Практическая работа №7 – Форматы представления данных

Синтаксическая совместимость

Системы Интернета вещей часто характеризуется большим количеством разнородных компонентов, и для корректного взаимодействия этих компонентов (т.е. для обмена данным без потерь и повреждения) необходимо поддерживать синтаксическую совместимость.

Некоторые форматы представления данных особенно хорошо подходят для обмена данными между компонентами системы Интернета вещей и между различными системами. Примером таких форматов представления данных может служить XML (расширяемый язык разметки) и JSON.

XML

XML (англ. eXtensible Markup Language) — расширяемый язык разметки. Рекомендован Консорциумом Всемирной паутины (W3C). Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

XML хранит данные в текстовом формате. Это обеспечивает независимый от программного и аппаратного обеспечения способ хранения,
транспортировки и обмена данными. XML также облегчает расширение или обновление до новых операционных систем, новых приложений или новых браузеров без потери данных.

Преимущества XML:

Поддержка метаданных

Одним из самых больших преимуществ XML является то, что метаданные возможно помещать в теги в форме атрибутов (в JSON атрибуты будут добавлены как другие поля-члены в представлении данных, которые НЕ могут быть желательны).

Визуализация браузера

Большинство браузеров отображают XML в удобочитаемой и организованной форме. Древовидная структура XML в браузере позволяет пользователям естественным образом сворачивать отдельные элементы дерева.
Эта функция будет особенно полезна при отладке.

Поддержка смешанного контента

Хорошим вариантом использования XML является возможность передачи смешанного контента в пределах одной и той же полезной нагрузки данных.
Этот смешанный контент четко различается по разным тегам.

Введение в XML

Простейший XML-документ выглядит следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<device category="sensors">
   <title lang="en">humidity sensor</title>
   <floor>2</floor>
   <room>203</room>
</book>
```

Первая строка — это XML декларация. Здесь определяется версия XML (1.0) и кодировка файла. На следующей строке описывается корневой элемент документа `<device>` (*открывающий тег*). Следующие 4 строки описывают дочерние элементы корневого элемента (`title, floor, room`). Последняя строка определяет конец корневого элемента `</device>` (*закрывающий тег*).

Документ XML состоит из *элементов* (elements). Элемент начинается *открывающим тегом* (start-tag) в угловых скобках, затем идет *содержимое* (content) элемента, после него записывается *закрывающий тег* (end-teg) в угловых скобках.

Информация, заключенная между тегами, называется содержимым или значением элемента: `<floor>2</floor>`. Т.е. элемент `floor` принимает значение `1`. Элементы могут вообще не принимать значения.

Элементы могут содержать *атрибуты*, так, например, открывающий тег `<title lang="en">` имеет атрибут `lang`, который принимает значение `en`. Значения атрибутов заключаются в кавычки (двойные или ординарные).

Структура XML

XML документ должен содержать корневой элемент. Этот элемент является «родительским» для всех других элементов.

Все элементы в XML документе формируют иерархическое дерево. Это дерево начинается с корневого элемента и разветвляется на более низкие уровни элементов.

Все элементы могут иметь подэлементы (дочерние элементы):

```
<корневой>
   <потомок>
     <подпотомок>.....</подпотомок>
   </потомок>
</корневой>
```

Правила синтаксиса (Валидность)

Структура XML документа должна соответствовать определенным правилам. XML-документ, отвечающий этим правилам, называется *валидным*

(*англ.* Valid — правильный) или *синтаксически верным*. Соответственно, если документ не отвечает правилам, он является *невалидным*.

Основные правила синтаксиса XML:

1. Теги XML регистрозависимы. Так, тег `<Device>` не то же самое, что тег `<device>`.

Открывающий и закрывающий теги должны определяться в одном регистре:

```
<Message>Это неправильно</message>

<message>Это правильно</message>
```

2. XML элементы должны соблюдать корректную вложенность:

```
<b><i>Некорректная вложенность</b></i>

<b><i>Корректная вложенность</i></b>
```

3. У XML документа должен быть корневой (или родительский) элемент.

4. Значения XML атрибутов всегда должны заключаться в кавычки:

```
<note date="12/11/2007">Корректная запись</note>

<note date=12/11/2007>Некорреткная запись</note>
```

Сущности

Некоторые символы в XML имеют особые значения и являются служебными. При размещении таких символов внутри XML элементов будет сгенерирована ошибка.

Чтобы ошибки не возникали, служебные символы необходимо заменять на соответствующие им сущности. В XML существует 5 предопределенных сущностей.

Таблица 7. Предопределенные сущности

XML

Сущность	Симв	Значение
<	<	Меньше, чем
>	>	Больше, чем
&	&	амперсанд
'	'	апостроф
"	"	кавычки

Кодировки

И еще один важный момент, который стоит рассмотреть — кодировки. Существует множество кодировок. Самыми распространенными кириллическими кодировками являются Windows-1251 и UTF-8.

В XML файле кодировка объявляется в декларации:

```
<?xml version="1.0" encoding="windows-1251"?>
```


JSON

JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми.

Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

Преимущества JSON

1. Более подробная структурная информация в документе

XML требует открытия и закрытия тегов, а JSON использует пары имя / значение, четко обозначенные «{«и»}» для объектов, «[«и»]» для массивов, «,» (запятую) для разделения пары и «:» (двоеточие) для отделения имени от значения.

Можно легко различить число 1 и строку "1", поскольку числа, строки (и логические значения) представлены по-разному в JSON.

Можно легко различать отдельные элементы и коллекции одного размера (используя массивы JSON).

2. Малый размер сообщения

При одинаковом объеме информации JSON почти всегда значительно меньше, что приводит к более быстрой передаче и обработке.

3. Близость к javascript

JSON является подмножеством JavaScript, поэтому код для его анализа и упаковки вполне естественно вписывается в код JavaScript.

4. Проще представить значение null

Синтаксис и структура JSON

Объект JSON представлен в формате «ключ-значение» и записывается в фигурных скобках. При сохранений объекта, он записывается и хранится в с расширением .json.

Пример JSON-объекта:

```
{"room" : "First",

"humidity" : 105
"temperature" : 23.1,
"cond_state" : true,}
```

Объект JSON может содержать множество строк, а также вложенные объекты.

Каждый объект состоит из пар ключ:значение. Каждая пара разделена запятой, каждый объект заключен в фигурные скобки.

Ключом может быть любая валидная строка, помещенная в двойные кавычки. Все ключи одного объекта обязательно должны быть уникальны. Не рекомендуется использовать символ пробела в названии ключа, лучше заменить его символом нижнего подчеркивания.

Значениям могут выступать как простые типы данных:

- строки (заключенные в кавычки);

- числа;
- логические данные (true или false);
- ноль,

Так и сложные типы данных (например, вложенными объектами или массивами JSON).

Как правило, данные в файлах .json записываются в столбик для удобочитаемости файла, однако JSON можно записать и в строку:

```
{ "first_name" : "John", "last_name": "Smith", "online" : true, }
```

Главным преимуществом JSON является то, что данные в этом формате поддерживают многие популярные языки программирования, потому их можно быстро передать.

Сложные типы в JSON

JSON может хранить вложенные объекты и массивы, которые будут передаваться в качестве значения присвоенного им ключа.

Вложенные объекты

Ниже приведен пример, в котором содержатся данные о датчиках. Для каждого датчика (“humidity”, “temperature”, “illuminance”,) в качестве значения передаётся вложенный объект, который, в свою очередь, тоже состоит из ключей и значений.

Первый вложенный объект JSON выделен красным.

```
{
    "humidity": {
        "place": "room1",
        "value": 345,
        "time": "2021.08.29 23:11:34",
        "state": true
    },
    "temperature": {
        "place": "room1",
        "value": 21,
        "time": "2021.08.29 23:11:24",
```



```
        "state": true
    },
    "illuminance" : {
        "place": "room2",
        "value": 432,
        "time": "2021.08.29 23:11:35",
        "state": true
    },
}
```

Вложенные массивы

В качестве значений в JSON-объектах можно использовать массивы JavaScript – упорядоченные наборы данных, заключенные в квадратные скобки, которые могут содержать данные различных типов.

Массив используют для передачи большого количества данных, которые можно сгруппировать. Для примера попробуем записать данные о показаниях температуры в различные моменты времени.

