

Програмиране за разпределени среди

Курсова работа

**„Функционалност, специфична за JSON формата,
която няма директен аналог в XML“**

Съставил: Ангел Любомиров Стойнов

Факултетен номер: 121222150

Група: 40

Съдържание

Детайлно задание	3
JSON	4
XML	5
Специфична функционалност разглеждана в курсовата работа.....	6
Разработка	7
Обща характеристика	7
Файлова структура	7
JsonFormat.cs.....	8
IncorrectXmlFormat.cs	9
XsdFormat.cs	10
XmlFormat.cs.....	12
Цитирани източници.....	13

Детайлно задание

Функционалност, специфична за JSON формата, която няма директен аналог в XML. Функционалността е по ваш избор.

- a) Намерете функционалност свързана с JSON , която няма директен аналог в XML
 - a. не че не може да се направи с XML, но не може под същата форма
- b) Намерете готов JSON файл или създайте подходящ JSON файл с достатъчно данни, върху който може да се демонстрира функционалността (от т.а).
- c) Създайте програма (на среда .Net), която да изпълнява функционалността (от т.а) върху демонстрационния файл (от т.б)
- d) Създайте еквивалентен XML файл на демонстрационния файл (от т.2)
- e) Създайте програма (на среда .Net), която да изпълнява функционалността (от т.а) върху еквивалентния файл (от т.е)
- f) Документирайте разликата и ефекта от разликата. Документирате всичко необходимо за да изпълните предните точки.

JSON

JSON (JavaScript Object Notation) е опростен формат за обмяна на данни. Той е базиран на едно подмножество на езика за програмиране **JavaScript**, Standard ECMA-262 3rd Edition - от декември 1999 г. **JSON** има текстов формат, напълно независим от реализацията на езика, но използва конвенции, които са познати на програмистите на С-подобни езици, включително **C**, **C++**, **C#**, **Java**, **JavaScript**, **Perl**, **Python**, и много други. Тези свойства правят **JSON** идеален формат за обмяна на данни.

JSON се състои от две структури:

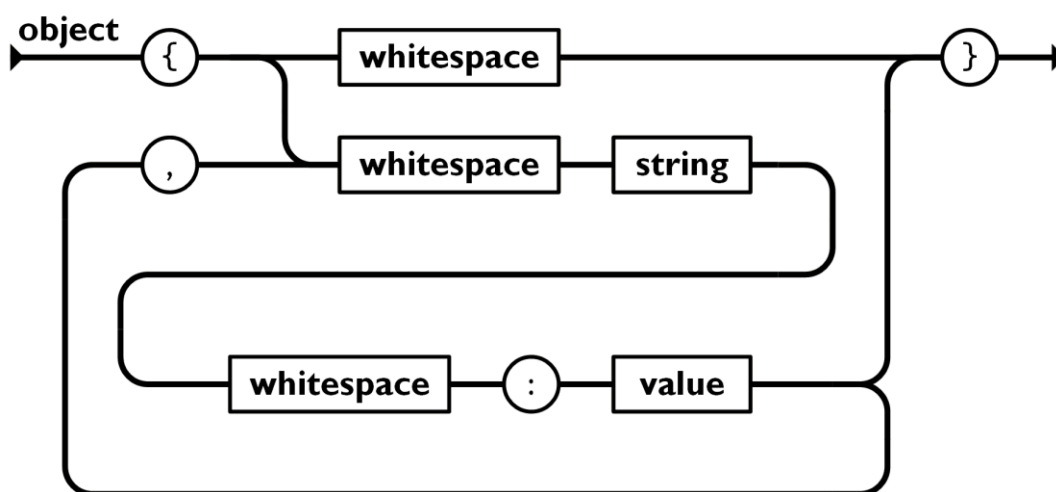
- Колекция от двойки име/стойност (key/pair).
- Подреден списък от стойности.

Обектът започва с **U+007B** ({) и се приключва с **U+007D** (}). Ключовете са в **U+0022** (“) и завършват с **U+003A** (:). Отделните записи за разделени с **U+002C** (,) [1]. Показано на фигура 1 и 2.

Една стойност може да бъде string (низ), число, boolean, null, обект или масив [2]. Тези структури могат да бъдат вложени. [3] [4]

```
1 {  
2     "name": "Angel",  
3     "age": 22,  
4     "universityStudent": true  
5 }
```

Фигура 1, примерен **JSON** файл.



