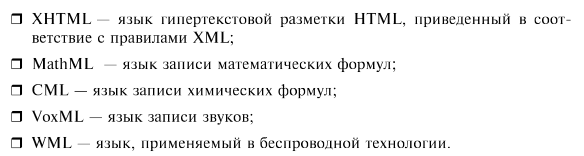
**XML** (eXtensible Markup Language) – расширяемый язык разметки.

Он не содержит готового набора тегов, а описывает правила создания своих собственных тегов.

Есть множество реализаций XML



Сам язык XML задает только общие правила, по которым создаются теги и оформляется документ XML.

**СТРУКТУРА XML ДОКУМЕНТА**

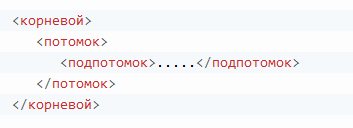
1. Объявление XML (XML Declaration) либо пролог, может отсутствовать.



1. Объявление типа **DTD (Document Type Definition).**  Внутри тега <!DOCTYPE> объявляются теги, используемые в документе, или приводится ссылка на файл с объявлениями.



1. Корневой элемент, который является родительским для всех ругих элементов. Один документ XML можно вложить в другой, корневой элемент используется чтобы не нарушить структуру. Имя корневого элемента указывается в DOCTYPE и считается именем всего документа.



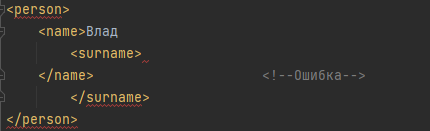
Документ XML состоит из элементов. Элемент состоит из открывающегося тега, содержимого и закрывающегося тега. XML регистрозависим.



XML требует закрывающие теги. Исключение составляют элементы с пустым содержимым.



Элементы могут быть вложенными. Таким образом документ представляет собой древовидную структуру. Нельзя чтобы элементы пересекались:



Секция **CDATA** используется для указания строки текста. Все спец-символы внутри (< > “”) не будут парситься. CDATA нельзя вкладывать друг в друга.



**АТРИБУТЫ**

У открывающихся тегов XML могут быть атрибуты, в виду *имя=”значение”*. Значение нужно обязательно заключать в кавычки.



В атрибутах желательно указывать информацию, не являющуюся частью данных (Метаданные).

При использовании атрибутов возникают некоторые проблемы:

* атрибуты не могут содержать множественные значения (элементы могут)
* атрибуты не могут содержать древовидные структуры (элементы могут)
* атрибуты сложно расширять (для будущих изменений)

**ПРОСТРАНСТВА ИМЕН**

Для предотвращения конфликтов тегов с одинаковыми именами используются пространства имен. Для этого имена тегов и атрибутов снабжают кратким префиксом, который отделяется от имени двоеточием.

Префикс имени связывается с идентификатором, образующим пространство имен. Имена тегов и атрибутов в пространстве имен должны быть уникальными.

Префикс и идентификатор пространства имен определяются атрибутом *xmlns:*

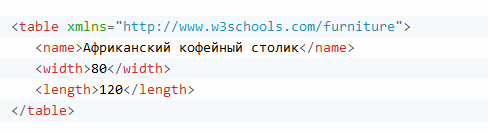




Имя с префиксом называется уточненным. Часть записанная после двоеточия – локальной частью.

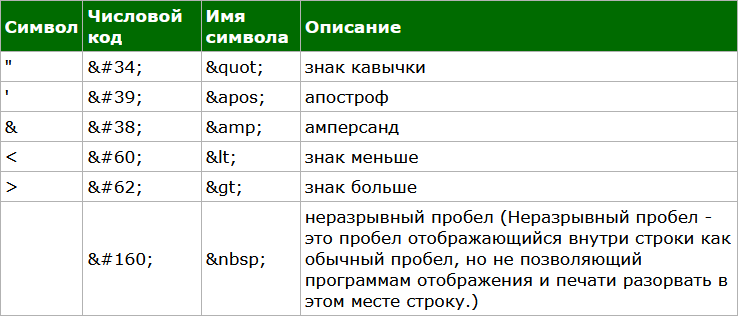
URI пространства имен не используется парсером. Его цель состоит в том, чтобы дать пространству имен уникальное имя. Однако компании часто используют существующие веб-страницы с информацией об этом пространстве имен.

