

Вебинар

Шибков Константин
17 октября 2023



JSON & XML туда и обратно

Skillbox



Пишу какой-то код на Python/PHP более 10 лет

Пишу на Java 4 года

Пишу код для СДЭК

Пишу в блоге sendel.ru, телеграм канале
@tree_monitors

Пишу материалы для обучения в Skillbox

Провожу встречи клуба @JavaKeyFrames

Соорганизатор Agile Ufa Meetup

JSON

JavaScript Object Notation



JSON всегда рядом с нами

1. Нажмите F12 в браузере Chrome/Firefox
2. Зайти во вкладку Network/Сеть
3. Перейдите на skillbox.ru
4. Смотрите Preview ответов на запросы

Если у вас веб-приложение

=

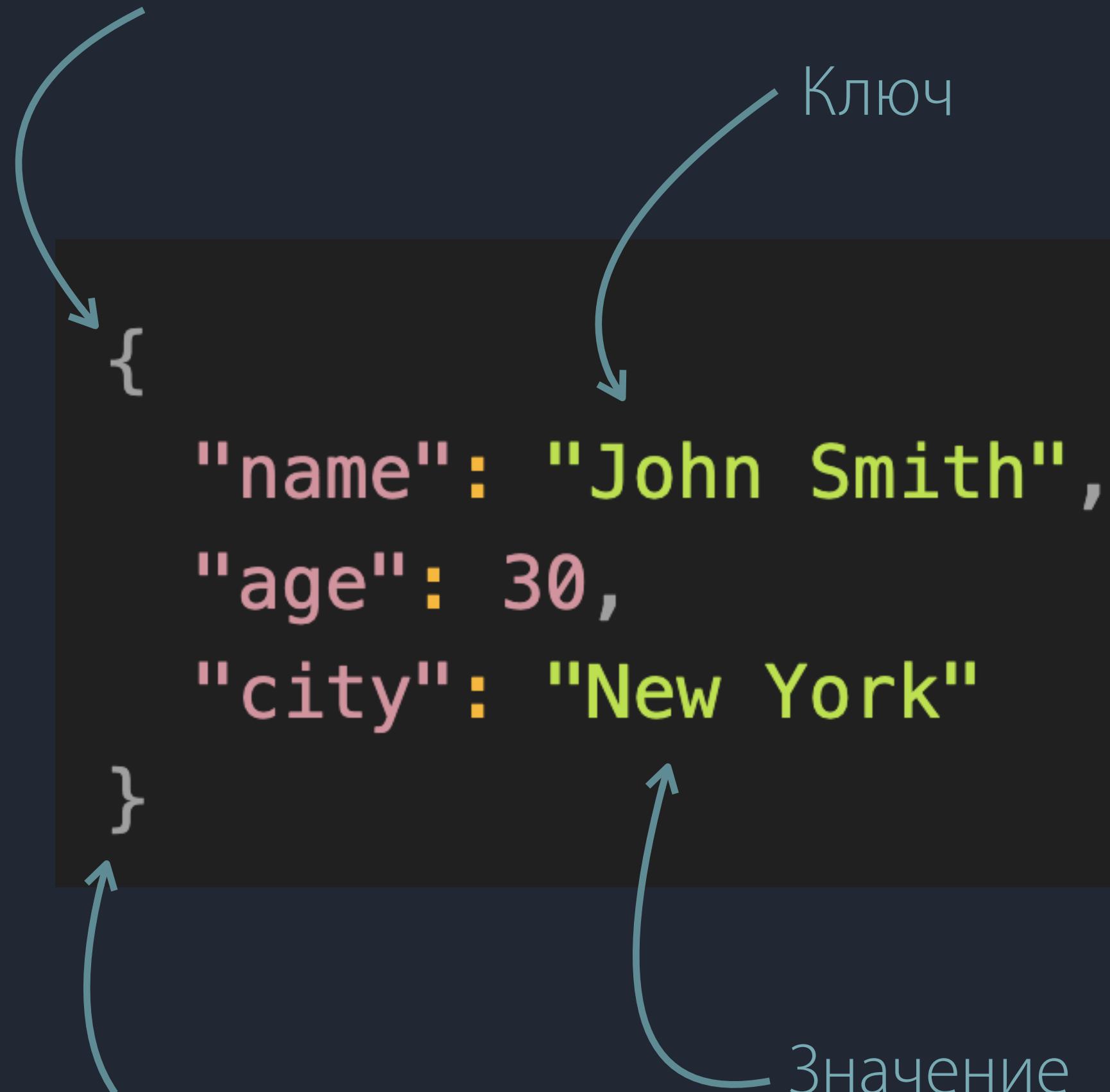
вы точно работаете с JSON

Если у вас есть Бд
=
скорее всего вы и
там используете JSON

Почему JSON хорош?

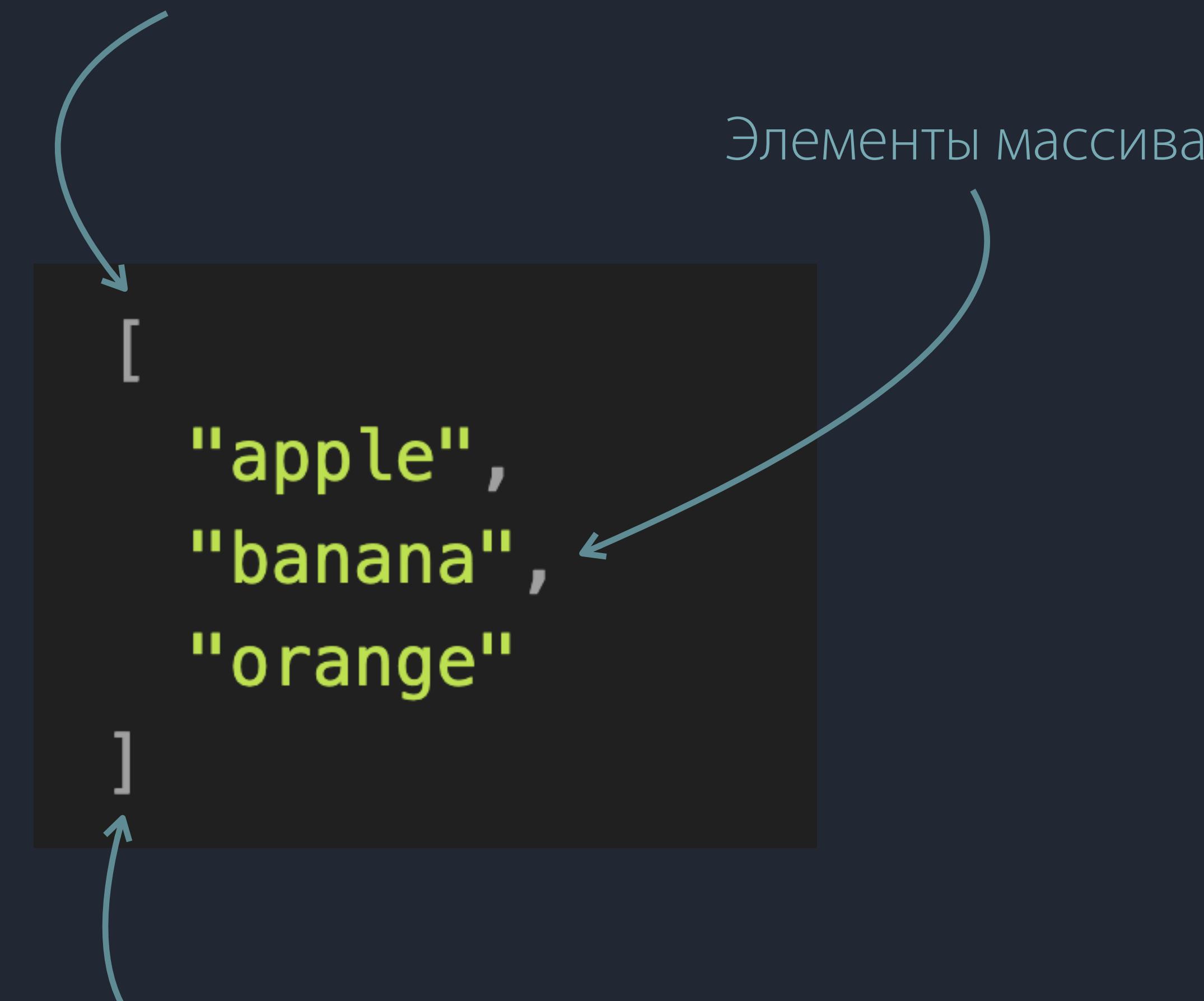
1. Универсальность
2. Структурированность
3. Читаемость
4. Поддержка

