіді по темі роботи [3, 4]:

Впровадження практичних результатів планується на підприємстві «Magnise».

В першому розділі випускної роботи розкриваються та описуються основні поняття, функції та ознаки. Проводиться аналіз основних систем та алгоритмів.

В другому розділі описані існуючі стандарти, формати та специфікації, які використовуються для вирішення поставленого завдання.

Третій розділ являє собою опис результату розробки специфікації абстрактного операційного формату даних eJSON.

1 АНАЛІЗ ТЕКСТОВИХ СТРУКТУР ДАНИХ

1.1 Аналіз предметної області

Текстовими даними як правило називаються послідовності з підмножини знаків, що включає тільки друковані знаки (літери, цифри, знаки пунктуації) і деякі керуючі знаки (прогалини, табуляції, переклади рядка). Існують методи (наприклад, UUENCODE або Base64), що дозволяють закодувати в текстовому форматі довільні дані будь-якого формату, що часто використовується для кодування бінарних даних.

Вимога до можливості розуміння вмісту людиною вносить додаткову надмірність в уявлення даних. Наприклад, число 123, для кодування якого досить одного 8-бітного байта, в текстовому вигляді кодується декількома цифровими символами – так, в десятковій системі числення для цього потрібно три знаки («123»), в двійковій – сім знаків («1111011»), в шістнадцятковій – два («7B»).

Текстовий формат не дозволяє використовувати команди форматування тексту, керувати атрибутами шрифтів, розмічати вміст [1].

Текстові дані можуть розділятися на рядки. На деяких платформах (в основному, в операційних системах сімейства UNIX) розбиття на рядки кодується одним керуючим знаком з кодом 10 в таблиці ASCII (найменування – Line Feed, LF), на інших (наприклад, в MS-DOS і Microsoft Windows) – парою керуючих знаків з кодами 13 та 10 (Carriage Return і Line Feed, CR / LF). У Mac OS (але не Mac OS X) розбиття кодується одним знаком з кодом 13.

Таке розбиття керуючим знаком або знаками продиктовано тим, як працювали друкарські машинки, через які здійснювався введення в деяких перших комп'ютерах – позиція введення там вказувалася становищем валика з папером, і для повороту валика і переходу до наступного рядка потрібно натискання однієї або двох клавіш або важелів.

Також, знаки розбиття рядків використовувалися для управління механічними принтерами (в якості яких могли виступати ті ж друкарські машинки, які використовуються і для введення) – знак LF викликав прокручування рулону з папером, а знак CR викликав повернення друкованої каретки (там, де вони були) в початок рядка. Звідси і назва знаків – англ. Line Feed (переклад рядка) і англ. Carriage Return (повернення каретки).

На деяких платформах розбивка на рядки робилося інакше – текст представлявся у вигляді послідовності записів фіксованої довжини, для чого більш короткі строки доповнювалися потрібною кількістю пробілів. Це відповідало поданням даних на перфокартах, які служили засобом введення і навіть зберігання даних, що мали фіксовану ширину (наприклад, 80 позицій – колонок).

Фактично, мови розмітки типу HTML, XAML, YAML і так далі являють собою текстову структуру даних.

Мова розмітки (тексту) в комп'ютерній термінології – набір символів або послідовностей, що вставляються в текст для передачі інформації про його виведення або будову. Належать класу комп'ютерних мов, хоча фактично мовами не являються. Текстовий документ, написаний з використанням мови розмітки, містить не тільки сам текст (як послідовність слів і знаків пунктуації), але і додаткову інформацію про різні його ділянках – наприклад, вказівку на заголовки, виділення, списки і т. Д. У більш складних випадках мова розмітки дозволяє вставляти в документ інтерактивні елементи і зміст інших документів.

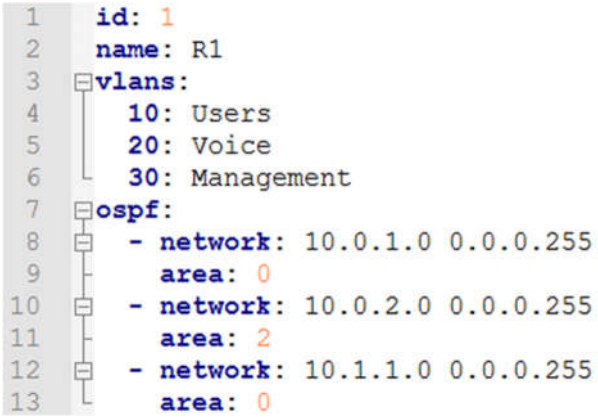

Рисунок 1.1 – Приклад опису даних за допомогою YAML

Розрізняють логічну і візуальну розмітки. У першому випадку мова йде тільки про те, яку роль відіграє дану ділянку документа в його загальній структурі (наприклад, «дана рядок є заголовком»). У другому – визначається, як саме буде відображатися цей елемент (наприклад, «цього рядка слід відображати жирним шрифтом»). Ідея мов розмітки полягає в тому, що візуальне відображення документа повинно автоматично виходити з логічної розмітки і не залежати від його безпосереднього змісту. Це спрощує автоматичну обробку документа і його відображення в різних умовах (наприклад, один і той же файл може по-різному відображатися на екрані комп'ютера, мобільного телефону і на друку, оскільки властивості цих пристроїв виведення істотно розрізняються). Однак це правило часто порушується: наприклад, створюючи документ в редакторі на зразок MS Word, користувач може виділяти заголовки жирним шрифтом, але ніде не вказувати, що цей рядок є заголовком.

Основна мета застосування текстових даних – «спільний знаменник», незалежність від окремих програм, що вимагають власного кодування або форматування і несумісних з іншими програмами. Текстові файли (файли в текстовому форматі) можуть бути відкриті, прочитані і відредаговані в будь-яких текстових редакторах, таких як MS-DOS Editor (англ.) (DOS), Блокнот (Windows), ed, vi і vim (UNIX, Linux), SimpleText (англ.), TextEdit (Mac OS X)

і т. п. Інші програми також як правило вміють читати і імпортувати текстові дані. Переглянути текстові файли можна також вбудованими командами (type в DOS і Windows) і утилітами (cat в Unix).

Текстовий формат часто використовуються для представлення даних, які самі не є чисто текстовими. У цьому випадку інші формати даних «надбудовуються» над простим текстом, для чого їх керуючі конструкції виражаються за допомогою друкованих слів і знаків пунктуації. Це забезпечує зручність роботи з даними на двох рівнях – наприклад, дані HTML і XML можна переглядати і редагувати з показом форматування в режимі WYSIWYG, а можна їх відкрити в звичайному текстовому редакторі і мати доступ до всіх тонкощів мови розмітки. При зберіганні даних в «довічним» вигляді (як це робиться, наприклад, в Microsoft Word ранніх версій) з ними нерідко можна працювати в інших програмах (через недоступність інформації про структуру формату) або навіть в різних версіях однієї і тієї ж програми.

У більшості мов програмування передбачається використання текстового формату для вихідного коду програм. Крім іншого, це дозволяє застосовувати до вихідного коду різноманітні утиліти для перетворень, оформлення, пошуку, статистики, аналізу та т. п.

У файлах конфігурації багатьох програм застосовується текстовий формат, навіть якщо там представлені числа і виконавчі перемикачі (так / ні). Це дещо ускладнює програми через необхідність перетворення текстових даних у внутрішній формат і назад, але з'являється можливість правити конфігурацію вручну, без використання засобів налаштування самої програми.

Скрутним є вказівка на якусь певну частину тексту, що зберігається в форматі текстових даних. Як покажчиків можуть використовуватися номери рядків або номера символів.

Структура даних (англ. Data structure) – програмна одиниця, що дозволяє зберігати і обробляти безліч однотипних і / або логічно пов'язаних даних в

обчислювальній техніці. Для додавання, пошуку, зміни і видалення даних структура даних надає певний набір функцій, з яких складається інтерфейс.

Термін «структура даних» може мати кілька близьких, але тим не менше різних значень:

- абстрактний тип даних;
- реалізація будь-якого абстрактного типу даних;
- примірник типу даних, наприклад, конкретний список.

В контексті функціонального програмування – унікальна одиниця (англ. Unique identity), що зберігається при змінах. Про неї неформально говорять як про одну структуру даних, незважаючи на можливу наявність різних версій.

Структури даних формуються за допомогою типів даних, посилань і операцій над ними в обраною мовою програмування.

Різні види структур даних підходять для різних додатків; деякі з них мають вузьку спеціалізацію для певних завдань. Наприклад, B-дерева зазвичай підходять для створення баз даних, в той час як хеш-таблиці використовуються повсюдно для створення різного роду словників, наприклад, для відображення доменних імен в інтернет-адреси комп'ютерів.

При розробці програмного забезпечення складність реалізації і якість роботи програм істотно залежить від правильного вибору структур даних. Це розуміння дало початок формальним методам розробки та мов програмування, в яких саме структури даних, а не алгоритми, є наріжним архітектури програмного засобу. Велика частина таких мов володіє певним типом модульності, що дозволяє структурам даних безпечно пере-використати в різних додатках. Об'єктно-орієнтовані мови, такі як Java, C # і C ++, є прикладами такого підходу.

Багато класичні структури даних представлені в стандартних бібліотеках мов програмування або безпосередньо вбудовані в мови програмування. Наприклад, структура даних хеш-таблиця вбудована в мови

програмування Lua, Perl, Python, Ruby, Tcl і ін. Широко використовується стандартна бібліотека шаблонів (STL) мови C ++.

Фундаментальними будівельними блоками для більшої частини структур даних є масиви, записи (struct в Ci і record в Паскалі), розмічені об'єднання (union в Ci) і посилання. Наприклад, двох-зв'язний список може бути побудований за допомогою записів і посилань, де кожен запис (вузол) буде зберігати дані і посилання на «лівий» і «правий» вузли.

XML (EXtensible Markup Language) – розширювана мова розмітки. Рекомендований Консорціумом Всесвітньої павутини (W3C). Специфікація XML описує XML-документи і частково описує поведінку XML-процесорів (програм, які читають XML-документи і забезпечують доступ до їх вмісту). XML розроблявся як мова з простим формальним синтаксисом, зручний для створення і обробки документів програмами і одночасно зручний для читання і створення документів людиною, з підкресленням націленості на використання в Інтернеті. Мова називається розширюваним, оскільки ним не фіксується розмітку, яка використовується в документах: розробник вільний створити розмітку відповідно до потреб конкретної області, будучи обмеженим лише синтаксичними правилами мови. Розширення XML – це конкретна граматика, створена на базі XML і представлена словником тегів і їх атрибутів, а також набором правил, що визначають які атрибути і елементи можуть входити до складу інших елементів. Поєднання простого формального синтаксису, зручності для людини, розширюваності, а також базування на кодуваннях Юнікод для подання змісту документів привело до широкого використання як власне XML, так і безлічі похідних спеціалізованих мов на базі XML в найрізноманітніших програмних засобах.

XML – мова розмітки, іншими словами, засіб опису документа. Саме в ніші документів, текстів, де частка різнотипних символьних даних велика, а частка розмітки мала – XML успішний. З іншого боку, обмін даними у відкритих системах не зводиться до обміну документами. Надмірність

розмітки XML (а в цілях розробки мови прямо вказано, що лаконічність не є пріоритетом проекту) позначається в ситуаціях, коли дані не вписуються в традиційну модель документа. Стрічка новин, наприклад, що оформляється з використанням синтаксису XML (формати RSS, Atom), являє собою не документ в традиційному розумінні, а потік однотипних міні-документів – багатослівна і надлишкова розмітка в цьому випадку становить істотну частину переданих даних.