Пространство имен по умолчанию позволяет не указывать префикс перед каждым именем:



Атрибуты не входят в пространство имен по умолчанию. Если без префикса, то атрибут не относится ни к одному пространству имен.

**ОСНОВНЫЕ ЭКРАНИРУЕМЫЕ СИМВОЛЫ**



**DTD**

* Объявление элемента

*<!ELEMENT br EMPTY>*  - пустой элемент

*<!ELEMENT name ANY> -* элемент может содержать что угодно

*<!ELEMENT name (#PCDATA)>* - элемент с текстовым содержимым и без вложенных элементов.

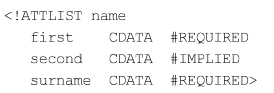
*<!ELEMENT np:name (np:firstchild\*, np:secondchild)\*> -* Элемент со вложенными элементами. Звездочка возле элемента – элемент м. б. указан 0 и более раз. Если за скобками, то все элементы м. б. указаны 0 и более раз.

Есть и другие знаки: «+» - 1 и более раз, «?» - 0 или 1 раз. Элемент без знаков должен быть указаан ровно 1 раз.

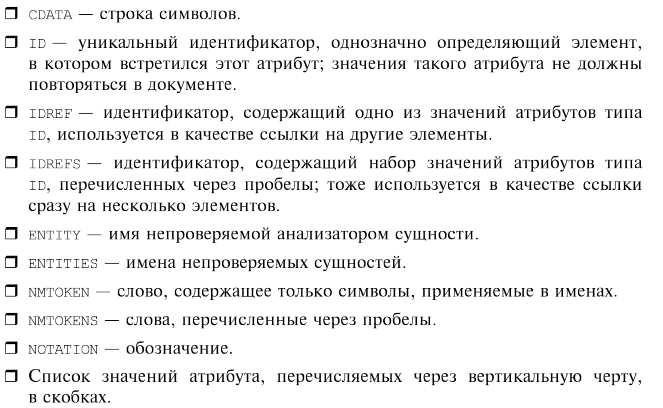
*<!ELEMENT name (#PCDATA, ch1, ch2)> -* Элемент в который можно вложить и текст и другие элементы.

*<!ELEMENT name (ch1 | ch2)>* - нужно указать либо первый, либо второй вложенный элемент (или).

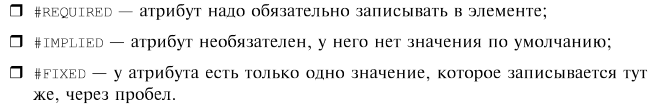
* Атрибуты объявляются одним списком, сразу после объявления самого элемента. Элементы имеют тип.



Типы аргументов:



В конце объявления атрибута указывается его обязательность



**Недостатки DTD**

* Нельзя указать точное количество повторений вложенных элементов.
* Нельзя указать точный тип содержимого элементов (например дату)

**XSD (XML Schema Definition Language)**

Схема XML сама записывается на языке XML. Вносит типизацию элементов, позволяет указывать точное кол-во вложенных элементов, легко расширяется и т.д.

Корневой элемент схемы носит название *schema.* XSD выделяет 2 типа элементов:

* *Простые* – элементы без атрибутов и вложенных элементов;
* *Сложные* – с атрибутами и/или вложенными элементами.

XSD gозволяет создавать свои типы, расширяя или сужая уже существующие. Базовым элементом в XSD является тип *anyType*, от него наследуются все типы.

Имена элементов и атрибутов, используемые используемые в XSD, определены в пространстве имен с идентификатором <http://www.w3.org/2001/XMLSchema>, и обычно используются с префиксом *xs* или *xsd*.

Для подключения схемы к документу используют

<note xmlns="http://msiter.ru"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://msiter.ru note.xsd">

**Встроенные простые типы XSD**

* *Вещественные числа:* decimal, float, double.
* *Целые числа:* integer (как подтип decimal без дробной части), long, int, short, byte, nonPositiveInteger, negativeInteger, nonNegativeInteger, positiveInteger, unsignedLong, unsignedInt, unsignedShort, unsignedByte.
* *Строки:* string, normalizedString (не содержит \n, \r, \t), token(нет \n, \r, \t, пробелов в начале и конце строки, несколько идущих подряд пробелов)
* *Дата и время*: duration (промежуток времени *P1Y2M3DT10H30M45S*): Y – год, M – месяц и т.д.;

dateTime (Дата и время в формате YYYY-MM-DDThh:mm:ss)

time (hh:mm:ss)

date (YYYY-MM-DD)

gYearMonth (YYYY-MM)

gMonthDay (-MM-DD)

gYear (YYYY)

gMonth (-MM-)