Начало объекта



Объект

Начало массива



Массив

Объект

начинается с { и заканчивается }

неупорядоченные коллекции пар "ключ-значение"

ключи это строки в кавычках

значения могут быть объектами, массивами,
строками, числами, boolean, null

ключи и значения разделяются двоеточием, пары
"ключ-значение" разделяются запятой

Объект

```
{  
  "name": "John Smith",  
  "age": 30,  
  "city": "New York"  
}
```

```
{  
  "apple":  
  {  
    "weight": 34.5,  
    "color": "#ff0000"  
  }  
}
```

Массив

упорядоченные списки значений

начинается массив с [и заканчивается]

элемент массива может быть объектом,
массивом, строкой, числом, boolean, null

элементы в массиве разделяются запятой

Массив

```
[  
  "apple",  
  "banana",  
  "orange"  
)
```

```
[1, true, {}, "name"]
```

```
[]
```

Типы данных JSON

числа: 1 1 . 35 0 3 . 7e-5 -56 . 78

строки: "Иван" "" "4566"

boolean: true false

null: null

Начало объекта

```
{  
    "name": "John Smith",  
    "age": 30,  
    "city": "New York",  
    "isEmployed": true,  
    "languages": ["Java", "JavaScript", "Python"],  
    "address": {  
        "street": "123 Main St",  
        "city": "San Francisco",  
        "country": "USA"  
    },  
    "projects": [  
        {  
            "name": "Project A",  
            "startDate": "2021-01-01",  
            "endDate": null,  
            "isCompleted": false  
        },  
        {  
            "name": "Project B",  
            "startDate": "2022-03-15",  
            "endDate": "2022-06-30",  
            "isCompleted": true  
        }  
    ]  
}
```

Массив languages

Объект address

Массив объектов projects

Конец объекта

Викторина валидности JSON

```
{  
  "alex" : {  
    "year": 20  
  }  
}
```

🚫 данные формата JSON должен быть заключены в {} или []

```
[  
  true,  
  1,  
  {}  
]
```

✓ массив может быть разнородный

```
{ "langs":  
  [  
    "ru", "en"  
  ]  
}
```

🚫 одинарные кавычки не допускаются

Викторина валидности JSON

```
{  
    "RU": ["ru", "kz"],  
    "KK": ["kz"]  
}
```

{}

[]

✓ объект с полями, можно
представить как
`Map<String, List<String>>`

✓ пустой объект

✓ пустой массив

Викторина валидности JSON

```
{  
  "chars": ["a", "b"]  
}
```

```
[  
  {"lang": "Java"}  
]
```

```
[{}, {}]
```

🚫 в объекте у каждого поля должно быть имя

✅ массив с одним элементом

🚫 два объекта не могут быть рядом без массива

JSON ↔ POJO

Любой JSON можно представить в виде Java объекта или структуры данных и наоборот

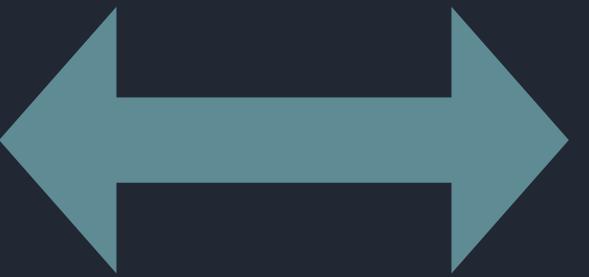
```
{  
  "name": "John Smith",  
  "age": 30,  
  "city": "New York"  
}
```



```
public record Person(  
    String name,  
    Integer age,  
    String city) {  
}
```