```
{
    "sensor" : "temperature",
    "location" : "room3",
    "data" : [
        {
            "temperature" : 23.1,
            "time"    : "2021.08.29 23:11:23",
        },
        {
            "temperature" : 23.0,
            "time"    : "2021.08.29 23:11:33",
        },
        {
            "temperature" : 22.9,
            "time"    : "2021.08.29 23:11:43",
        },
        {
            "temperature" : 22.5,
            "time"    : "2021.08.29 23:11:54",
        },
    ],
}
```

При помощи вложенных массивов и объектов можно создать сложную иерархию данных. **Работа с XML и JSON**

Для работы с объектами XML и JSON существуют стандартные библиотеки.

Например, для языка программирования Python это:

- Модуль json: https://docs.python.org/3/library/json.html
- Модуль ElementTree: https://docs.python.org/3/library/xml.etree.elementtree.html

Задание практической работы №7 Часть 1

1. С компьютера в аудитории или личного устройства подпишитесь на несколько MQTT-топиков **в составе стенда WB-demo-kit v.2**, согласно вариантам.

Таблица 8-1. Варианты для выполнения практической работы № 7

№ варианта	Датчики
1	1. Датчик температуры устройства WB-MSW v.3 (5) 2. Датчик движения устройства WB-MSW v.3 (5) 3. Датчик шума устройства WB-MSW v.3 (5) 4. Датчик освещенности устройства WB-MS v.2 (12)
2	1. Датчик шума устройства WB-MSW v.3 (5) 2. Датчик освещенности устройства WB-MS v.2 (12) 3. Датчик CO2 устройства WB-MSW v.3 (5) 4. Датчик температуры 1-wire DS18B20 (2)
3	1. Датчик температуры 1-wire DS18B20 (1) 2. Датчик шума устройства WB-MSW v.3 (5) 3. Датчик CO2 устройства WB-MSW v.3 (5) 4. Качество воздуха VOC устройства WB-MS v.2 (12)
4	1. Датчик движения устройства WB-MSW v.3 (5) 2. Датчик шума устройства WB-MSW v.3 (5) 3. Датчик освещенности устройства WB-MS v.2 (12) 4. Датчик температуры устройства WB-MS v.2 (12)
5	1. Датчик CO2 устройства WB-MSW v.3 (5) 2. Датчик шума устройства WB-MSW v.3 (5) 3. Датчик освещенности устройства WB-MS v.2 (12) 4. Датчик температуры устройства WB-MSW v.3 (5)
6	1. Датчик температуры 1-wire DS18B20 (2) 2. Датчик влажности устройства WB-MSW v.3 (5) 3. Качество воздуха VOC устройства WB-MS v.2 (12) 4. Датчик шума устройства WB-MSW v.3 (5)
7	1. Датчик движения устройства WB-MSW v.3 (5) 2. Качество воздуха VOC устройства WB-MS v.2 (12) 3. Датчик влажности устройства WB-MSW v.3 (5) 4. Датчик температуры 1-wire DS18B20 (1)

2. На любом языке программирования реализуйте программу (скрипт), которая бы каждые 5 секунд упаковывала последние полученные данные в файлы формата JSON и XML. В одной записи должно быть 6 полей: 4

показаний датчиков, время формирования файла, номер чемодана (последние две цифры IP-адреса).

3. На любом языке программирования реализуйте программу-парсер, которая бы выводила в консоль данные, полученные из сгенерированных в п.2 файлов.

В отчете приведите:

1. Команды для подписки на топики
2. Структуру данных, получаемых от стенда с описанием каждого поля. Пример структуры данных, получаемых от физического объекта представлен на рис. 29.

```
id: int
temperature: float
humidity: float
name: string
```

Рисунок 29. Пример структуры данных

3. Листинг с комментариями с указанием языка программирования для пунктов 2 и 3 задания.
4. Скриншоты результатов работы скриптов пунктов 2 и 3 задания
5. Краткое описание сторонних библиотек и используемых из них функций, использованных для реализации заданий пунктов 2 и 3.

Задание практической работы №7 Часть 2.

С компьютера в аудитории или личного устройства подпишитесь на несколько MQTT-топиков **в составе стенда WB-demo-kit v.3,** согласно вариантам.

Таблица 8-2. Варианты для выполнения практической работы № 7

№ варианта	Датчики
1	1. Датчик температуры устройств1-wire DS18B20 (ВК1) 2. Датчик VOC устройства WB-MSW v.4 (А6) 3. Напряжение на любом устройстве стенда
2	1. Датчик уровня шума устройства WB-MSW v.4 (А6) 2. Датчик освещенности устройства WB-MS v.4 (А6) 3. Напряжение на любом устройстве стенда

3	1. Датчик температуры устройств1-wire DS18B20 (ВК2) 2. Датчик CO2 устройства WB-MSW v.4 (A6) 3. Напряжение на любом устройстве стенда
4	1. Датчик движения устройства WB-MSW v.4 (A6) 2. Датчик температуры устройства WB-MS v.4 (A6) 3. Напряжение на любом устройстве стенда
5	1. Датчик CO2 устройства WB-MSW v.4 (A6) 2. Датчик освещенности устройства WB-MS v.4 (A6) 3. Напряжение на любом устройстве стенда
6	1. Датчик влажности устройства WB-MSW v.4 (A6) 2. Датчик движения устройства WB-MSW v.4 (A6) 3. Напряжение на любом устройстве стенда
7	1. Датчик освещенности устройства WB-MS v.4 (A6) 2. Датчик VOC устройства WB-MSW v.4 (A6) 3. Напряжение на любом устройстве стенда

Далее, как и в части 1 реализуем в части 2, п.2,п.3,п.2,п.3,п.4,п.5

Дополнительное задание практической работы №7

Часть 1.

1. Приведите структуру данных, получаемых от физического устройства разрабатываемой системы (или mock-объекта, его заменяющего), с указанием типов данных и описанием каждого поля. Пример структуры данных, получаемых от физического объекта представлен на рис. 30.

```
id: int
temperature: float
humidity: float
name: string
```

Рисунок 30. Пример структуры данных

2. На любом языке программирования реализуйте программу (скрипт), которая бы генерировала JSON-файл на основе данных из предыдущего пункта задания. Для демонстрации работы скрипта достаточно считать вводимые пользователем в консоли показатели.

3. На любом языке программирования реализуйте программу-парсер, которая бы выводила в консоль данные, полученные из сгенерированного в п.2 файла.

В отчете приведите структуру данных, листинги с комментариями, указанием языка программирования и результат выполнения пунктов 2 и 3.

Часть 2.

Соберите физическое устройство или реализуйте на любом языке программирования его программный эмулятор. Эмулятор должен генерировать поток данных и выводить их в консоли.

В отчете приведите схему подключения и фотографии собранного устройства для физического устройства или листинг кода с комментариями его программного эмулятора.