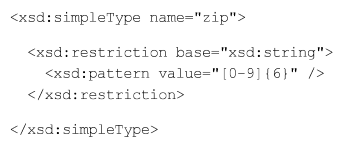
gDay (-DD)

* *Двоичные типы*: hexBinary (16-ричная форма), base64Binary
* *Прочие типы*: anyURI, QName (Имя в XML вместе с префиксом), NOTATION (запись математических, химических символов, нот и т.д.)

Можно с помощью встроенных типов определить **новые простые типы**.



Например создать почтовый адрес как 6 цифр



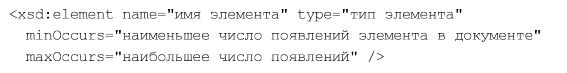
* Список – несколько значений через пробел одного типа



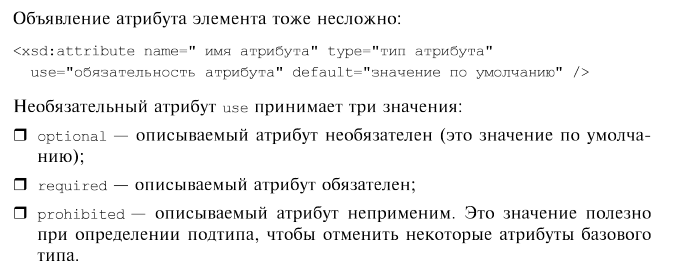
Создается:



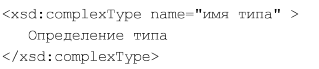
**Объявление элементов и их атрибутов в XSD**



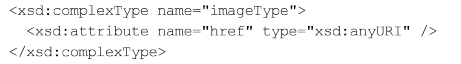
*minOccurs* и *MaxOccurs* по умолчанию = 1.



Сложный тип определяется компонентом complexType:



* Пустой тип, только с атрибутами:



**Включение файлов схемы в другую схему**

В создаваемую схему можно включить файлы, содержащие другие схемы с помощью *include* и *import*.



Include используем если схема файла из того же пространства имен (Например хотим разбить большую схему на 2 файла). Если включаемый файл из другого пространства имен, то надо использовать import.

**XLink**

Специальный язык создания ссылок, является одной из реализаций XML. Он определяет только атрибуты, которые используются для создания ссылок.

Особенности Ссылки XLink:

* Может указывать сразу на несколько документов.
* Можно создать ссылку в одном, а использовать в других документах.
* Сослаться можно не только на XML, но и на любой ресурс: изображение, чертеж, программу.
* Возможность организовать обратные ссылки.
* Возможность создать сложные ссылки (Ссылки, которые содержат в себе другие ссылки).

В языке XLink объявлено 10 атрибутов:

* *type* – тип ссылки;
* *href* – адрес ресурса, с которым связана ссылка;
* *show* – определяет способ показа полученного ресурса;
* *actuate* – устанавливает момент активизации ссылки (*onload* – при загрузке, *onRequest* – по какому-то событию, *other* – другой способ);
* *label –* позволяет пометить ссылку, чтобы к ней можно было обратится из других ссылок.
* *from, to* – указывают начальные и конечные пункты ссылки. Используются в дугах (arc);
* *role* – указывает на ресурс, описывающий ссылку;
* *arcrole –* то же, что и role но для ссылок arc и simple.
* *title –*  устанавливает заголовок ссылки. Предназначена для человека, играет роль комментария, поясняя действие ссылки или отмечая способ ее активизации.

**Типы ссылок** (Значения атрибута type)

* *none* – элемент не является ссылкой;
* *locator* – удаленный информационный ресурс;
* *simple* – похожа на ссылку html. Связывает один документ с другим, или другой частью того же самого документа;
* *resource* – локальный ресурс, чаще всего текущий XML документ, в котором записана ссылка;
* *arc* – дуга. Ссылка между текущим ресурсом и внешним (Входящая и выходящая), либо между двумя внешними ресурсами (сторонняя).

**XPOINTER**

Уточненная ссылка на часть ресурса. Не поддерживается браузерами.