JSON ↔ POJO

```
{  
  "apple":  
  {  
    "weight": 34.5,  
    "color": "#ff0000"  
  }  
}
```



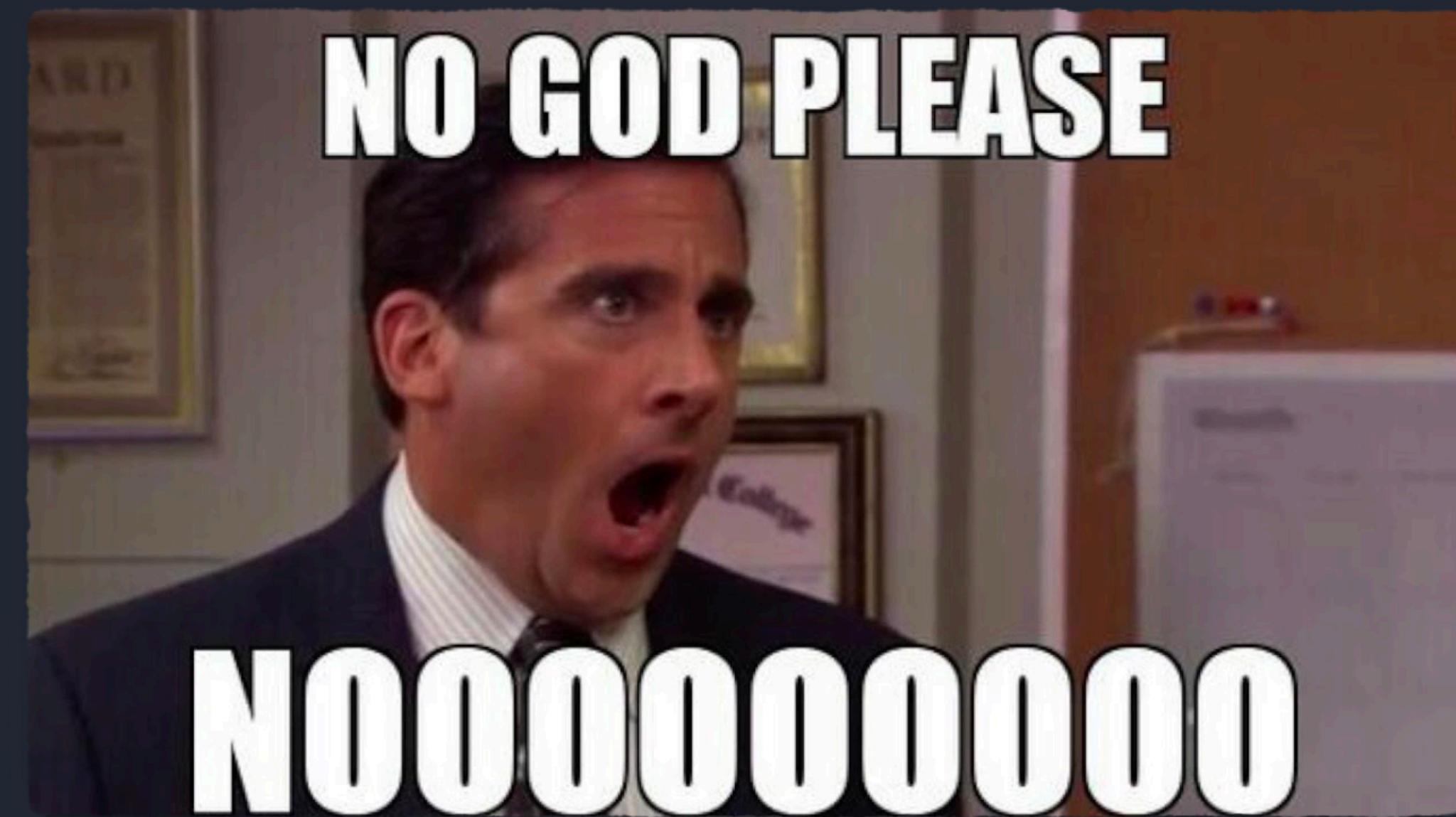
```
record Basket(Fruit apple) {}  
  
record Fruit(Double weight, String color) {}
```