Фигура 2, структура на **JSON** файл. [2]

XML

Extensible Markup Language (XML) представлява универсален формат за описание и съхранение на данни, който улеснява обмена на информация между различни компютърни системи — като уебсайтове, бази данни и външни приложения.

Благодарение на предварително дефинираните синтактични правила, XML осигурява лесно и надеждно предаване на данни през всякакъв тип мрежа, тъй като получателят може точно да интерпретира съдържанието според тези правила. [5]

XML файловете могат да съдържат метаданни, които описват структурата, значението или контекста на самите данни. XML често се използва като основа за различни комуникационни и авторизиращи протоколи, като например SAML, SOAP и други. [6] [7]

Тези протоколи използват XML за формализиране и сигурен обмен на данни между приложения и системи в уеб среда. XML е доста по-вербозен отколкото JSON. Синтаксисът на XML е доста подобен на HTML (фиг. 3). Състои се от отварящи и затварящи тагове, които не са предварително дефинирани. Те се създават от програмиста. Таговете могат да съдържат вложени (дъщерни) елементи, което позволява изграждане на йерархична структура на данните. [5]

Всеки XML документ започва с декларация, която указва версията и кодирането, например [4]:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <person>
3      <name>Angel</name>
4      <age>22</age>
5      <universityStudent>true</universityStudent>
6  </person>
```

Фигура 3, синтаксис на XML. Всички тагове са създадени от програмиста и не са стандартно дефинирани от самия markup language.

Специфична функционалност разглеждана в курсовата работа

В основата си **XML** не поддържа примитивни типове, всеки елемент се разглежда и третира като string (низ) [5]. За да бъде уточнен примитивен тип (number, boolean и т.н.) е необходимо програмно да се добави атрибута **type** (фиг. 4) [8]. Това става възможно с използването на **XSD – XML Schema Definition**, който описва структурата и типовете данни (фиг. 5) [9] [10]. Това допълнително ще утежни файла, особено, ако той е дълъг и комплексен. Въпреки това **XML** ще може да постигне ниво на типизация, подобно на **JSON** [4] [11], но не по подразбиране, а чрез допълнителна схема или логика на приложението.

Друго решение би било, използване на десериализация, то е част от **.NET** средата, но това означава стойностите на **.xml** файла да бъдат облечени (wrap) в **C#** клас.

```
1    <?xml version="1.0" encoding="UTF-8"?>
2    <root>
3        <name type="string">Angel</name>
4        <age type="number">22</age>
5        <universityStudent type="boolean">true</universityStudent>
6    </root>
```

Фигура 4, примерен начин за дефиниране на тип в **XML**.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

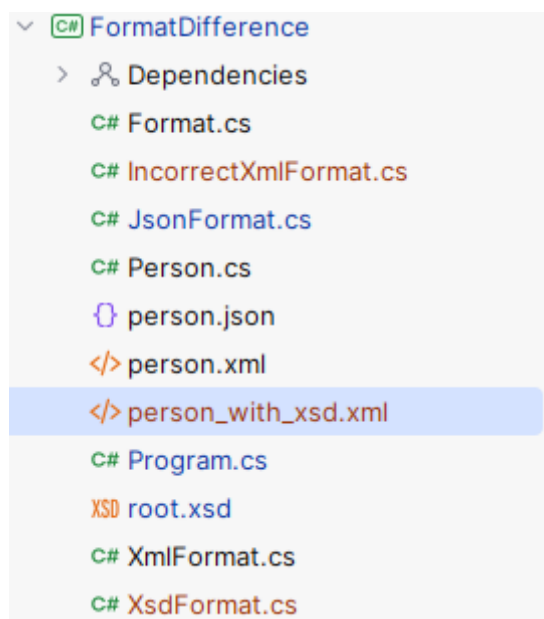
Фигура 5, примерен **XSD** използван за дефиниране на тип.

Разработка

Обща характеристика

За демонстрация на посочената функционалност е разработено конзолно **.NET** приложение. Целта му е да покаже различията на **JSON** и **XML** формата. Съдържа примери как може да се имплементира подобна функционалност на **XML**. Всички **.xml** и **.json** файлове са създадени предварително и могат да бъдат намерени в solution.

Файлова структура



Фигура 6, структура на приложението.

JsonFormat.cs