Метки элементов в XML документе создаются с помощью атрибутов ID. Ссылки на помеченные элементы указываются атрибутами **IDREF** или **IDREFS.**

**XPOINTER –** не является реализацией XML. Он задает правила записи меток и обращения к ним с помощью ссылок языка XLink.

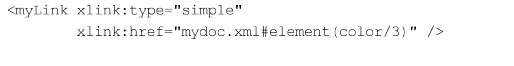
XPointer для ссылки на id использует синтаксис, похожий на HTML.



Есть еще указатели, основанные на **схеме:**



1. найти элемент section, который вложен в chapter, который вложен в book.
2. Найти третий по счету элемент, вложенный в color. Используется следующим образом:

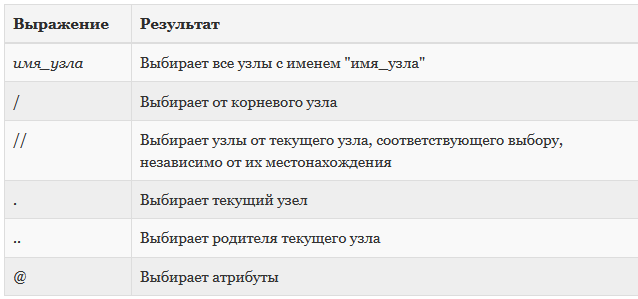


**XPath**

Используется для навигации по элементам и атрибутам XML документа. Он представляет документ в виде дерева.

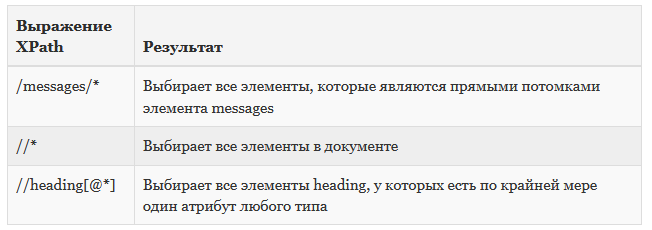
Чтобы выбрать узлы и наборы узлов в XPath используются выражения пути. Так же имеется большое количество функций для работы со строковыми, числовыми и логическими значениями, функции для сравнения даты и времени, для манипуляции узлами и последовательностями и др.

**Наиболее полезные выражения пути:**



**Предикаты** позволяют найти конкретный узел или узел с конкретным значением, всегда заключаются в квадратные скобки.

Чтобы найти неизвестные заранее узлы XML документа, XPath позволяет использовать специальные символы:



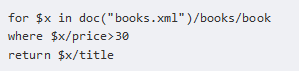
Подробнее почитать про XPath: <https://msiter.ru/tutorials/xpath>

**XQuery**

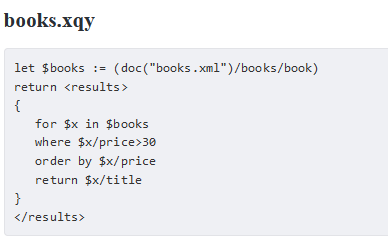
Функциональный язык, который используется для извлечения информации, хранящейся в формате XML. Построен на основе XPath, использует XPath выражения для навигации по документам XML. Аналог SQL для БД.

Практически все инструменты имеют поддержку XQuery, а ЯП -библиотеки.

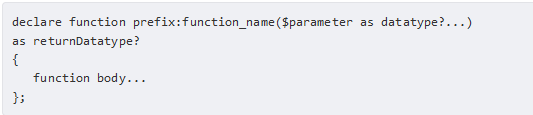
Синтаксис:

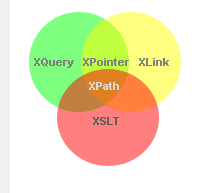


* **For** – Выбирает коллекцию всех узлов.
* **Let** – Помещает результат в переменную XQuery.
* **Where** – Выбирает узлы, соответствующие условию.
* **Order** – Упорядочивает узлы, указанные в соответствии с критериями.
* **Return** – возвращает окончательный результат.



XQuery позволяет создавать последовательности, использовать регулярные выражения, конструкцию if-then-else, объявлять собственные функции:





**XSLT -** это XML язык, который может использоваться для преобразования документов XML в другие форматы, например, HTML. Включает в себя XPath и XQuery.

При помощи XSLT можно добавлять/удалять элементы и атрибуты в конечный файл, реорганизовывать и сортировать элементы, выполнять тесты, определять, какие элементы скрыть или отобразить и др.