JSON ↔ POJO

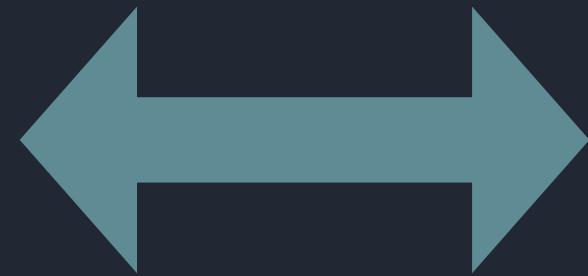
```
[1, true, {}, "name"]
```



```
List<Object> objects;
```



```
{  
  "name": "John Smith",  
  "age": 30,  
  "city": "New York",  
  "isEmployed": true,  
  "languages": ["Java", "JavaScript", "Python"],  
  "address": {  
    "street": "123 Main St",  
    "city": "San Francisco",  
    "country": "USA"  
  },  
  "projects": [  
    {  
      "name": "Project A",  
      "startDate": "2021-01-01",  
      "endDate": null,  
      "isCompleted": false  
    },  
    {  
      "name": "Project B",  
      "startDate": "2022-03-15",  
      "endDate": "2022-06-30",  
      "isCompleted": true  
    }  
  ]  
}
```



```
public record Person (  
  String name,  
  Integer age,  
  String city,  
  Boolean isEmployed,  
  List<String> languages,  
  Address address,  
  List<Project> projects  
) {}
```

```
public record Address (  
  String street,  
  String city,  
  String country  
) {}
```

```
public record Project (  
  String name,  
  LocalDate startDate,  
  LocalDate endDate,  
  Boolean isCompleted  
) {}
```

Основные библиотеки для JSON



JACKSON

```
// Создание объекта ObjectMapper  
ObjectMapper objectMapper = new ObjectMapper();  
  
// Преобразование объекта в JSON  
String json = objectMapper.writeValueAsString(obj);  
  
// Преобразование JSON в объект  
MyClass obj = objectMapper.readValue(json, MyClass.class);
```

`@JsonProperty("Date")`

GSON

```
// Создание объекта Gson  
Gson gson = new Gson();  
  
// Преобразование объекта в JSON  
String json = gson.toJson(obj);  
  
// Преобразование JSON в объект  
MyClass obj = gson.fromJson(json, MyClass.class);
```

`@SerializedName("jsonField")`

JACKSON

для парсинга дат

```
@JsonProperty("Timestamp")
ZonedDateTime timestamp,
```

и зависимость для java.time

```
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
    <version>2.15.2</version>
</dependency>
```