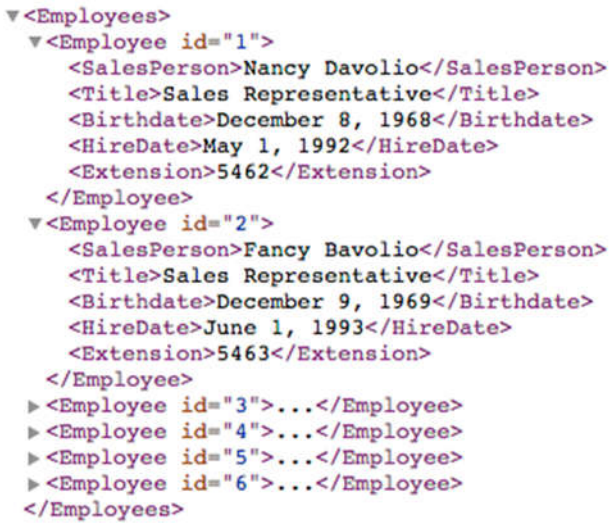

Рисунок 1.2 – Приклад опису даних за допомогою XML

W3C стурбований ефективністю застосування XML, і відповідні робочі групи займаються цією проблемою (до початку 2013 року нормативні документи не розроблені).

Інша ситуація, коли формати XML можуть виявитися не кращим рішенням – робота з даними з простою структурою і невеликим за обсягом змістом полів даних. У цьому випадку частка розмітки в загальному обсязі велика, а програмна обробка XML може виявитися невиправдано витратною, в порівнянні з роботою з даними більш простої структури. У цій області розробники розглядають засоби, спочатку орієнтовані на дані, такі як INI, YAML, JSON.

1.2 Аналіз об’єкту

YAML (акронім англ. «Yet Another Markup Language» – «Ще одна мова розмітки», пізніше – рекурсивний акронім англ. «YAML Is not Markup Language» – «YAML – не мова розмітки») – «дружній» формат серіалізації даних, концептуально близький до мов розмітки, але орієнтований на зручність введення-виведення типових структур даних багатьох мов програмування.

У трактуванні назви відображена історія розвитку: на ранніх етапах YAML розшифровувався як Yet Another Markup Language («Ще одна мова розмітки») і навіть позиціонувався як конкурент XML, але пізніше був перейменований з метою акцентувати увагу на даних, а не на розмітці документів.

У певному проекті потрібно зберігати конфігурацію, що описує відображення (англ. Bindings) IRC-команд на функції, за допомогою регулярних виразів.

Ось вихідна конфігурація, представлена у вигляді таблиці:

ircEvent	method	regexp
PRIVMSG	newUri	"^http://.*"
PRIVMSG	deleteUri	"^delete.*"
PRIVMSG	randomUri	"^random.*"

Рисунок 1.3 – Таблиця вихідної конфігурації

У YAML ця конфігурація може бути представлена наступним чином:

```
bindings:
   - ircEvent: PRIVMSG
     method: newUri
     regexp: '^ http: //.*'
   - ircEvent: PRIVMSG
     method: deleteUri
     regexp: '^ delete. *'
   - ircEvent: PRIVMSG
     method: randomUri
     regexp: '^ random. *'
```

Порівняння з XML

Для порівняння, в XML-представлення дана конфігурація може бути представлена наступним чином:

```
<Bindings>
    <Binding>
        <IrcEvent> PRIVMSG </ ircEvent>
        <Method> newUri </ method>
        <Regexp> ^ http: //.* </ regexp>
    </ Binding>
    <Binding>
        <IrcEvent> PRIVMSG </ ircEvent>
        <Method> deleteUri </ method>
        <Regexp> ^ delete. * </ Regexp>
    </ Binding>
    <Binding>
        <IrcEvent> PRIVMSG </ ircEvent>
        <Method> randomUri </ method>
        <Regexp> ^ random. * </ Regexp>
    </ Binding>
</ Bindings>
```

Альтернативний варіант, який використовує атрибути:

```
<Bindings>
        <Binding ircEvent = "PRIVMSG" method = "newUri" regexp =
"^ http: //.*" />
        <Binding ircEvent = "PRIVMSG" method = "deleteUri"
regexp = "^ delete. *" />
        <Binding ircEvent = "PRIVMSG" method = "randomUri"
regexp = "^ random. *" />
</ Bindings>
```

Говорячи про відмінності YAML від XML, також слід зазначити, що вкладені XML-елементи можуть використовуватися для відображення довільних структур, а YAML ближчий до відображення типових моделей даних з Ruby, Perl, Python, Java, дозволяючи описувати вільні сполучення послідовностей, зіставлень і скалярних типів – тобто ближче до реальних структурам даних мов програмування, і не вимагає різних угод про DOM-відображення структур даних на документи і назад, як потрібно в XML.

Порівняння JSON та XML

В процесі розробки додатків з клієнт-серверною архітектурою неминуче виникає проблема вибору формату обміну даними між клієнтськими і серверним модулями. Вирішення цієї проблеми може мати значний вплив як на роботу програми, так і на трудомісткість подальшої модернізації. Час відгуку, обсяг переданої інформації по каналу зв'язку, розширюваність і портіруемость, необхідні ресурси і інші параметри можуть залежати від формату обміну даними.

Метою дослідження є визначення критеріїв для проведення порівняльного аналізу найбільш популярних форматів обміну даними і вибір з них найбільш відповідає вимогам до побудови нових розподілених інформаційних систем.

В даний час існує значна кількість різних форматів, рекомендованих в літературі для використання в розподілених інформаційних системах. Найчастіше в співтоваристві розробників, перевагу віддають одному з трьох найбільш використовуваних форматів обміну даними: XML, JSON, YAML.

XML (Extensible Markup Language) – простий, дуже гнучкий текстовий формат, який є підмножиною SGML (ISO 8879) [3], який дозволяє визначати власні теги і атрибути. Мова називається розширюваним, оскільки ним не фіксується розмітку, яка використовується в документах: розробник вільний створити її відповідно до особливостей конкретної предметної області, будучи

обмеженим лише синтаксичними правилами мови [4]. Можливість створення власних тегів робить XML універсальним.

JSON (Java Script Object Notation) – являє собою полегшений формат обміну даними між комп'ютерами [5]. Відповідно до визначення стандарту сценарного мови програмування ECMA (Європейської асоціації виробників комп'ютерів), він є похідним від літералів Java Script. JSON більш компактний, ніж XML, його конструкції легше аналізуються засобами Java Script, для якого JSON є внутрішнім використовуваним типом даних. Основна сфера застосування JSON – програмування web-додатків, де він служить альтернативою XML.

YAML – формат серіалізації даних, концептуально близький до мов розмітки, але орієнтований на зручність введення-виведення типових структур даних багатьох мов програмування. В даний час акронім YAML транслюється як «YAML Is not Markup Language» («YAML – не мова розмітки»). У назві відображено історію розвитку: на ранніх етапах акронім був абревіатуру виразу «Yet Another Markup Language» («Ще одна мова розмітки») і навіть розглядався як конкурент XML, але пізніше був перейменований, щоб акцентувати увагу на даних, а не на розмітці документів.

В даний час не існує сформульованих критеріїв, завдяки яким можна було б порівняти формати обміну даними в додатках з клієнт-серверною архітектурою. Однак на різних ресурсах, присвячених IT-тематики, професійними розробниками та архітекторами інформаційних систем не раз робилися спроби такі критерії рекомендувати. Варто відзначити, що, незважаючи на значне число характерних рис, рекомендованих в якості критеріїв порівняння, не всі вони представляють теоретичну або практичну користь. Нижче представлені ті критерії порівняльного аналізу, які в даний час є найбільш важливими при прийнятті рішення в процесі розробки клієнт-серверного додатка.

- може читатися людиною – припускає просту і зручну розмітку переданих даних. При цьому мова повинна мати незначну кількість символів-роздільників (дужки, лапки і т.д.);

- простота серіалізації – перетворення об'єкта (даних) в потік байтів для подальшого зберігання або передачі по каналу зв'язку, в пам'ять або файл;

- простота десеріалізації – перетворення потоку байтів в об'єкт даних;

- можливість перевірки формату вхідних даних – наявність в форматі обміну даними внутрішнього мови опису структури документа (JSON-Schema, XML-Schema), необхідного для здійснення попередньої перевірки на відповідність приходять даних, наприклад, з боку клієнта;

- ефективність стиснення даних – включає швидкість виконання алгоритму компресії і коефіцієнт стиснення;

- поширеність – наявність великої кількості розробників, що використовують той чи інший формат обміну даними;

- динаміка розвитку, яка характеризується швидкістю популяризації та розвитку.

Як функціонально, так і синтаксично JSON є підмножиною мови YAML. Зокрема, специфікація YAML 1.2 вказує, що «будь-який файл у форматі JSON є коректним файлом у форматі YAML». Найбільш поширений парсер YAML здатний обробляти і JSON. Специфікація YAML до версії 1.2 в повному обсязі покривала JSON, в першу чергу через відсутність рідної підтримки UTF-32 в YAML, а також вимоги пробілу після роздільник-коми; крім того, специфікація JSON включала коментарі в стилі / * * /.

Найбільш важливою відмінністю YAML є набір розширень синтаксису, яким немає аналогів в JSON:

- підтримка реляційних даних: в YAML-документі можна посилатися на якір, який зустрівся раніше в файлі / потоці; таким чином можна висловити рекурсивні структури;

- підтримка розширюваних типів даних крім примітивів: рядків, чисел, логічних значень і т. д.;

- підтримка блочного синтаксису з відступами; він дозволяє описати структуровані дані без використання зайвих символів: дужок, лапок і т. д.

Порівняння трьох розглянутих форматів обміну інформацією буде виконано по перерахованим вище критеріям відповідно до даних, опублікованих не раніше 2013 року. Для кількісної оцінки відповідності того чи іншого критерію використовується п'ятибальна шкала як одна з найпростіших і розповсюджених в системах оцінювання. Потім підраховується середній бал оцінки кожного формату. Слід зазначити, що використовуваний метод порівняльного аналізу є суб'єктивним.

Зручність читання формату для обміну даними викликає суперечки серед розробників програмного забезпечення, тому що, на думку багатьох, є занадто суб'єктивним. Деякі фахівці стверджують, що це один з найважливіших критеріїв, і наводять докази того, що один формат зручніше для читання, ніж інший. Так, в результаті навіть побіжного перегляду інтернет-ресурсів, можна зробити висновок про незручність (більш скрутному сприйнятті) XML.

Для розгляду другої характеристики необхідно навести приклади перетворення об'єкта даних (XML, JSON і YAML) в серверному модулі інформаційної системи. В якості мови програмування для ілюстрації обраний C# як один з найбільш популярних мов з великою кількістю додаткових бібліотек і підтримкою мільйонів розробників програмного забезпечення.

Основною перевагою YAML, як запевняють його творців, є зручність читання. Однак твердження досить спірно, враховуючи наявність великої кількості можливостей інших форматів і досвід роботи з ними. Надалі передбачається використання отриманої методики критеріїв при розробці різних розподілених інформаційних систем.

Представлені критерії допоможуть визначитися розробникам програмного забезпечення в необхідності впровадження будь-якого формату обміну даними. Так як всі представлені формати досить активно розвиваються, то пропонується використання не результатів порівняльного аналізу, а саме використання критеріїв для визначення найкращого формату на поточний період.