Корневым элементом, декларирующим документ таблицы стилей XSL, является **<xsl:stylesheet>** или **<xsl:transform>**.

Элемент **<xsl:value-of>** используется для извлечения значения из отобранного XML элемента.

<td><xsl:value-of select="catalog/cd/title"/></td>

Элемент <xsl:for-each> позволяет организовывать циклы в процессе XSLT преобразования.

<xsl:for-each select="catalog/cd">

Элемент **<xsl:sort>** используется для сортировки выходных данных.

<xsl:sort select="artist"/>

Элемент **<xsl:if>** позволяет задать проверку на соответствие определенному условию содержимое XML файла.

<xsl:if test="price &gt; 10">

**ТИПЫ ПАРСЕРОВ XML**

**DOM parser**

Он загружает весь XML-контент в память в виде древовидной структуры. Чтобы получить содержимое перебираем узлы.

**SAX Parser**

Не загружает полный XML в память, а анализирует XML строку за строкой, вызывая различные события, когда встречает различные элементы. И мы обрабатываем эти события. Более эффективное управление памятью по сравнению с DOM.

**YAML**

Язык сериализации данных (не является языком разметки).

* YAML удобен для восприятия человеком.
* Независимость от языка программирования. Поддерживает списки и словари.
* Однозначность.
* Высокая скорость обработки.
* Чувствителен к регистру.

Используется в:

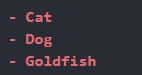
* Файлах конфигурации
* При обмене сообщениями между процессами
* Файлах журналов

YAML поддерживает три структуры данных:

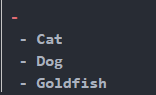
* Объект (ассоциативный массив, словарь ) - набор пар ключ-значение
* Массив – набор значений, упорядоченных по порядку.
* Скаляры – единое неделимое значение, поддерживаются следующие типы:
* Строка
* Логическое значение – true, false
* Целое число
* Плавающая точка – 12.30
* Null – «~»
* Время - 2001-12-14t21:59:43.10-05:00
* Дата - 1976-07-31

**Синтаксис**

* Отступы(пробелы) используются для обозначения структуры (аля питон)
* Члены списка обозначаются дифисом «-», либо заключены в квадратные скобки и разделены запятыми «[el, el2, el3]»



Вложенный массив обозначается дополнительным дефисом



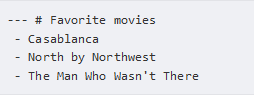
* Ассоциативные массивы представляются в виде пары ключ значение, разделенных двоеточием и заключаются в фигурные скобки

«{name: John Smith, age: 33}»





* Несколько документов с отдельными потоками разделяются 3 дефисами «---»



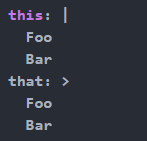
* Комментарии обозначаются решеткой «#», и должны отделяться от основного текста по крайней мере одним пробелом. Комментарии только однострочные
* Для приведения типа данных используются два восклицательных знака



* По умолчанию строки не заключаются в кавычки. Если строка содержит пробелы или специальные символы, тогда необходимы кавычки. Могут использоваться как одинарные, так и двойные кавычки



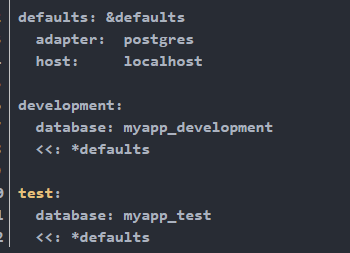
* Для многострочного текста используйте «|» либо «>». Первый сохраняет переход на новую строку (\n), второй \n заменяет пробелом.



Результат: { this: 'Foo\nBar\n', that: 'Foo Bar\n' }

«|» может использоваться с «+» и «-». По умолчанию в конце добавляется один \n. Со знаком + в конце сохраняются все \n. Со знаком минус \n в конец строки не добавляются.