GSON

для парсинга дат нужен
кастомный сериализатора

```
public class ZonedDateTimeAdapter implements JsonSerializer<ZonedDateTime>, JsonDeserializer<ZonedDateTime> {
    2 usages
    private static final DateTimeFormatter formatter = DateTimeFormatter.ISO_ZONED_DATE_TIME;

    • Konstantin Shibkov
    @Override
    public JsonElement serialize(ZonedDateTime src, Type typeOfSrc, JsonSerializationContext context) {
        return new JsonPrimitive(formatter.format(src));
    }

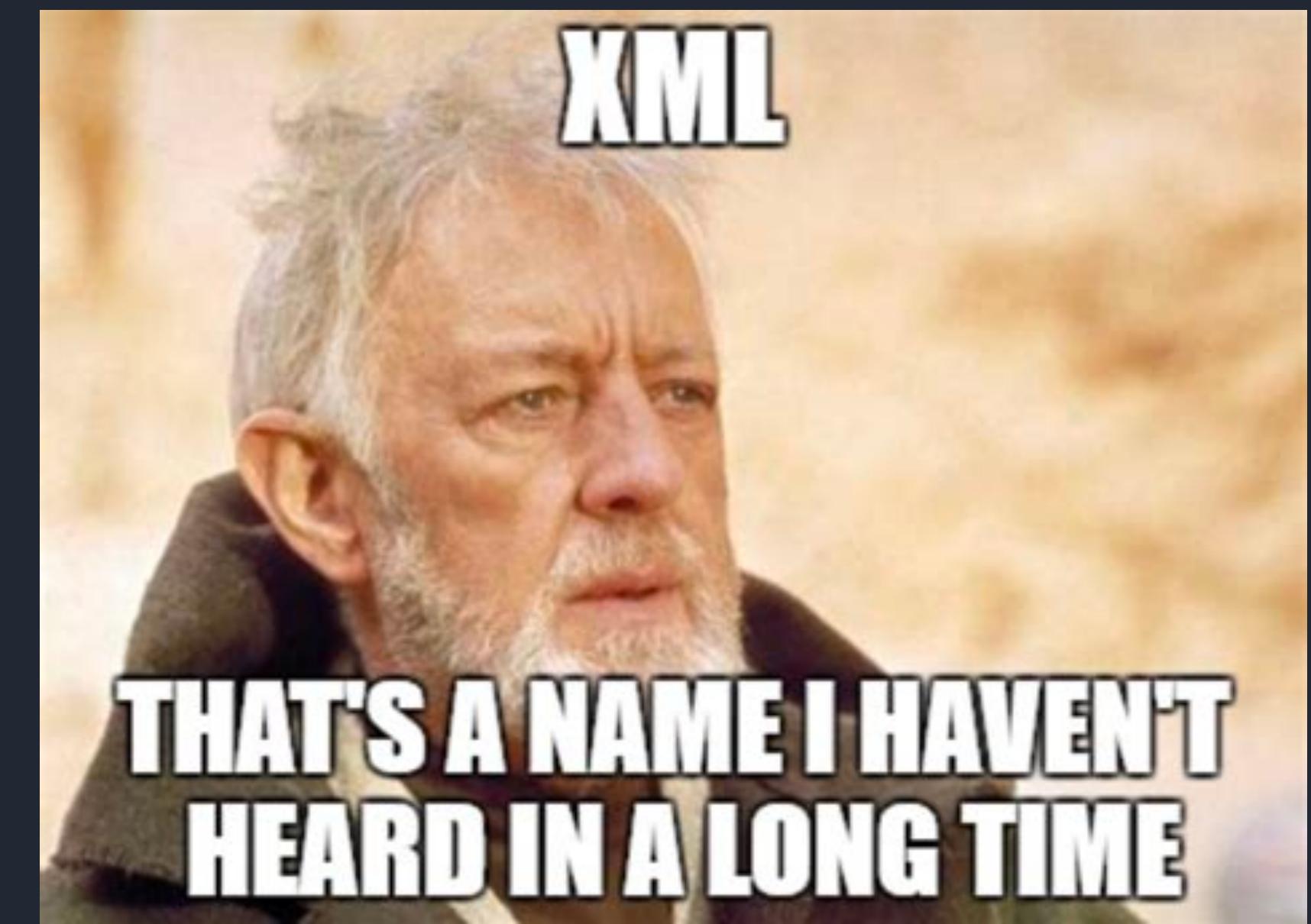
    • Konstantin Shibkov *
    @Override
    public ZonedDateTime deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context)
        throws JsonParseException {
        return ZonedDateTime.parse(json.getAsString(), formatter);
    }
}
```

и регистрация в билдере

```
Gson gson = new GsonBuilder()
    .registerTypeAdapter(ZonedDateTime.class, new ZonedDateTimeAdapter())
    .create();
```