Приклад серіалізації об'єкта XML:

```
XmlSerializer x = new XmlSerializer(typeof(Students));
TextWriter writer = new StreamWriter(filename);
x.Serialize(writer, Emps);
```

Рисунок 1.4 – XML

Приклад серіалізації об'кта JSON:

```
List<Student> students = new List<Student>();
JavaScriptSerializerserializer = JavaScriptSerializer();
string json = serializer.Serialize(students);
```

Рисунок 1.5 – JSON

Приклад серіалізації об'єкта YAML:

```
List<Student> students = new List<Student>();
varserializer = new YamlSerializer();
string yaml = serializer.Serialize(students);
```

Рисунок 1.6 – YAML

По наведених прикладах можна зробити висновок, що в цілому все три формату досить зручні і прості для серіалізації об'єктів даних. Зауважимо, що JSON і YAML мають однакову структуру програмного коду серіалізації об'єкта, відрізняючись лише найменуванням власних типів. Два зазначених формату схожі один на одного і відрізняються лише синтаксичними роздільниками.

Нижче наведені приклади десеріалізації об'єкта даних для пояснення прийняття рішення по третьому критерію.

Наявність можливості перевірки формату вхідних даних має розглядатися як для серверної, так і для клієнтської частин інформаційної системи. При використанні XML і JSON є можливість здійснити перевірку даних через мову опису структури документа – schema. У YAML така операція вже впроваджується, проте поки не отримала широкого поширення і підтримки серед розробників. Однак в разі клієнтського модуля, як правило, складніше реалізувати виявлення помилок в даних всіх трьох форматів.

Реалізація алгоритмів стиснення даних не є поширеною завданням при передачі даних в розглянутих форматах. Однак в деяких додатках компресія даних дійсно необхідна, наприклад в мережах мобільних пристроїв і рухомих об'єктів з метою економії енергії та зменшення трафіку. Небагато компаній і розробники проводили дослідження і тестування з метою порівняльного аналізу. За результатами тестування бенчмарка, JSON виявився більш компактним, а також показав кращу ефективність стиснення в порівнянні з XML.

При використанні більш складних і нестандартних алгоритмів, що збільшують трудомісткість процесу розробки, можна домогтися гарних показників і у XML. Деякі з ефективних методів стиснення даних в XML форматі описані в технічних матеріалах на сайті IBM. Слід зазначити відсутність опублікованих результатів тестування алгоритмів стиснення для формату YAML. Можна припустити, що у дослідників поки не виникло інтересу до даного питання в зв'язку зі специфічною областю застосування цього формату.

Поширеність і динаміка розвитку форматів обміну даними не менш важливі для прийняття рішення щодо подальшого їх використання в розподілених інформаційних системах, оскільки від популярності будь-якої технології залежить її підтримка спільнотою розробників програмного

забезпечення і розвиток в майбутньому. Результати проведеного дослідження, проведеного в кінці 2013 року, показали значно зростаючу популярність JSON в порівнянні з XML. За YAML інформація відсутня.

За результатами проведеного дослідження, відбитим в таблиці, можна зробити висновок про те, що, незважаючи на значну кількість web-ресурсів та сервісів, документації і бібліотек, які застосовують в якості формату обміну даними XML, при побудові нових розподілених інформаційних систем слід розглянути використання JSON. В даний час в професійному співтоваристві розробників програмного забезпечення досить часто наголошують на тому, що цей формат є швидко розвиваються і простим для реалізації обміну даними в системах. Зауважимо, що формат обміну даними YAML був запропонований порівняно недавно і тільки починає свій шлях розвитку, але вже зараз становить серйозну конкуренцію аналогам за деякими характеристиками. Основною перевагою YAML, як запевняють його творців, є зручність читання. Однак твердження досить спірно, враховуючи наявність великої кількості можливостей інших форматів і досвід роботи з ними.

Надалі передбачається використання отриманої методики критеріїв при розробці різних розподілених інформаційних систем. Представлені критерії допоможуть визначитися розробникам програмного забезпечення в необхідності впровадження будь-якого формату обміну даними. Так як всі представлені формати досить активно розвиваються, то пропонується використання не результатів порівняльного аналізу, а саме використання критеріїв для визначення найкращого формату на поточний період.

Плюси:

- простий синтаксис, який призводить до зменшення накладних витрат «розмітки» в порівнянні з XML;
- проста у використанні з JavaScript, оскільки розмітка являє собою підмножина буквеної нотації об'єктів JS і має ті ж основні типи даних, що і JavaScript;

- схема JSON для опису і типу даних і перевірки структури;
- jsonPath для добування інформації в глибоко вкладених структурах.

Мінуси:

- простий синтаксис підтримується тільки декількома різними типами даних.

XML. Плюси:

- узагальнена розмітка; можна створювати «діалекти» для будь-яких цілей;
- XML-схема для типу даних, перевірка структури. Робить також можливим створення нових типів даних;
- XSLT для перетворення в різні формати виводу;
- XPath / XQuery для добування інформації в глибоко вкладених структурах;
- вбудована підтримка просторів імен.

Мінуси:

- багато супровідної розмітки в порівнянні з JSON (призводить до більшої кількості даних за той же обсяг інформації).

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
            "employees" :  [
            {
                  "name" : "James Kirk",
                  "age" : 40,
            },
            {
                  "name" : "Jean-Luc Picard",
                  "age" : 45,
            },
            {
                  "name" : "Wesley Crusher",
                  "age" : 27,
            }
                              ]
      }
}
```

Рисунок 1.7 – Порівняння розмітки структур

XML (розширювана мова розмітки) часто використовується XHR, тому що це стандартна мова мовлення, що може використовуватися будь-якою мовою програмування і підтримується як на стороні сервера, так і на стороні клієнта, тому це найбільш гнучке рішення. XML можна розділити на кілька частин, щоб зазначена група могла розвинути частина програми, не зачіпаючи інші частини. Формат XML також може бути визначений XML-DTD або XML-схемою (XSL) і може бути протестований.

JSON – формат обміну даними, який стає більш популярним в якості можливого формату JavaScript-додатків. В основному це масив позначень об'єктів. JSON має дуже простий синтаксис, тому його легко впізнати. А також підтримка JavaScript для синтаксичного аналізу JSON з функцією eval. З іншого боку, функція eval має негативи. Наприклад, програма може дуже повільно аналізувати JSON, і через безпеки eval може бути дуже ризикованим.

Це не означає, що JSON не дуже хороший, просто потрібно бути більш обережним.

XML найкраще підходить для великих веб-сайтів, наприклад, для торгових сайтів або чогось подібного. XML може бути більш безпечним і зрозумілим. Можна створити базову структуру даних і схему, щоб легко протестувати корекцію і легко розділити її на частини.

Важлива річ у JSON полягає в тому, щоб забезпечити передачу даних зашифрованими з міркувань безпеки. Безсумнівно, JSON набагато швидше, ніж XML. Я бачив, що XML займає 100 мс, де JSON займав лише 60 мс. Дані JSON легко маніпулювати.

XML – мова розмітки, іншими словами, засіб опису документа. Саме в ніші документів, текстів, де частка різнотипних символьних даних велика, а частка розмітки мала – XML успішний. З іншого боку, обмін даними у відкритих системах не зводиться до обміну документами. Надмірність розмітки XML (а в цілях розробки мови прямо вказано, що лаконічність не є пріоритетом проекту) позначається в ситуаціях, коли дані не вписуються в традиційну модель документа. Стрічка новин, наприклад, що оформляється з використанням синтаксису XML (формати RSS, Atom), являє собою не документ в традиційному розумінні, а потік однотипних міні-документів – багатослівна і надлишкова розмітка в цьому випадку становить істотну частину переданих даних.

W3C стурбований ефективністю застосування XML, і відповідні робочі групи займаються цією проблемою (до початку 2013 року нормативні документи не розроблені).

Інша ситуація, коли формати XML можуть виявитися не кращим рішенням – робота з даними з простою структурою і невеликим за обсягом змістом полів даних. У цьому випадку частка розмітки в загальному обсязі велика, а програмна обробка XML може виявитися невиправдано витратною, в порівнянні з роботою з даними більш простої структури. У цій області

розробники розглядають засоби, спочатку орієнтовані на дані, такі як INI, YAML, JSON.

MIME – стандарт, що описує передачу різних типів даних по електронній пошті, а також, в загальному випадку, специфікація для кодування інформації і форматування повідомлень таким чином, щоб їх можна було пересилати через Інтернет.

MIME визначає механізми для передачі різного роду інформації всередині текстових даних (зокрема, за допомогою електронної пошти), а саме: текст на мовах, для яких використовуються кодування, відмінні від ASCII, і нетекстові дані, такі, як картинки, музика, фільми і програми. MIME є також фундаментальним компонентом комунікаційних протоколів, таких як HTTP, яким потрібно, щоб дані передавалися в контексті повідомлень, подібних e-mail, навіть якщо дані реально не є e-mail.

Основний формат електронних повідомлень визначено в RFC 5322, який є оновленою версією RFC 2822 (який, в свою чергу, є оновленою версією RFC 822). Ці стандарти визначають схожі формати для текстових e-mail-заголовків і вмісту і правил, що відносяться до общеіспользуемих полях, таким як To :, Subject :, From: і Date :. MIME визначає набір e-mail-заголовків для визначення додаткових атрибутів повідомлення, включаючи тип контенту, і визначає безліч кодувань, які можуть бути використані для подання 8-бітних бінарних даних за допомогою символів з 7-бітного ASCII.

1.3 Постановка задачі

Основною метою є розробка власного унікального текстового формату даних, який відрізнятиметься від описаних форматів зручністю, економією та інформативністю. Для досягнення поставленої мети необхідно провести аналіз

принципів функціонування форматів даних, проаналізувати існуючі рішення, розробити власний абстрактний текстовий формат та розробити мінімальну специфікацію.

2 АНАЛІЗ ІСНУЮЧИХ ТЕКСТОВИХ СТРУКТУР

2.1 Аналіз мови розмітки YAML

YAML (акронім англ. «Yet Another Markup Language» – «Ще одна мова розмітки», пізніше – рекурсивний акронім англ. «YAML Is not Markup Language» – «YAML – не мова розмітки») – «дружній» формат серіалізації даних, концептуально близький до мов розмітки, але орієнтований на зручність введення-виведення типових структур даних багатьох мов програмування.

У трактуванні назви відображена історія розвитку: на ранніх етапах YAML розшифровувався як Yet Another Markup Language («Ще одна мова розмітки») і навіть позиціонувався як конкурент XML, але пізніше був перейменований з метою акцентувати увагу на даних, а не на розмітці документів.

Відповідно до цілей, озвучених Кларком Евансом (англ. Clark Evans), YAML 1.0 покликаний:

- бути легко зрозумілим людині;
- підтримувати структури даних, рідні для мов програмування;
- бути стерпним між мовами програмування;
- використовувати цільну модель даних для підтримки звичайного інструментарію;
- підтримувати потокову обробку;
- бути виразним і розширюваним;
- бути легким у реалізації та використанні.

До поточної редакції YAML (1.2) в ці цілі були внесені деякі зміни:

- пункти 2 і 3 помінялися місцями;
- пункт 5 був замінений на «YAML підтримує обробку в один прохід».

Приклад розмітки YAML:

```yaml
---
key: value
map:
    key1: "foo:bar"
    key2: value2
list:
  - element1
  - element2
# This is a comment
listOfMaps:
  - key1: value1a
    key2: value1b
  - key1: value2a
    key2: value2b
---
```

```
some
other
listing
```

Рисунок 2.1 – Приклад використання мови YAML

Порівняння з XML

Для порівняння, в XML-представлення дана конфігурація може бути представлена наступним чином:

```xml
<bindings>
    <binding>
        <ircEvent>PRIVMSG</ircEvent>
        <method>newUri</method>
        <regexp>^http://.*</regexp>
    </binding>
    <binding>
        <ircEvent>PRIVMSG</ircEvent>
        <method>deleteUri</method>
        <regexp>^delete.*</regexp>
    </binding>
    <binding>
        <ircEvent>PRIVMSG</ircEvent>
        <method>randomUri</method>
        <regexp>^random.*</regexp>
    </binding>
```

```
</bindings>
```

Альтернативний варіант, який використовує атрибути:

```
<bindings>
        <binding        ircEvent="PRIVMSG"        method="newUri"
regexp="^http://.*" />
        <binding      ircEvent="PRIVMSG"      method="deleteUri"
regexp="^delete.*" />
        <binding      ircEvent="PRIVMSG"      method="randomUri"
regexp="^random.*" />
    </bindings>
```

Говорячи про відмінності YAML від XML, також слід зазначити, що вкладені XML-елементи можуть використовуватися для відображення довільних структур, а YAML ближчий до відображення типових моделей даних з Ruby, Perl, Python, Java, дозволяючи описувати вільні сполучення послідовностей, зіставлень і скалярних типів – тобто ближче до реальних структурам даних мов програмування, і не вимагає різних угод про DOM-відображення структур даних на документи і назад, як потрібно в XML.

Невеликий список основних елементів YAML:

- потоки YAML використовують друковані Unicode-символи, як UTF-8, так і UTF-16;

- відступи з пробілів (символи табуляції не допускаються) використовуються для позначення структури;

- коментарі починаються з символу "решітки" (#), можуть починатися в будь-якому місці рядка і тривають до кінця рядка;

- списки позначаються початковим дефісом (-) з одним членом списку на рядок, або члени списку полягають в квадратні дужки ([]) і розділяються комою і пропуском (,);

- асоціативні масиви представлені двокрапкою з пробілом (:) у вигляді ключ: значення, по одній парі ключ-значення на рядок, або у вигляді пар, ув'язнених у фігурні дужки і розділених комою і пропуском (,);

- ключ в асоціативному масиві може мати в якості префікса знак питання (?), що дозволяє вказати складний ключ, наприклад представлений у вигляді списку;

- рядки записуються без лапок, однак можуть бути укладені в одиночні або подвійні лапки;

- всередині подвійних лапок можуть бути використані екрановані символи в С-стилі, що починаються з зворотного слеша (\);

- YAML дозволяє задавати підстановки за допомогою якорів & і алиасов (*). Приклад:

```
aliases:  #последовательность настроек
- &myAlias1
datakey: dataval 1
moredata: morevals 1
- &myAlias2
datakey: dataval 2
moredata: morevals 2
config:
- *myAlias1   # *myAlias1 после парсинга будет заменен на
[{"datakey": "dataval 1", "moredata": "morevals 1"}]
```

- явне завдання типу оформляється шляхом '!! [вказівку типу]'. Приклад, !! str 100 після парсинга видасть значення "100";

- значення типу Дата / Час задаються в форматі YYYY-MM-DD або YYYY-MM-DD HH: MM: SS. Якщо необхідно задати дату, як рядок, потрібно укладати її в лапки ("2012-12-21").

2.2 Аналіз формату даних JSON

JSON (англ. JavaScript Object Notation, зазвичай вимовляється як / dʒeɪsən / JAY-sən) – текстовий формат обміну даними, заснований на JavaScript. Як і

багато інших текстові формати, JSON легко читається людьми. Формат JSON був розроблений Дугласом Крокфорд.

Незважаючи на походження від JavaScript (точніше, від підмножини мови стандарту ECMA-262 1999 роки), формат вважається незалежним від мови і може використовуватися практично з будь-якою мовою програмування. Для багатьох мов існує готовий код для створення і обробки даних в форматі JSON.

За рахунок своєї лаконічності в порівнянні з XML, формат JSON може бути більш підходящим для серіалізації складних структур. Якщо говорити про веб-додатках, в такому ключі він доречний в задачах обміну даними як між оглядачем і сервером (AJAX), так і між самими серверами (програмні HTTP-сполучення).

Оскільки формат JSON є підмножиною синтаксису мови JavaScript, то він може бути швидко де-серіалізований вбудованою функцією eval(). Крім того, можлива вставка цілком працездатних JavaScript-функцій. У мові PHP, починаючи з версії 5.2.0, підтримка JSON включена в ядро у вигляді функцій json_decode () і json_encode (), які самі перетворять типи даних JSON в відповідні типи PHP і навпаки.

JSON-текст являє собою (в закодованому вигляді) одну з двох структур:

- набір пар ключ: значення. У різних мовах це реалізовано як запис, структура, словник, хеш-таблиця, список з ключем або асоціативний масив. Ключем може бути тільки рядок (чутливі до регістру: імена з буквами в різних регістрах вважаються різними), значенням – будь-яка форма;
- упорядкований набір значень. У багатьох мовах це реалізовано як масив, вектор, список або послідовність. Це універсальні структури даних: як правило, будь-який сучасний мову програмування підтримує їх в тій чи іншій формі. Вони лягли в основу JSON, так як він використовується для обміну даними між різними мовами програмування.

Як значення в JSON можуть бути використані:

- запис – це неврегульована безліч пар ключ: значення, укладену в фігурні дужки «{}». Ключ описується рядком, між ним і значенням стоїть символ «:». Пари ключ-значення відокремлюються один від одного комами;

- масив (одновимірний) – це впорядкована множина значень. Масив полягає в квадратні дужки «[]». Значення розділяються комами;

- число;

- літерали true, false і null;

- рядок – це впорядкована множина з нуля або більше символів юнікоду, укладену в подвійні лапки. Символи можуть бути вказані з використанням escape-послідовностей, що починаються з зворотної косої межі «\» (підтримуються варіанти \ ', \ ", \\, \ /, \ t, \ n, \ r, \ f і \ b), або записані шістнадцятковим кодом в кодуванні Unicode у вигляді \ uFFFF.

Рядок дуже схожий на однойменний тип даних в мовах C і Java. Число теж дуже схоже на C- або Java-число, за винятком того, що використовується тільки десятковий формат. Прогалини можуть бути вставлені між будь-якими двома синтаксичними елементами.

Наступний приклад показує JSON-уявлення даних про об'єкт, що описує людину. В даних є строкові поля імені і прізвища, інформація описує адреса, і масив, що містить список телефонів. Як видно з прикладу, значення може являти собою вкладену структуру.

Приклад використання формату JSON:

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   "address": {
       "streetAddress": "Московское ш., 101, кв.101",
       "city": "Ленинград",
       "postalCode": "101101"
   },
   "phoneNumbers": [
       "812 123-1234",
       "916 123-4567"
   ]
```

```
}
```

Мовою XML подібна структура виглядала б приблизно так:

```
<person>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <address>
    <streetAddress>Московское ш., 101, кв.101</streetAddress>
    <city>Ленинград</city>
    <postalCode>101101</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

або так:

```
<person firstName="Иван" lastName="Иванов">
  <address streetAddress="Московское ш., 101, кв.101" city="Ленинград" postalCode="101101" />
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

Як функціонально, так і синтаксично JSON є підмножиною мови YAML. Зокрема, специфікація YAML 1.2 вказує, що «будь-який файл у форматі JSON є коректним файлом у форматі YAML». Найбільш поширений парсер YAML здатний обробляти і JSON. Специфікація YAML до версії 1.2 в повному обсязі покривала JSON, в першу чергу через відсутність рідної підтримки UTF-32 в YAML, а також вимоги пробілу після роздільник-коми; крім того, специфікація JSON включала коментарі в стилі / * * /.

Найбільш важливою відмінністю YAML є набір розширень синтаксису, яким немає аналогів в JSON:

- підтримка реляційних даних: в YAML-документі можна посилатися на якір, який зустрівся раніше в файлі / потоці; таким чином можна висловити рекурсивні структури;

- підтримка розширюваних типів даних крім примітивів: рядків, чисел, логічних значень і т. д.;

- підтримка блочного синтаксису з відступами; він дозволяє описати структуровані дані без використання зайвих символів: всіляких дужок, лапок і т. д.

2.3 Аналіз мови розмітки XML

XML (/ ˌeks em el / англ. EXtensible Markup Language) – розширювана мова розмітки. Рекомендований Консорціумом Всесвітньої павутини (W3C). Специфікація XML описує XML-документи і частково описує поведінку XML-процесорів (програм, які читають XML-документи і забезпечують доступ до їх вмісту). XML розроблявся як мова з простим формальним синтаксисом, зручний для створення і обробки документів програмами і одночасно зручний для читання і створення документів людиною, з підкресленням націленості на використання в Інтернеті. Мова називається розширюваним, оскільки ним не фіксується розмітку, яка використовується в документах: розробник вільний створити розмітку відповідно до потреб конкретної області, будучи обмеженим лише синтаксичними правилами мови. Розширення XML – це конкретна граматика, створена на базі XML і представлена словником тегів і їх атрибутів, а також набором правил, що визначають які атрибути і елементи можуть входити до складу інших елементів. Поєднання простого формального синтаксису, зручності для людини, розширюваності, а також базування на кодуваннях Юнікод для

подання змісту документів привело до широкого використання як власне XML, так і безлічі похідних спеціалізованих мов на базі XML в найрізноманітніших програмних засобах. XML є підмножиною SGML.

Елемент і його розмітка.

Елемент є поняттям логічної структури документа. Кожен документ містить один або кілька елементів. Межі елементів представлені початковим і кінцевим тегами. Ім'я елемента в початковому і кінцевому тегах елемента має збігатися. Елемент може бути також представлений тегом порожнього, тобто не включає в себе інші елементи і символьні дані, елемента.

Тег (англ. Tag) – конструкція розмітки, яка містить ім'я елемента.

Початковий тег: <element1>

Кінцевий тег: </ element1>

Тег порожнього елемента: <empty_element1 />

В елементі атрибути можуть використовуватися тільки в початковому тегу і тезі порожнього елемента.

Приклад кулінарного рецепта, розмічені за допомогою XML:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE recipe>
<recipe name="хліб" preptime="5min" cooktime="180min">
   <title>
      Простий хліб
   </title>
   <composition>
      <ingredient amount="3" unit="стакан">Мука</ingredient>
      <ingredient                                amount="0.25"
unit="грамм">Дріжжі</ingredient>
      <ingredient      amount="1.5"      unit="стакан">Тепла
вода</ingredient>
   </composition>
   <instructions>
     <step>
        Змішати всі інгрідієнти та ретельно замісити.
     </step>
     <step>
        Прикрити тканиною і залишити в теплому приміщення на
одну годину.
     </step>
```