```
5 ^o public class JsonFormat : Format
6 {
7     private const string JsonPath = "person.json";
8
9     [1 usage 2 Angel L Stoykov *]
9 ^o public override void Display()
10 {
11     var path = Path.Combine(AppContext.BaseDirectory, JsonPath);
12     if (!File.Exists(path))
13     {
14         Console.WriteLine("File not found.");
15         return;
16     }
17
18     var json :string = File.ReadAllText(path);
19
20     using var doc :JsonDocument = JsonDocument.Parse(json);
21     JsonElement root = doc.RootElement;
22
23     foreach (var prop :JsonProperty in root.EnumerateObject())
24     {
25         var value :JsonElement = prop.Value;
26         Console.WriteLine($"{prop.Name} => {value.ValueKind}");
27     }
28 }
29 }
```

Фигура 7, JsonFormat.cs класа

JSON притежава вградена типова система, която разпознава автоматично различните примитивни стойности — низове, числа, булеви стойности и null. Дори без десериализация, библиотеката **System.Text.Json** може да идентифицира всеки тип чрез свойството **ValueKind**.

Прочитаме **.json** файла (фиг. 8) типовете съвпадат с очакваните (фиг. 9).

```
1 {
2     "name": "Angel",
3     "age": 22,
4     "universityStudent": true
5 }
```

Фигура 8, използваният **.json** файл за изпълнението на JsonFormat.cs

```
Name: Angel Type: System.String
Age: 22 Type: System.Int32
Status: True Type: System.Boolean
```

Фигура 9, изходните данни, които се визуализират при изпълнението на JsonFormat.cs

IncorrectXmlFormat.cs

```
public class IncorrectXmlFormat
{
    private const string XmlPath = "person.xml";

    1 usage
    public static void XmlDifference()
    {
        XmlDocument xmlDocument = XmlDocument.Load(XmlPath);
        var root:XElement? = xmlDocument.Root;

        var name = root.Element("name").Value; // "Angel"
        var age:string = root.Element("age").Value; // "10"
        var universityStatus:string = root.Element("universityStudent").Value; // true

        Console.WriteLine($"Before proper deserialization - XML TYPES:");
        Console.WriteLine($"name: {name.GetType()}"); // String
        Console.WriteLine($"age: {age.GetType()}"); // String
        Console.WriteLine($"status: {universityStatus.GetType()}"); // String
    }
}
```

Фигура 10, IncorrectXmlFormat.cs класа

Отново се прочита файл, но този път файлът е **XML** (фиг. 11). Основната разлика е изходните данни. Има съществена разлика, типовете не са очакваните. Вместо да се визуализира System.String, System.Int32, System.Boolean, получените стойности са три пъти System.String (фиг. 12).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <person>
3      <name>Angel</name>
4      <age>22</age>
5      <universityStudent>true</universityStudent>
6  </person>
```

Фигура 11, показва използваният .xml файл за изпълнението на XmlFormat.cs

```
Before proper deserialization - XML TYPES:
name: System.String
age: System.String
status: System.String
```

Фигура 12, полученият резултат е неочакван, различава се по типовете.

XsdFormat.cs

```
2 usages
6 ^o public class XsdFormat : Format
7 {
8     private const string XmlPath = "person_with_xsd.xml";
9     private const string XsdPath = "root.xsd";
10
11 ^o public override void Display()
12 {
13     var xml:string = Path.Combine(AppContext.BaseDirectory, XmlPath);
14     var xsd:string = Path.Combine(AppContext.BaseDirectory, XsdPath);
15
16     var schema = new XmlSchemaSet();
17     schema.Add(targetNamespace: "", schemaUri: xsd);
18
19     var settings = new XmlReaderSettings
20     {
21         Schemas = schema,
22         ValidationType = ValidationType.Schema
23     };
24     settings.ValidationEventHandler += ValidationCallback;
25
26     using var reader = XmlReader.Create(xml, settings);
27     while (reader.Read()) { }
28     Console.WriteLine("XML is valid according to the XSD schema.");
29 }
30
31 1 usage
32 private static void ValidationCallback(object? sender, ValidationEventArgs e)
33 {
34     Console.WriteLine($"Validation error: {e.Message}");
35 }
```