XML

eXtensible
Markup
Language



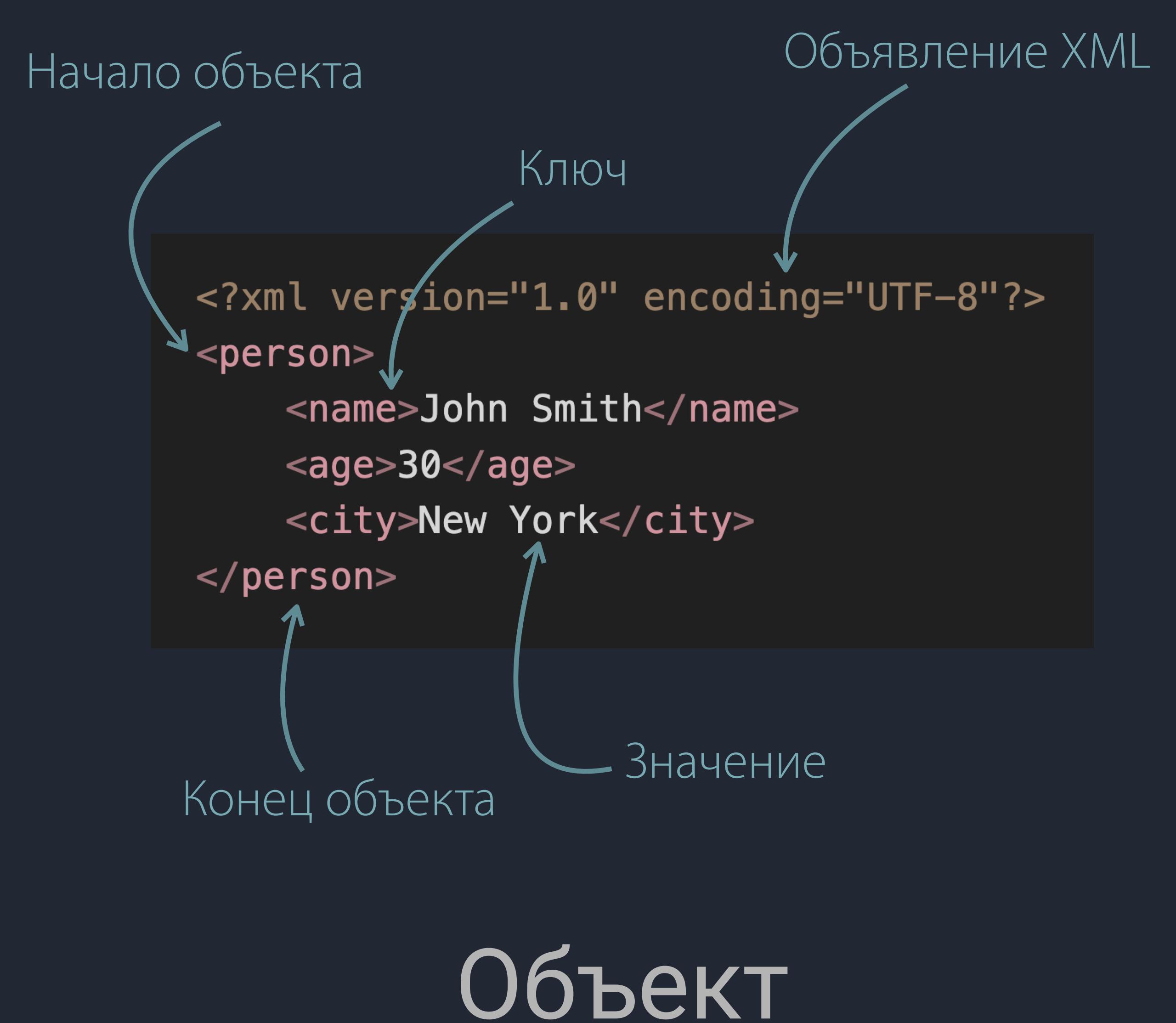
Почему XML хорош?

1. Универсальность
2. Структурированность
3. Читаемость
4. Поддержка

видите SOAP



работаете с XML



Только один
корневой элемент

```
<? xml version="1.0" encoding="UTF-8"?>
<person>
    <name>John Smith</name>
    <age>30</age>
    <city>New York</city>
</person>
<fruit>
    <name>Apple</name>
</fruit>
```

🚫 два корневых элемента

Теги могут содержать
атрибуты

```
<?xml version="1.0" encoding="UTF-8"?>
<book id="1">
    <title>Война и мир</title>
    <author>Лев Толстой</author>
    <year>1869</year>
</book>
```