```
          <step>
             Замісити ще раз, поставити на противень та засунути
в піч.
          </step>
        </instructions>
     </recipe>
```

Ефективність використання XML.

XML – мова розмітки, іншими словами, засіб опису документа. Саме в ніші документів, текстів, де частка різнотипних символьних даних велика, а частка розмітки мала – XML успішний. З іншого боку, обмін даними у відкритих системах не зводиться до обміну документами. Надмірність розмітки XML (а в цілях розробки мови прямо вказано, що лаконічність не є пріоритетом проекту) позначається в ситуаціях, коли дані не вписуються в традиційну модель документа. Стрічка новин, наприклад, що оформляється з використанням синтаксису XML (формати RSS, Atom), являє собою не документ в традиційному розумінні, а потік однотипних міні-документів – багатослівна і надлишкова розмітка в цьому випадку становить істотну частину переданих даних.

W3C стурбований ефективністю застосування XML, і відповідні робочі групи займаються цією проблемою (до початку 2013 року нормативні документи не розроблені).

Інша ситуація, коли формати XML можуть виявитися не кращим рішенням – робота з даними з простою структурою і невеликим за обсягом змістом полів даних. У цьому випадку частка розмітки в загальному обсязі велика, а програмна обробка XML може виявитися невиправдано витратною, в порівнянні з роботою з даними більш простої структури. У цій області розробники розглядають засоби, спочатку орієнтовані на дані, такі як INI, YAML, JSON.

2.4 Аналіз стандарту MIME

MIME – стандарт, що описує передачу різних типів даних по електронній пошті, а також, в загальному випадку, специфікація для кодування інформації і форматування повідомлень таким чином, щоб їх можна було пересилати через Інтернет.

MIME визначає механізми для передачі різного роду інформації всередині текстових даних (зокрема, за допомогою електронної пошти), а саме: текст на мовах, для яких використовуються кодування, відмінні від ASCII, і нетекстові дані, такі, як картинки, музика, фільми і програми. MIME є також фундаментальним компонентом комунікаційних протоколів, таких як HTTP, яким потрібно, щоб дані передавалися в контексті повідомлень, подібних e-mail, навіть якщо дані реально не є e-mail.

Основний формат електронних повідомлень визначено в RFC 5322, який є оновленою версією RFC 2822 (який, в свою чергу, є оновленою версією RFC 822). Ці стандарти визначають схожі формати для текстових e-mail-заголовків і вмісту і правил, що відносяться до общеіспользуемих полях, таким як To :, Subject :, From: і Date :. MIME визначає набір e-mail-заголовків для визначення додаткових атрибутів повідомлення, включаючи тип контенту, і визначає безліч кодувань, які можуть бути використані для подання 8-бітних бінарних даних за допомогою символів з 7-бітного ASCII.

MIME також визначає правила для кодування символів з Extended ASCII (з кодами 128-255) в заголовках e-mail-повідомлення, таких як Subject.

MIME розширюємо для нових типів – його визначення включає метод для реєстрації нових типів контенту та інших атрибутів.

Організація даних

Формат MIME підтримує передачу кількох сутностей в межах одного повідомлення. Причому суті можуть передаватися не тільки у вигляді однорівневої послідовності, але і у вигляді ієрархії з вкладенням елементів один в одного. Для позначення множини вмісту використовуються медіатіпи

multipart / *. Робота з такими типами здійснюється за загальними правилами, описаним в RFC 2046 (якщо інше не визначено конкретним медіа-типом). Якщо одержувачу невідомо, як працювати з типом, то він обробляє його так само, як multipart / mixed.

Для передачі множинного повідомлення в заголовок Content-Type додається параметр boundary (межа), який позначає послідовність символів, які поділяють частини повідомлення. Межа може складатися з цифр, букв і символів « '() + _, -. /: =?». При використанні спеціальних символів (не цифри і букв) значення параметра boundary слід укладати в подвійні лапки ". Максимальна довжина кордону – 70 символів.

Початок кожної частини повідомлення позначається рядком --boundary. Кінець останнього повідомлення позначається рядком --boundary--. Найперші символи розриву рядків CRLF (коди 13 і 10), якими починаються і закінчуються прикордонні рядки, які не входять в вміст самої частини. Якщо за ними слідують ще переноси рядків, то вони вже належать включається частини.

Перед першою частиною і після останньої може бути додатковий текст. Він називається преамбулою і епілогом, відповідно. У протоколі HTTP ці елементи ігноруються. В електронному листі преамбула може містити текст, виведений клієнтами електронної пошти, які не розуміють формату MIME.

На самому початку включається частини розташовуються заголовки, що описують її вміст (Content-Type, Content-Length і т. П.). Перед безпосередньо тілом частини обов'язково повинна бути порожній рядок, навіть якщо заголовки відсутні. Якщо не визначений Content-Type, то він береться за замовчуванням – text / plain.

Оскільки старий стандарт RFC 822 все ще використовується, а MIME можливо, зміниться і доповниться в майбутньому, поштовій програмі необхідно знати, застосований новий стандарт в конкретному листі чи ні. Тому

в заголовок введено нове поле "MIME-Version", яке оголошує версію стандарту, відповідно до якого написано цей лист.

Всі поштові повідомлення, складені відповідно до MIME-стандартом, повинні мати це поле в своєму заголовку, наприклад:

```
MIME-Version: 1.0
```

Так як можливо, в майбутньому формат заголовка листа може розширитися, формально зміст поля "MIME-version" дається наступним чином:

```
версія: = "MIME-Version" ":" 1 * DIGIT "." 1 * DIGIT
```

Майбутні значення версії формату, коор можуть замінити "1.0", повинні бути цілими числами, розділеними крапкою. Якщо лист отримано зі значенням версії MIME, відмінним від "1.0", воно не буде розглядатися поштовою програмою, як відповідне даної специфікації.

Важливо, що поле заголовка "MIME-Version", має розташовуватися в самам початку листа. Це не обов'язково для кожної з частин тіла листи в разі многочастевого листи, але обов'язково для заголовків частин типу "message", якщо і тільки якщо ця частина сама по собі декларована як відповідна специфікації MIME.

Неможливо повністю визначити як поштова програма, яка підтримує MIME, повинна інтерпретувати лист, що має значення MIME-version, відмінне від "1.0". Але, як мінімум, поштова програма повинна попередити користувача про те, що лист написано в незнайомому їй форматі.

Всі поля заголовка, включаючи MIME-Version, Content-type, і т.д., повинні відповідати загальним синтаксичним правилам, визначеним в RFC 822. Зокрема, допускається включення коментарів (тобто, наступні 2 прикладу еквівалентні):