Фигура 13, XsdFormat.cs класа

Ще използваме **schema** (фиг. 14), за да валидираме съдържанието на **XML** файла (фиг. 15). Ако всичко е наред, ще се визуализира ред 28, но ако има проблем, например при тип **number** е въведено “two”, тогава ще се изпълни статичния метод **ValidationCallback**.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3    <xs:element name="root">
4      <xs:complexType>
5        <xs:sequence>
6          <xs:element name="name">
7            <xs:complexType>
8              <xs:simpleContent>
9                <xs:extension base="xs:string">
10                 <xs:attribute name="type" type="xs:string" use="required"/>
11               </xs:extension>
12             </xs:simpleContent>
13           </xs:complexType>
14         </xs:element>
15         <xs:element name="age">
16           <xs:complexType>
17             <xs:simpleContent>
18               <xs:extension base="xs:integer">
19                 <xs:attribute name="type" type="xs:string" use="required"/>
20               </xs:extension>
21             </xs:simpleContent>
22           </xs:complexType>
23         </xs:element>
24         <xs:element name="universityStudent">
25           <xs:complexType>
26             <xs:simpleContent>
27               <xs:extension base="xs:boolean">
28                 <xs:attribute name="type" type="xs:string" use="required"/>
29               </xs:extension>
30             </xs:simpleContent>
31           </xs:complexType>
32         </xs:element>
33       </xs:sequence>
34     </xs:complexType>
35   </xs:element>
36 </xs:schema>

```

Фигура 14, XSD – **schema**, която дефинира три основни типа и **attribute - type**, който е задължително да бъде използван в **.xml** файла.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:noNamespaceSchemaLocation="root.xsd">
4    <name type="string">Angel</name>
5    <age type="number">22</age>
6    <universityStudent type="boolean">true</universityStudent>
7  </root>

```

Фигура 15, **.xml** файл, който използва горепосочената **schema**. По този начин се дефинират **primitive** типове данни.

XmlFormat.cs

```
7 ^o public class XmlFormat : Format
8 {
9     private const string XmlPath = "person.xml";
10
11 ^o public override void Display()
12 {
13     var path = Path.Combine(AppContext.BaseDirectory, XmlPath);
14     if (!File.Exists(path))
15     {
16         Console.WriteLine("File not found.");
17         return;
18     }
19
20     var xmlContent :string = File.ReadAllText(path);
21     using var stringReader = new StringReader(xmlContent);
22
23     var xmlSerializer = new XmlSerializer(typeof(Person));
24     var person = (Person) xmlSerializer.Deserialize(stringReader);
25
26     if (person == null)
27     {
28         Console.WriteLine("Person is null.");
29         return;
30     }
31
32     Console.WriteLine("\nName: " + person.name + " Type: " + person.name.GetType());
33     Console.WriteLine("Age: " + person.age + " Type: " + person.age.GetType());
34     Console.WriteLine("Status: " + person.universityStudent + " Type: " + person.universityStudent.GetType());
35 }
36 }
```

Фигура 16, XmlFormat.cs класа

За да постигнем десериализация ще създаден клас Person.cs, който ще приеме стойностите на **.xml** файла. За целта е използван **XmlSerializer**, който успява да извърши автоматична програмна типизация.

Важно е да се отбележи, че това не е свойство на **XML**, а на **.NET** средата, която извършва типова конверсия въз основа на дефинирания модел.

Цитирани източници

- [1] „Unicode“, [Онлайн]. Available: <https://unicodes.jesetane.com/>.
- [2] „json“, [Онлайн]. Available: <https://www.json.org/json-en.html>.
- [3] „ECMA-404“, December 2017. [Онлайн]. Available: https://ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf.
- [4] D. Crockford, „rfc4627“, Network Working Group, July 2006. [Онлайн]. Available: <https://www.ietf.org/rfc/rfc4627.txt>.
- [5] „what-is-xml“, Amazon AWS, [Онлайн]. Available: <https://aws.amazon.com/what-is/xml/>.
- [6] „SAML“, webopedia, [Онлайн]. Available: <https://www.webopedia.com/definitions/saml/>.
- [7] K. Lane, „SOAP API“, Postman, 28 June 2023. [Онлайн]. Available: <https://blog.postman.com/soap-api-definition/>.
- [8] „storing-primitives“, stackoverflow, [Онлайн]. Available: <https://stackoverflow.com/questions/50183989/storing-primitives-in-xml>.
- [9] „XSD“, microsoft, [Онлайн]. Available: [https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ms764635\(v=vs.85\)](https://learn.microsoft.com/en-us/previous-versions/windows/desktop/ms764635(v=vs.85)).
- [10] „XSD Schema“, w3schools, [Онлайн]. Available: https://www.w3schools.com/xml/schema_intro.asp.
- [11] „JSON primitives“, yugabyte, [Онлайн]. Available: https://docs.yugabyte.com/preview/api/ysql/datatypes/type_json/primitive-and-compound-data-types/.