XML ↔ POJO

```
<?xml version="1.0" encoding="UTF-8"?>
<person>
    <name>John Smith</name>
    <age>30</age>
    <city>New York</city>
</person>
```



```
public record Person(String name, int age, String city) {
```

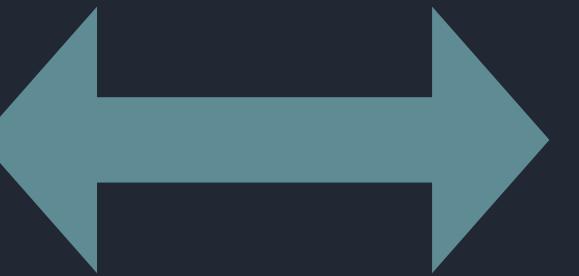
XML ↔ POJO

```
<?xml version="1.0" encoding="UTF-8"?>
<book id="1">
    <title>Война и мир</title>
    <author>Лев Толстой</author>
    <year>1869</year>
</book>
```



```
public record Book(String id, String title, String author, int year) {}
```

```
{  
  "name": "John Smith",  
  "age": 30,  
  "city": "New York",  
  "isEmployed": true,  
  "languages": ["Java", "JavaScript", "Python"],  
  "address": {  
    "street": "123 Main St",  
    "city": "San Francisco",  
    "country": "USA"  
  },  
  "projects": [  
    {  
      "name": "Project A",  
      "startDate": "2021-01-01",  
      "endDate": null,  
      "isCompleted": false  
    },  
    {  
      "name": "Project B",  
      "startDate": "2022-03-15",  
      "endDate": "2022-06-30",  
      "isCompleted": true  
    }  
  ]  
}
```



```
<root>  
  <name>John Smith</name>  
  <age>30</age>  
  <city>New York</city>  
  <isEmployed>true</isEmployed>  
  <languages>  
    <language>Java</language>  
    <language>JavaScript</language>  
    <language>Python</language>  
  </languages>  
  <address>  
    <street>123 Main St</street>  
    <city>San Francisco</city>  
    <country>USA</country>  
  </address>  
  <projects>  
    <project>  
      <name>Project A</name>  
      <startDate>2021-01-01</startDate>  
      <endDate/>  
      <isCompleted>false</isCompleted>  
    </project>  
    <project>  
      <name>Project B</name>  
      <startDate>2022-03-15</startDate>  
      <endDate>2022-06-30</endDate>  
      <isCompleted>true</isCompleted>  
    </project>  
  </projects>  
</root>
```

JSON

легковесный синтаксис,
основанный на объектах
и массивах

XML

Синтаксис

использует разметку с
открывающими и закрывающими
тегами, более многословен

Размер файла

меньше, так как меньше
служебной информации

больше, XML-файлы содержат
много дополнительных символов

JSON

XML

Типы данных

поддерживает простые типы данных, такие как строки, числа, логические значения, массивы и объекты

не определяет явно типы данных, и данные обычно представлены как текстовые строки.

Расширяемость

не предоставляет встроенных механизмов структуры данных, что делает его менее расширяемым для сложных сценариев

его структура может быть легко расширена путем добавления новых элементов и атрибутов

JSON

не предоставляет встроенной поддержки наследования

XML

Поддержка наследования

позволяет определять схемы с поддержкой наследования, что полезно в некоторых сценариях, таких как описание данных с разными уровнями детализации

Простота парсинга

парсинг данных обычно более простой и эффективный

требует более сложных алгоритмов и может быть медленнее

Когда использовать XML

XML прекрасно подходит для представления структурированных документов, шаблоны со сложной структурой.

Если вам необходимо определить сложную схему данных с поддержкой наследования и типов данных. XML схемы или DTD (Document Type Definition)

Интеграция с legacy. Когда у вас нет другого выбора.

Использование SOAP.
Снова нет выбора.

Когда использовать JSON

Веб-разработка и API, так как
быстрее обработка и передача.



Для хранения структурированных
данных внутри приложения, базы
данных, конфигураций

В большинстве случаев
предпочтительный
формат передачи
текстовых данных.

Не успел задать вопрос?
Пишите в телегу!

@SENDEL



@THREE_MONITORS