* Можно встраивать один объект в другой. Мы можем пометить объект меткой «&» и использовать его внутри другого с помощью «\*»



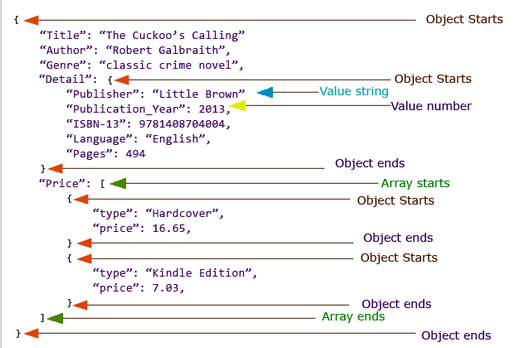
**JSON**

JSON: JavaScript Object Notation. Легкий текстовый открытый формат обмена данными. Полностью независим от языка.   
В основном используется для сериализации и передачи данных по сетевому соединению между сервером и веб-приложениями.

Существует 4 основных типа данных в JSON:

* Строки
* Числа
* Логические значения
* Null

Объекты (список пар ключ: значение, ключ всегда строка) заключаются в фигурные скобки «{}», массивы (список значений )в квадратные «[]». Массивы и объекты могут быть вложенными.



**JSONP**

**JSON SCHEMA**

Так же как и XML shema позволяет описать структуру json документа.

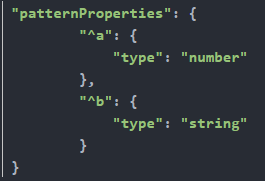
Она позволяет:

* Ограничить тип данных для элементов документа JSON.
* В зависимости от типа проверяемых данных, также могут быть применены дополнительные правила.



Ключевые слова для описания json shema:

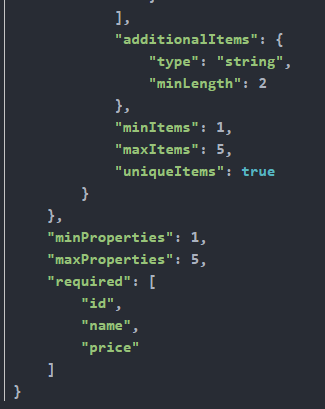
* **"$schema" –** информация о версии схемы JSON.
* **"title" –** заголовок элемента.
* **"description" –** описание элемента.
* **"type" –** Ограничение типа данных элемента. Имеет следующие значения: object, array, integer, number, null, Boolean, string.
* **"properties" –** определяет различные ключи и типы их значений, которые будут использоваться в файле.
* **"required" –** список обязательных элементов.
* **"minProperties", "maxProperties" -** min и max количество вложенных элементов.
* **patternProperties –** вложенный элемент, ключ которого удовлетворяет регулярке



**Пример схемы с различными ключевыми словами**

Подробнее про ключевые слова https://russianblogs.com/article/12801344871/

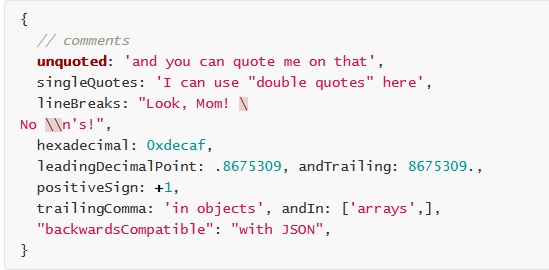




**JSON5**

Список фич, которых нет в JSON, но есть в JSON5

* Запятая после последнего элемента в объекте
* Запятая после последнего элемента в массиве
* Строки могут заключатся в одинарные кавычки
* Строки могут быть многострочными, перенос должен экранироваться \
* Строки могут включать экранированные символы
* Числа могут быть шестнадцатеричными
* Числа могут иметь ведущую или завершающую десятичную точку
* Числа могут быть положительной, отрицательной бесконечностью и NaN
* Числа могут начинаться с явного знака плюс
* Допускаются однострочные и многострочные коментарии



**JSONP**

JSONP — это метод передачи данных JSON, не заботясь о кросс-доменных ограничениях.

Запрос данных с сервера, находящегося в другом домене, может вызвать определенные проблемы из-за политики ограничения домена.

Запрос же внешних скриптов с сервера, находящегося в другом домене, не вызывает таких проблем.

JSONP пользуется этим преимущество, запрашивая файлы при помощи тега **<script>** вместо объекта **XMLHttpRequest**.



С указанного адреса нам возвращается не чистый json, а json обернутый в вызов функции.

jsonObj = '{"name":"John", "age":30, "city":"New York"}';

myFunc(jsonObj)

В результате возвращается вызов функции с именем "myFunc" с данными JSON в качестве параметра. Такая функция должна существовать на клиенте, и обрабатывать json.