```
MIME-Version: 1.0
MIME-Version: 1.0 (Generated by GBD-killer 3.7)
Поле Contrent-Type
```

Призначення цього поля – найбільш повний опис даних, що містяться в тілі, з тим, щоб поштовий агент (програма) одержувача могла вибрати відповідний механізм для їх обаботкі. Вперше це поле було визначено в RFC 1049, але мало більш простий синтаксис.

Дане поле включає в себе ідентифікатори типу і підтипу, а також може містити деяку допоміжну інформацію, яка може знадобитися для конкретного типу даних. Після ідентифікаторів типу і підтипу решта поля – просто набір парамеров, заданих в порядку "атрибут / значення". Набір параметрів залежить від типу даних. (Зокрема, не може бути глобально-значущих параметрів, справедливих відразу для всіх типів вмісту ьела листи. Глобальні механізми в MIME-моделі реалізовані за допомогою введення додаткових полів "Content-*"). Черговість параметрів значення не має. У числі певних параметрів – "charset", який декларує символьний набір (кодування, кодову сторінку – це все синоніми) тіла листи. Коментарі допускаються.

При відсутності поля Content-type або поля MIME-Version в заголовку MIME-листи можна бути точно впевненим, що лист має мовне кодування саме US-ASCII, оскільки можуть ще зустрічатися поштові програми, які не використовують угоди MIME. Але хоча можливо, що лист, що не містить в заголовку полів MIME-Version і Content-Type, може містити все, що завгодно, наприклад, юніксовиій tar-архів, стиснений gzip'ом і оброблений uencode, все ж, творцям поштових програм рекомендується залишати цей факт без уваги і орієнтуватися на значення за замовчуванням, тобто "Text / plain; charset = us-ascii".

Необхідно врахувати, що в майбутньому очікується помітне збільшення числа реєстрованих типів і особливо підтипів вмісту листів. Якщо поштова

програма зустріне невідоме їй значення поля Content-Type, вона повинна інтерпретувати вміст цього листа як "application/octet-stream" (див. вище).

Поле заголовку Content-Transfer-Encoding

Багато типів даних, що пересилаються через email вимагають "натурального" уявлення, тобто, 8-бітний набір символів або двійкового коду (що для машини – одне і те ж, тільки представимо для користувача по-різному). У такому вигляді дані не можуть бути переслані по 7-бітовим поштовим протоколам, наприклад, RFC 821, який, до того ж, обмежує довжину рядка 1000 символів.

Стандартні механізми конвертації пошти в 7-бітний коротко-стрічковий формат, оптимальний для поштового транспорту, описує поле заголовка Content-Transfer-Encoding.

Основний формат електронних повідомлень визначено в RFC 5322, який є оновленою версією RFC 2822 (який, в свою чергу, є оновленою версією RFC 822). Ці стандарти визначають схожі формати для текстових e-mail-заголовків і вмісту і правил, що відносяться до общеіспользуемих полях, таким як To :, Subject :, From: і Date :. MIME визначає набір e-mail-заголовків для визначення додаткових атрибутів повідомлення, включаючи тип контенту, і визначає безліч кодувань, які можуть бути використані для подання 8-бітних бінарних даних за допомогою символів з 7-бітного ASCII.

MIME також визначає правила для кодування символів з Extended ASCII (з кодами 128-255) в заголовках e-mail-повідомлення, таких як Subject.

MIME розширюємо для нових типів – його визначення включає метод для реєстрації нових типів контенту та інших атрибутів.

Формат MIME підтримує передачу кількох сутностей в межах одного повідомлення. Причому суті можуть передаватися не тільки у вигляді однорівневої послідовності, але і у вигляді ієрархії з вкладенням елементів один в одного. Для позначення множини вмісту використовуються медіатіпи multipart / *. Робота з такими типами здійснюється за загальними правилами,

описаним в RFC 2046 (якщо інше не визначено конкретним медіа-типом). Якщо одержувачу невідомо, як працювати з типом, то він обробляє його так само, як multipart / mixed.

Взагалі, поле Content-Type самого верхнього рівня використовується для оголошення загального типу даних, в той час як підтип визначає спеціальний формат для даних цього типу. Так, значення "image / xyz" поля Content-Type повідомляє користувача програмі, що дані є графічним зображенням (image), навіть якщо ця поштова програма не має поняття про спеціальному форматі "xyz" цієї картинки. Але ця інформація може бути використана програмою, наприклад, щоб вирішити, чи показувати користувачеві строкоие дані невідомого підтипу – показ таких даних може бути виправданий для незнайомих підтипів тексту, але не для незнайомих підтипів графіки, аудіо або відео. З цієї причини, дані зареєстрованого підтипу аудіо, графіки, тексту або відео не повинні містити в собі частини іншого підтипу – для утримання в листі даних одного типу, але різних підтипів слід використовувати тип "multipart" або "application".

Хоча багато параметрів (модифікатори підтипів) мають сенс лише для конкретного типу, деякі все ж є глобальними в тому сенсі. що вони можуть бути застосовані до всіх типів (наприклад, параметр "boundary" застосовується лише до типу "multipart", а параметр "charset" може використовуватися з декількома типами).

Поки імен типів тільки сім, і поки цього достатньо. Крім того, передбачається, що розширення існуючого набору підтримуваних типів даних буде проводитися зарахунок введення нових підтипів цих спочатку певних типів даних. В майбутньому додавання імен типів верхнього рівня може бути зроблено тільки після ухвалення нової версії стандарту MIME. Якщо з якої-небудь іншої причини в існуючій версії використовується Незареєстровані тип содежімого, йому повинно бути дано ім'я, яке починається з "X-", щоб

підкреслити його нестандартний статус і заздалегідь попередити конфлікт з офіційним ім'ям типу, яке може бути введено пізніше.

Існує два прийнятних механізму для введення нових підтипів для поля Content-Type:

Нестандартні значення (що починаються з "X-") можуть бути опредлени за домовленістю для двох або більше людей, що спілкуються один з одним поштових агентів (програм) без будь-якої зовнішньої реєстрації та стандартизації.

Нові стандартні значення підтипів повинні бути задокументовані, зареєстровані і випробувані в IANA, як описано в додатку "E" RFC 1521. Якщо новй підтип пропонується для широкого використання, формат, описуваний їм, повинен бути описаний в опублікованій специфікації і, можливо, запропонований до стандартизації .

Сім визначених типів верхнього рівня, більш детально, є наступними:

– text – текстова інформація. Основою підтип – "plain" – відповідає звичайному неформатіровнному тексту і не вимагає спеціального програмного забезпечення для відображення цього тексту за винятком підтримки національних кодувань. Інші підтипи використовуються в разі розміченого тексту, коли за допомогою спеціальної програми можна поліпшити його візуалізацію, але для розуміння ідеї змісту можна обійтися і без додаткового ПЗ. Можливі підтипи можуть описувати формати, які легко читаються різними текстовими процесорами;

− multipart – вміст листа складається з певної кількості частин, що містять дані різних взаємонезалежних типів. Спочатку визначено чотири підтипи:

− mixed – основний;

− alternative – для подання одних і тих же даних в різних форматах;

− parallel – якщо різні частини документа повинні проглядатися одночасно;

− digest – якщо кожна з частин тіла листи має тип "message";

− message – лист в листі. Тіло, що містить дані типу "message", саме є листом або частиною листа, повністю відформатованого відповідно до стандарту RFC 822, яке, в свою чергу, може містити своє власне поле заголовка "Content-Type".

Підтипи:

− rfc822 – основний;

− partial – визначено для частково-цитованих листів для запобігання фрагментації тел містяться листів в разі занадто великий їх загальної довжини для можливостей поштового транспорту;

− external-body – використовується, щоб вказати, що тіло листа дуже велике і знаходиться поза листи.

При відсутності поля Content-type або поля MIME-Version в заголовку MIME-листи можна бути точно впевненим, що лист має мовне кодування саме US-ASCII, оскільки можуть ще зустрічатися поштові програми, які не використовують угоди MIME. Але хоча можливо, що лист, що не містить в заголовку полів MIME-Version і Content-Type, може містити все, що завгодно, наприклад, юніксовиій tar-архів, стиснений gzip'ом і оброблений uencode, все ж, творцям поштових програм рекомендується залишати цей факт без уваги і орієнтуватися на значення за замовчуванням, тобто "Text / plain; charset = us-ascii".

Необхідно врахувати, що в майбутньому очікується помітне збільшення числа реєстрованих типів і особливо підтипів вмісту листів. Якщо поштова програма зустріне невідоме їй значення поля Content-Type, вона повинна інтерпретувати вміст цього листа як "application/octet-stream" (див. вище).

Поле заголовку Content-Transfer-Encoding

Багато типів даних, що пересилаються через email вимагають "натурального" уявлення, тобто, 8-бітний набір символів або двійкового коду (що для машини – одне і те ж, тільки представимо для користувача по-

різному). У такому вигляді дані не можуть бути переслані по 7-бітовим поштовим протоколам, наприклад, RFC 821, який, до того ж, обмежує довжину рядка 1000 символів.

Стандартні механізми конвертації пошти в 7-бітний коротко-стрічковий формат, оптимальний для поштового транспорту, описує поле заголовка Content-Transfer-Encoding.

На відміну від типів вмісту, збільшення кількості значень Content-Transfer-Encoding не є необхідним і навіть небажано. Але встановлення єдиного механізму конвертації не представляється можливим. Існує протиріччя між бажанням ефективно "стиснути" бінарні дані і бажанням трансформувати дані, які, хоча б частково є 7-бітовим текстом, так, щоб їх все-таки можна було читати. З цієї причини необхідні принаймні 2 механізму конвертації: "читабельний" і "щільно утискають".

Значення не чутливі до регістру букв, тобто, Base64, BASE64 і bAsE64 – одне і те ж. Значення "7BIT" означає, що тіло листа вже має 7-бітний формат і не потребує додаткової обробки для пересилання поштою. Це значення належить за замовчуванням, якщо поле заголовка Content-Transfer-Encoding відсутня.

Значення "8bit", "7bit" і "binary" означають, що ніякої трансформації вмісту не проводиться. Однак, вони зроблені різними для індикації того, що з себе представляє вміст листа, і, відповідно, способу обробки, який може знадобитися для даної транспортної системи. Зокрема:

"7bit" означає, що дані є текстом, мають короткі строки і мовне кодування US-ASCII.

"8bit" означає короткі строки, але в них можуть міститися не-ASCII символи (128-255).

"Binary" означає, що тіло листа може містити не-ASCII символи, але рядки можуть бути довільної довжини, тобто занадто довгими для SMTP-

транспорту, і може несоблюдаться угоду за ознакою кінця рядка (CRLF), прийняте в SMTP-транспорті.

Хоча на перший погляд різниця в значеннях Content-Transfer-Encoding може Показати неважливою – адже всі вони означають, що ніякого перетворення немає, але чітка розмітка важлива для поштових шлюзів між різними поштовими системами, що мають різні можливості і особливості роботи, число яких з часом зростає.

Специфікація на поштовий транспорт для пересилання некодованих 8-бітних даних дана в RFC-1426. Однак, немає стандартизованих транспортів рочти Internet, для яких є прийнятним включення в тіло листа некодованих двійкових даних. Таким чином, значення "binary" фактично не є легальним в Internet. Але відповідно до MIME, при використанні поштовою системою транспорту, який вміє працювати з двійковими даними, в разі, коли необхідно надіслати двійкові дані по e-mail, необхідно вказати це в заголовку в поле Content-Transfer-Encoding. Так, значення "image / xyz" поля Content-Type повідомляє користувача програмі, що дані є графічним зображенням (image), навіть якщо ця поштова програма не має поняття про спеціальному форматі "xyz" цієї картинки. Але ця інформація може бути використана програмою, наприклад, щоб вирішити, чи показувати користувачеві строкоие дані невідомого підтипу – показ таких даних може бути виправданий для незнайомих підтипів тексту, але не для незнайомих підтипів графіки, аудіо або відео. З цієї причини, дані зареєстрованого підтипу аудіо, графіки, тексту або відео не повинні містити в собі частини іншого підтипу – для утримання в листі даних одного типу, але різних підтипів слід використовувати тип "multipart" або "application".

П'ять значень, визначених для поля Content-Transfer-Encoding, нічого не говорять про тип вмісту крім вказівки алгоритму кодування яких вимог до поштової транспорту в разі некодованих даних.

3 РЕАЛІЗАЦІЯ СПЕЦИФІКАЦІЇ РОЗРОБЛЕНОЇ СТРУКТУРИ

3.1 Опис eJSON

eJSON (extendedJSON) – це розширення об’єктної нотації JSON. Він являє собою рівень абстракції над структурою даних JSON. eJSON розширює описові можливості JSON за допомогою додаткового рівня абстракції, який представляє собою Деклараційний рівень.

Будь який екземпляр JSON структури може бути представлений за допомогою eJSON. Проте eJSON не варто використовувати безпосередня для опису даних. Задача eJSON – описати дані чистого JSON на вищому рівні абстракції.

Реалізація абстрактного рівня поверх JSON представляє собою введення додаткових операторів, символів, механізмів тощо.

eJSON розширює JSON за допомогою оголошень змінних, реалізації стандартних структур даних (список, словник, масив тощо). При цьому розділ оголошень (Деклараційний розділ) повинен бути чітко відділений від самих даних. Так як eJSON не є кінцевим результатом, а лише рівнем над JSON.

Введені можливості для розширення JSON:

- змінні;
- звернення за посиланням;
- індексування;
- коментарі;
- словник;
- композиція;
- рефлексія;
- умовні оператори;
- арифметичні операції;
- конкатенація стрічок;

- інтерполяція стрічок;
- вхідні параметри.

Будь який eJSON транслюється в чистий JSON. Приклад eJSON формату відображено в додатку Б.}

Даний приклад демонструє підхід до опису деяких окремих елементів eJSON. Для опису змінних, структур та інших елементів eJSON виділено спеціальний Деклараційний об’єкт, який визначається після тіла даних.

Завдяки використання змінних повторювані дані можуть бути перевикористані без затрат пам’яті.

В прикладі в першому блоці описано безпосередньо самі дані. В другому блоці описані елементи eJSON, які можуть бути використані в першому блоці.

Описаний приклад eJSON після трансляції в JSON виглядатиме наступним чином:

```
{
    "test": {
        "arg": 1,
        "asd": "asd"
    },
    "varvalue": {
        "my": 25
    },
    "myvalue": 25,
    "numvalue": 50,
    "tst": 75,
    "arrImpl": [
        25,
        7,
        123,
        777
    ],
    "arrEl": 123,
    "dictEl": "23452345"
}
```

3.2 Принципи та особливості eJSON

Основним принципом eJSON є економія пам’яті шляхом оголошення змінних та їх використання, а також покращення читабільності за допомогою поширеного синтаксису декларації елементів. Тобто місце для безпосередніх даних від-окремлено від місця оголошення змінних.

3.2.1 Змінні

В eJSON визначений механізм оголошення та використання змінних. Усі змінні оголошуються в спеціальному блоці. Згідно стандарту JSON документ представляє собою об’єкт або масив. Тобто весь документ представляє собою один великий об’єкт або масив об’єктів. Кожний документ починається з символів “{“ – якщо документ є об’єктом та “[“ – якщо документ вміщує масив.

Для відповідності стандарту та збереження загального підходу до опису даних для оголошення змінних eJSON виділено окремий блок, який описується після блоку даних. Для подальшого опису ці блоку будуть називатись «блок даних» – блок даних JSON та «Деклараційний блок» – блок опису змінних, структур eJSON. В документі eJSON змінні оголошуються в декларативному блоці, а в блоці даних вони використовуються. Для оголошення змінної та посилання на неї використовується символ “$”:

```
// Блок даних
{"dataElement": $variable },

// Деклараційний блок
{
    $variable: 50
}
```

При трансляції документу eJSON Деклараційний блок компілюється та посилання на зміні в блоці даних заміняються на значення скомпільованих змінних.

Завдяки природі посилань та змінних дані, які повторюються можна описати за допомогою механізму змінних та скоротити об’єм кінцевого файлу або структури.

3.2.2 Звернення за посиланням

В eJSON немає чіткої типізації і тому неможливо перевірити чи правильно було використано та описано зміну. Взамін eJSON пропоную «нечітку типізацію». Тобто обмеження змінних за розміром, типом тощо диктується мовою, в об’єкт якої серіалізується eJSON.

eJSON пропонує механізм посилання на окремі поля та властивості оголошених об’єктів. Тобто структура eJSON дозволяє оголосити комплексний об’єкт на який можливо здійснити посилання з блоку даних:

```
// Блок даних
{
    "dataElement": $object.field
},

// Деклараційний блок
{
    $object: {
        $field: 45
    }
}
```

В даному прикладі в декларативному блоці оголошено об’єкт з назвою «object», який вміщає в собі поле з назвою «field». В блоці даних можна звернутись до конкретного поля об’єкту за допомогою символу посилання – «.» (крапка).

Даний документ eJSON буде інтерпретовано в JSON:

```
{
```

```
    "dataElement": 45
}
```

Також в блоці даних можна використовувати безпосередньо об’єкти.

3.2.3 Індексування

Для роботи з масивами в eJSON реалізовано принцип індексування масивів. Всі елементи оголошеного масиву матимуть такий порядок в якому вони були оголошені.

Для звернення на конкретний елемент масиву в блоці даних використовуються цифрові індекси та символи «[]»:

```
// Блок даних
{
    "arrayElement": $arr[0]
},

// Декларацiйний блок
{
    $array: [
        25,
        50,
        75,
        100
    ]
}
```

Інтерпретований JSON:

```
{
    "arrayElement": 25
}
```

Оголошений масив можна використовувати як окремий елемент даних.

3.2.4. Коментарі

eJSON представляє стандартний механізм коментарів. Коментарі ніяк не впливають на блок даних чи Деклараційний блок. Для додавання однолінійних коментарів використовуються символи «//»:

```
// Блок даних
{
    // Коментар
    "arrayElement": 25
}
```

Для додавання багато лінійних коментарів використовуються символи «/*» та «*/» (відкриваючий та закриваючий маркери відповідно):

```
// Блок даних
{
    /*
    Багато
    лінійний
    коментар
    */
    "arrayElement": 25
}
```

Коментарі можуть бути оголошені в будь-якому місці документу eJSON.

3.2.5. Словник

Словник представляє собою структуру даних типу «ключ-значення». Словник це масив елементів, в якому доступ до елементів відбувається не за допомогою індексу, а за допомогою ключа. Ключем може бути значення будь-якого примітивного типу – число, стрічка тощо.

В eJSON словники мають обмеження по ключам – він може бути лише стрічкового типу.

Оголошуються вони як зміна в декларативному блоці:

```
// Декларацiйний блок
{
    $dictionary: [
        {
            $key: "123455",
            $value: "value"
        }
    ]
}
```

В даному прикладі оголошено словник з одним записом, який має ключ «123455» та значення «value». Так як словник дещо відрізняється від масиву, а саме – він не може зберігати прості об’єкти, а може зберігати лише об’єкти типу «ключ-значення», назва цих елементів обмежена змінними «$key» та «$value».

eJSON словник та трансляція:

```
// Блок даних
{
    "dictionaryElement": $dictionary["123455"]
},
// Декларацiйний блок
{
    $dictionary: [
        {
            $key: "123455",
            $value: "value"
        }
    ]
}
```

Після трансляції в JSON:

```
{
    "dictionaryElement": "value"
}
```

3.2.6. Композиція

eJSON надає спеціальний механізм для реалізації композиції документів. В даному контексті композиція документів означає що один документ може бути використаний в якості частини іншого.

Для використання композиції необхідно щонайменше два eJSON документи (один з них може бути JSON документом) та шлях до документа. Для оголошення посилання на зовнішній документ використовується спеціальний символ «&»:

```
// Декларацiйний блок
{
    &anotherDocument: "C:/another_document.json"
}
```

Далі зовнішній документ можна використовувати як зміну типу «об’єкт» або «масив» (залежить від типу документа). Для доступу до полів документа використовується символ «.»:

```
// Блок даних
{
    "compositionProperty": $anotherDocument.property,
    "compositionNum": $anotherDocument.num
},
// Декларацiйний блок
{
    &anotherDocument: "C:/another_document.json"
}

// Уявимо що це зовнішній документ
{
    "property": "value",
    "num": 123
}
```

Після трансляції:

```
{
    "compositionProperty": "value",
    "compositionNum": 123
}
```

Важливо помітити, що eJSON є строго регістровим. Тобто змінні з однаковим іменем, але з використанням літер різного регістру будуть інтерпретуватись як окремі змінні.

3.2.7. Рефлексія

eJSON дозволяє використовувати змінні примітивних типів для опису імен полів блоку даних, що представляє собою простий механізм рефлексії. Проте якщо змінна виявиться не примітивного типу (наприклад – об’єктом) то інтерпретатор поверне помилку. Для використання значень змінних в якості імен для полів блоку даних реалізовано наступний синтаксис: «”$[ім’я змінної]”»:

```
// Блок даних
{
    "$property": 12345
},
// Деклараційний блок
{
    $property: "name"
}
```

Після трансляції:

```
{
    "name": 12345
}
```

Приклад з посиланням на поле об’єкту:

```
// Блок даних
{
    "$var.name": 12345
},
// Деклараційний блок
{
    $var: {
        $name: "number"
    }
}
```

Після трансляції:

```
{
    "number": 12345
}
```

3.2.8. Умовні оператори

eJSON реалізує простий механізм постановки умов. Це дозволяє в залежності від виконання цих умов отримувати різний результуючий JSON. Для опису умови необхідно скористатись конструкцією «if [умова] then [результат 1] else [результат 2]»:

```
// Блок даних
{
    "number": $result
},
// Деклараційний блок
{
    $result: if $num > 0 then true else false,
    $num: 10
}
```

Після трансляції:

```
{
    "number": true
}
```

Для простих умов з одним результатом можна упустити оператор «else». Але якщо умова не виконається результатом буде «null»:

```
// Блок даних
{
    "number": $result
},
// Деклараційний блок
{
    $result: if $num > 0 then true,
    $num: -10
}
```

Після трансляції:

```
{
    "number": null
}
```

3.2.9. Арифметичні операції

eJSON надає деякий перелік арифметичних операцій:

- додавання;
- множення;
- віднімання.

Ділення відсутнє через типізацію статично-типізованих мов програмування. В більшості таких мов цілі числа і числа з плаваючою крапкою являють собою окремі типи. Результатом ділення може бути не ціле число, тому при спробі серіалізувати поле це може призвести до помилки.

Арифметичні операції в eJSON:

```
// Блок даних
{
    "number": $multiply
},
// Деклараційний блок
{
    $multiply: $num1 * $num2,
    $num1: 16,
    $num2: 2
}
```

Після трансляції:

```
{
    "number": 32
}
```

Арифметичні операції можна проводити лише над числовими змінними (за виключенням додавання). Якщо в eJSON документі описано операції множення над змінними типу «об’єкт», то при десеріалізації виникне помилка.

3.2.10 Конкатенація стрічок

Для конкатенації (злиття) стрічок в eJSON визначено оператор додавання – «+»:

```
// Блок даних
{
    "concat": $concatResult
},
// Деклараційний блок
{
    $concatResult: $str1 + $str2,
    $str1: "Hello, ",
    $str2: "world!"
}
```

Після трансляції:

```
{
    "concat": "Hello, world!"
}
```

3.2.11 Інтерполяція стрічок

Для інтерполяції стрічкових змінних в eJSON використовуються символи «&» (амперсанд), який встановлюється перед початком стрічки та «{}» для виділення елементів стрічки:

```
// Блок даних
{
    "concat": $interpolation
},
// Декларацийний блок
{
    $interpolation: &"{$str1} with {$str2}.",
    $str1: "By nice",
    $str2: "Alex"
}
```

Після трансляції:

```
{
    "concat": "Be nice with Alex."
}
```

3.2.12 Вхідні параметри

eJSON реалізує механізм обробки вхідних параметрів, тобто параметрів які оголошуються на рівні серіалізації. Для оголошення вхідних параметрів використовується символ «*». В самому документі вхідний параметр представляє собою звичайну змінну анонімного типу:

```
// Блок даних
{
```

```
    "concat": $inputParam
},
// Декларацiйний блок
{
    *inputParam
}
```

Проте через природу типізації неможливо бути впевненим що при десеріалізації буде передано значення очікуваного типу. Саме тому над вхідними параметрами заборонені будь-які операції, звернення за посиланням, індексом чи ключем.

3.3 Приклади використання eJSON

Найбільш вірогідним способом застосування розробленого офрмату eJSON є опис конфігураційних фалів для веб-додатків. Наступний приклад демонструє застосування формату eJSON для опису конфігураційних файлів для веб-додатку на основі платформи ASP.NET Core.

По замовчуванню ASP.NET Core використовує стандарт JSON для опису конфігураційних файлів. При цьому для кожного середовища описується власний, окремий файл з назвою середовища в назві файлу (рисунок 3.1).

Кожен файл представляє власні налаштування для кожного середовища. Наприклад при роботі додатку в середовищі «Production», будуть використовуватись лише файли appsettings.json і appsettings.Production.json. Файл appsettings.json являє собою узагальнений файл, який розподіляє власні налаштування серед фалйів середовищ. Досить часто подібні файли вміщують однакові налаштування з мінімальними відмінностями, що не є ефективно. Приклад файлу appsettings.Production.json відображено в додатку В. Приклад файлу appsettings.Staging.json выдображено в додатку Г.

Рисунок 3.1 – Стандартний набір конфігураційних фалів проекту на основі ASP.NET Core

Дані конфігураційні файли представляють налаштування для розвертання та реалізації клієнтів Amazon Web Services. Один файл описує налаштування для релізної версії проекту, інший для розробницької версії. Як видно з прикладів файли майже не відрізняються за вмістом. Саме тут можна реалізувати деякі принципи eJSON. За допомогою зовнішніх параметрів, композиції, умовних операторів та індексування можна помістити налаштування усіх середовищ та при цьому значно спростити набір налаштувань. Приклад трансльованого eJSON відображено в додатку Д.

Об’єднаний конфігураційний файл за допомогою eJSON. При цьому можна описувати необмежену кількість налаштувань для безлічі середовищ, проте файл буде рости набагато повільніше, ніж при використанні стандартного JSON.

ВИСНОВКИ

Під час виконання магістерської науково-дослідної роботи було розроблено абстрактне операційне розширення формату даних JSON – eJSON. Результатом стала специфікація текстового структурного формату даних eJSON, який являє собою розширення існуючого формату JSON. eJSON реалізовую набір інструментів та принципів для підвищення ефективності передачі та зберігання даних у форматі JSON. eJSON може використовуватись в якості атомарної структури для зберігання даних, або транслюватись в дані типу JSON.

1. Введено поняття змінної. Для реалізації змінних використано текстовий маркер «$». Це дало можливість повторно використовувати дані, які повторюються, створювати умови, операції та посилання в документі.

2. Введено поняття композиції. Для реалізації композиції використано текстовий маркер «&». Це дало можливість створювати композитні документи з посиланнями один на одного. Дані з одних документів можуть бути використані в іншому документі.

3. Введено поняття вхідних параметрів. Для реалізації використано текстовий символ «*». Це дозволило створювати залежності від зовнішніх середовищ та умов серіалізації.

4. Введено поняття рефлексії. Для реалізації використано текстовий символ «$». Це дозволило використовувати оголошенні змінні в якості назв елементів документу.

5. Введено поняття умовних операторів. Для реалізації використано синтаксис типу «якщо (умова) то (результат 1) інакше (результат 2)». Це дозволило описувати умови для десеріалізації даних в залежності від інших даних.

6. Введено поняття арифметичних операцій. Для реалізації використано стандартний механізм роботи з операторами та операндами. Це дозволило створювати спеціальні «нечіткі» дані, які формуються з результатів арифметичних операцій під час де-серіалізації.

7. Введено поняття коментарів. Для реалізації використано текстові символи «//» для одно-стрічкових коментарів та «/*(коментар)*/» для багато-стрічкових. Це дозволило вводити додатковий опис даних, призначених для людей.

Для подальших досліджень можна виділити необхідність дослідження природи структурування даних в залежності від умов та цілей, а також ідею розподілення форматування на абстрактні рівні.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Березький О.М. Методичні рекомендації до виконання магістерської роботи з освітнього ступеня “Магістр”. Спеціальність: 123 – Комп’ютерна інженерія. Магістерська програма – Комп’ютерна інженерія" / О.М. Березький, Л.О. Дубчак, Г.М. Мельник /Під ред. О.М. Березького – Тернопіль: ТНЕУ, 2018.– 41 с.
2. Гураль І.В. Методичні вказівки до оформлення курсових проектів, звітів про проходження практики, випускних кваліфікаційних робіт для студентів спеціальності «Комп’ютерна інженерія» / І.В. Гураль, Л.О. Дубчак / Під ред. О.М. Березького. – Тернопіль: ТНЕУ, 2019. – 33 с.
3. Бодров О. М. Проектування та розробка мови розмітки опису проектних рішень SDML (Solution Description Markup Language) Частина 1. 15 квітня 2019 р., с. 18;
4. Бодров О. М. Абстрактна операційна специфікація форматування даних даних eJSON (extended JavaScript Object Notation) Частина 1. 14 листопада 2019., с. 27.
5. RFC 2046 "Plain text does not provide for or allow formatting commands, font attribute specifications, processing instructions, interpretation directives, or content markup."
6. Молчанов И.Н. Машинные методы решения прикладных задач. Алгебра, приближение функций. – К.: Наук. думка, 1987. 288 с.
7. Химич А.Н. Оценки возмущений для решения задачи наименьших квадратов // Кибернетика и системный анализ. 1996. № 3. С. 95-102.
8. Химич А.Н. Оценки полной погрешности решения систем линейных алгебраических уравнений для матриц произвольного ранга // Компьютерная математика. 2002. № 2. С. 41—49.

9. Химич А.Н., Войцеховский С.А., Брусникин В.Н. О достоверности линейных математических моделей с приближенно заданными исходными данными // Ма тематические машины и системы. 2004. № 3. С. 54—62.

10. Михалевич В.С., Молчанов И.Н., Сергиенко И.В. и др. Численные методы для многопроцессорного вы числительного комплекса ЕС / Под ред. И.Н. Мол чанова. М.: Изд-во ВВИА, 1986. 401 с

11. Молчанов И.Н., Химич А.Н., Попов А.В. и др. Об эффективной реализации вычислительных алгоритмов на MIMD-компьютерах // Искусственный интеллект. 2005. № 3. С. 175—184.

12. Химич А.Н., Молчанов И.Н., Попов А.В., Чистякова Т.В., Яковлев М.Ф. Параллельные алгоритмы решения задач вычислительной математики. К.: Наук. думка, 2008. 247 с.

13. Молчанов И.Н. Проблемы интеллектуализации MIMD-компьютеров // Кибернетика и системный анализ. 1998. № 1. С. 37-46.

14. Молчанов И.Н. Интеллектуальные компьютеры средство исследования и решения научно-тех нических задач // Кибернетика и системный анализ. 2004. № 1. С. 175-179.

15. Сергиенко И.В., Молчанов И.Н., Химич А.Н. Интеллектуальные технологии высокопро изво дительных вичислений // Кибернетика и системный анализ. 2010. № 5. С. 64—176.

16. Молчанов И.Н., Мова В.И., Стрюченко В.А. Ин теллектуальные компьютеры для исследования и решения научно-технических задач новое направление в развитии вычислительной техники // Зв’язок. 2005. № 7. С. 45—46.

17. Перевозчикова О.Л., Тульчинский В.Г., Ющенко Р.А. Построение и оптимизация параллельных компьютеров для обработки больших объемов данных // Кибернетика и системный анализ. 2006. № 4. С. 117—129.

18. Гонсалес Р., Вудс Р. Цифровая обработка изображений/ Пер. с англ. М.: Техносфера, 2005. 1072 с.

19. Гонсалес Р., Вудс Р., Эддинс С. Цифровая обработка изображений в среде MATLAB. Москва: Техносфера, 2006. 616с.

20. Ту Дж., Гонсалес Р. Принципы распознавания образов.- М.:Мир, 1998.-411 с.

21. Шлезингер М.И., Главач В. Десять лекций по статистическому и структурному распознаванию. Киев: Наук. думка, 2004, 546 с.

22. Введение в MATLAB: Учеб. пособие/ Л. А. Мироновский, К. Ю. Петрова; ГУАП. СПб., 2006. 164 с.

23. Фурман Я.А. Введение в контурный анализ. Приложения к обработке изображений и сигналов/Я. А. Фурман, А. В. Кревецкий, А. К. Передреев, А. А. Роженцов, Р. Г. Хафизов, И. Л. Егошина, А. Н. Леухин/ М.: ФИЗМАТЛИТ, 2003, 592стр.

24. William K. Pratt Digital image processing/ Third Edition/ John Wiley & Sons, Inc. 2001. 723 c/

25. Поспелов Д.А. Искусственный интеллект. Справочник. Книга 2. Модели и методы/М.: Радио и связь, 1990. 304 с.

26. Бодянский Е.В., Руденко О.Г. Искусственные нейронные сети: архитектуры, обучение, применения/Харьков: Телетех, 2004. 369с.

27. Архипов, В. Системы для «умного» здания / В. Архипов .- М. : "СтройМаркет", 1999.- № 45.- 182с.

28. Евсеев, Ю.А. Симисторы и их применение в бытовой электроаппаратуре / Ю.А Евсеев, С. С. Крылов .- М.: Энергоатомиздат, 1990. 120 с.

29. Быков, В. С. Введение в технологию X-10 / В. С. Быков .М.: Лаборатория домашних технологий i-Home, 2006.- 25 с.

30. Управление умным домом через интернет [Электронный ресурс] [2006].- Режим доступа: http://smarthouse.rostov.ru.

31. Акустические системы, системы освещения, конференц-системы и системы оповещения [Электронный ресурс] [2008].- Режим доступу : http://www.iberi.ru/?trid=693.

32. Томас Уильямс Современные системы безопасности NovaLit, 2009 324 ст.

33. Шахинпур М., Зенкевич С.Л., Дмитриев С.С. Курс робототехники 1980. 527 с.

34. Схемотехника аналоговых и аналого-цифровых электронных устройств. Волович Г. И. 2005. 772 с.

35. Амосов В.В. Схемотехника и средства проектирования цифровых устройств. 2007 год. 542 с.

36. Китаев Ю.В. Основы цифровой техники. Учебное пособие. 2007 год. 87 с.

37. Иванов Л. В. Факторный анализ в социальных исследованиях / Л. В. Иванов, В. С. Смирнов. М. : Наука, 1996. 352 с.

38. Власов П. К. Психология менеджмента / П. К. Власов, А. В. Лепницкий, И. М. Лущихина [и др.]. Х. :Гуманит. центр, 2007. 300 с.

39. Харман Г. Современный факторный анализ М. : Статистика, 1972. 489 с.

40. Классификация и кластер / ред. Дж. Вэн Райзин. М. : Мир, 1980. 389 с.

41. Петров В. С. Применение методов кластерного анализа при обработке данных экспертного опроса / В. С. Петров // Изв. АН СССР. Сер.: Техн. кибернетика. 1985. Т. 202, № 3. С. 15–18.

42. Zhang Z. Experimental research on the localized electrochemical micro-machining / Zhang, Zhu // Електрохімія – Russian Journal of Electrochemistry. 2008, Т. 44, № 8. С. 926–930.

43. Тишков В. Т. Кластерный анализ в социальных исследова¬ниях / В. Т. Тишков // Вестн. Харьк. политехн. ин та. Сер.: Техн. кибернетика и ее приложения. 1990. № 260, вып. 10. С. 5–7.

44. Лобанова Л. С. Оптимізація обчислень в інтегральному методі найменших квадратів наближення функцій однієї та двох змінних / Л. С. Лобанова, В. О. Пасічник, О. О. Черняк // Вісник НТУ «ХПІ». 2010. № 9. С. 57–62.

45. Иванов Л. В. Новый алгоритм классификации объектов, описываемых качественными признаками / Л. В. Иванов, В. С. Петров // Статистический анализ социально-экономи¬ческих данных / ред. Р. В. Сидоров. К. : Наук.думка. 1997. С. 57–65.

46. Иванов Л. В. Статистический подход к обработке социоло¬гических данных / Л. В. Иванов // Труды междунар. конф. «Социология и математика». Т. 2. Х. : НТУ «ХПИ», 2006. С. 5–9.

47. Гамаюн І. П. Оцінювання міри схожості між об’єктам, що характеризуються кількісними і номіналь¬ними ознаками / І. П. Гамаюн, О. М. Безменова // Інформа¬ційні технології: наука, техніка, технологія, освіта, здоров’я. Тези доповідей XXI міжнародної науково-практичної конфе¬ренції. Ч. 1 (29–31 травня 2013 р., Харків). Х. : НТУ «ХПІ», 2013. С. 9.

48. Національний технічний університет «Харківський політех¬нічний інститут». Вісник НТУ «ХПІ» / Національний технічний університет «Харківський політехнічний інститут». НТУ «ХПІ», 2014. Режим доступу : http://vestnik.kpi.kharkov.ua. Дата звертання : 30 березня 2014.

49. Азаренков В. И. Аналитическое решение уравнения тепло-проводности в задачах анализа и синтеза температурных полей радиоэлектронной аппаратуры :дис. канд. техн. наук : 01.05.02 / Азаренков Х., 2015. 225 с.

50. Северин В. П. Моделі та методи оптимізації показників якості систем автоматичного керування енергоблоку атомної електростанції :

автореф. дис. на здобуття наук. ступеня д-ра техн. наук : спец. 05.13.07 «Автоматизація технологічних процесів» / В. П. Северин. Х., 2007. 35 с.

51. Джафарі Хенджані Сайед Моджтаба. Багатокритеріальний синтез інтелектуальних систем керування енергоблоків АЕС генетичними алгоритмами : автореф. дис. на здобуття наук. сту¬пеняканд. техн. наук : спец. 05.13.07 «Автоматизація технологічних процесів» / Джафарі Хенджані Сайд Моджтаба. Х., 2010. 20 с.

52. ДСТУ 3582–97. Скорочення слів в українській мові у бібліографічному описі. К. : Держстандарт України, 1998. –27 с.

53. ГОСТ 8.586.5–2005. Методика выполнения измерений расхода и количества жидкостей и газов с помощью сужающих устройств. М. :Стандартинформ, 2007. 10 с.

54. Палкин М. В. Пат. 2187888, Российская Федерация. Способ ориентирования по крену летательного аппарата с опти¬ческой головкой самонаведения / М. В. Палкин. 2006 р.

55. Чугаева В. 2187888, Российская Федерация. Приемо-передающее устройство / В. И. Чугаева. 2002 р.

56. Основи інформатики: Методичні вказівки до лабораторних робіт: У 2 ч. / Укл .: І.В. Юрченко.- Чернівці: Рута, 2000.- 79 с.

57. Система управління базами Даних Microsoft Access: Методичні вказівки до лабораторних робіт / Укл .: В.С. Сікора, І.В. Юрченко.- Чернівці: Рута, 2002.- 40 с.

58. Комп'ютерні мережі: Методичні вказівки до лабораторних робіт / Укл .: В.С. Сікора, І.В. Юрченко.- Чернівці: Рута, 2002.- 43 с.

59. Операційна система Microsoft Windows: Методичні вказівки до лабораторних робіт / Укл .: В.С. Сікора, І.В. Юрченко.- Чернівці: Рута, 2003.- 48 с.

60. Текстовий редактор Microsoft Word: Методичні вказівки до лабораторних робіт / Укл .: В.С. Сікора, І.В. Юрченко.- Чернівці: Рута, 2003.- 56 с.

61. Семчук А.Р., Юрченко І.В. Економічна інформатика. Навчальний посібнік.- Чернівці: МВІЦ "Місто", 2008.- 426 с.

62. Симонович С., Євсєєв Г. Практична інформатика: універсальний курс.- М .: АСТ-ПРЕСС; Інфорком-Пресс, 1999.- 480 с.

63. Руденко В.Д. та ін. Базовий курс інформатики; за заг. ред. В.Ю.Бікова: [Навч. посіб.]. К .: Вид. група BHV. Кн. 1: Основи інформатики. 2005. 320 с .: іл.

64. Руденко В.Д. та ін. Базовий курс інформатики; за заг. ред. В.Ю.Бікова: [Навч. посіб.]. К .: Вид. група BHV. Кн. 2: Інформаційні технології. 2006. 368 с .: іл.