Библиотека libjson-c

COLLABORATORS			
	TITLE : Библиотека libjson-c		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Сергей Ларионов	27 октября 2015	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1.0	27 октября 2015		СЛ

Содержание

Список иллюстраций

JSON (JavaScript Object Notation) простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования JavaScript, определенного в стандарте ”ECMA 262 3rd Edition December 1999”. Сам JSON описан в стандарте ”ECMA 404 The JSON Data Interchange Standard”.

1 Введение в JSON

JSON - текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам C подобных языков, таких как C, C\++, C#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными.

В основе стандарта JSON лежат два базовых понятия:

1. Коллекция пар ключ/значение. В разных языках, эта концепция реализована как *объект*, запись, структура, словарь, хэш, именованный список или ассоциативный массив.
2. Упорядоченный список значений. В большинстве языков это реализовано как массив, вектор, *список* или последовательность.

Это универсальные структуры данных. Почти все современные языки программирования поддерживают их в какой либо форме. Логично предположить, что формат данных, независимый от языка программирования, должен быть основан на этих структурах.

2 Типы данных JSON

json_type_null Указатель со значением NULL;

json_type_boolean Булевский тип, принимающий только два значения: `true` и `false`;

json_type_double Число с плавающей точкой;

json_type_int Целое число;

json_type_object Объект, содержащий пары ’key:value`;

json_type_array Массив (возможно - разнотипных) элементов, с доступом по индексу.

json_type_string Строка.

В нотации JSON это выглядит так:

Объект - неупорядоченный набор пар ключ/значение. Объект начинается с { (*открывающей фигурной скобки*) и заканчивается } (*закрывающей фигурной скобкой*). Каждое имя сопровождается : (*двоеточием*), пары ключ/ значение разделяются , (*запятой*) .

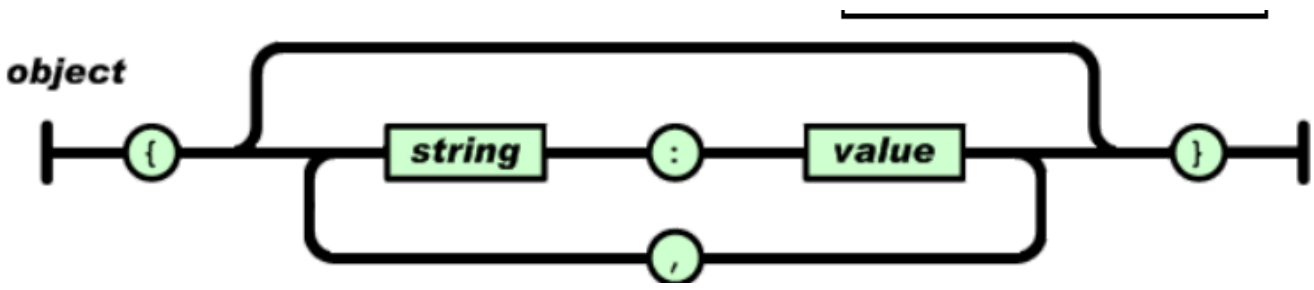

Рис. 1: Синтаксическая диаграмма понятия ”Объект”

Массив - упорядоченная коллекция значений. Массив начинается с [(*открывающей квадратной скобки*) и заканчивается] (*закрывающей квадратной скобкой*). Значения разделены , (*запятой*).

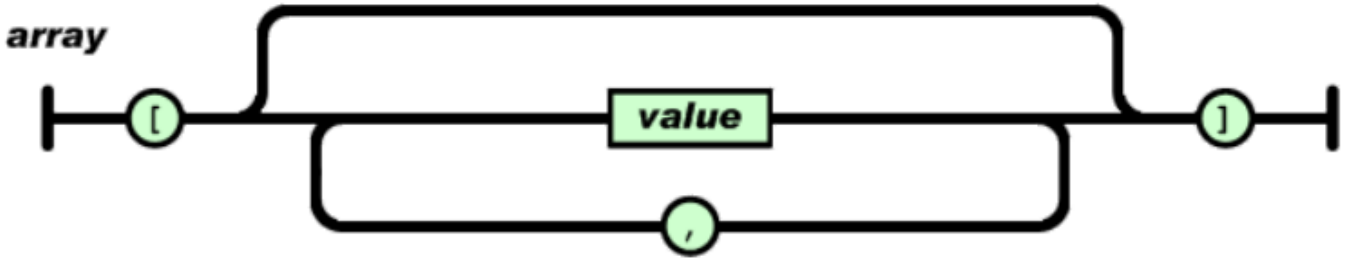

Рис. 2: Синтаксическая диаграмма понятия ”Массив”

Значение - может быть *строкой* в двойных кавычках, *числом, true, false, null*, *объектом* или *массивом*. Эти структуры могут быть вложенными.

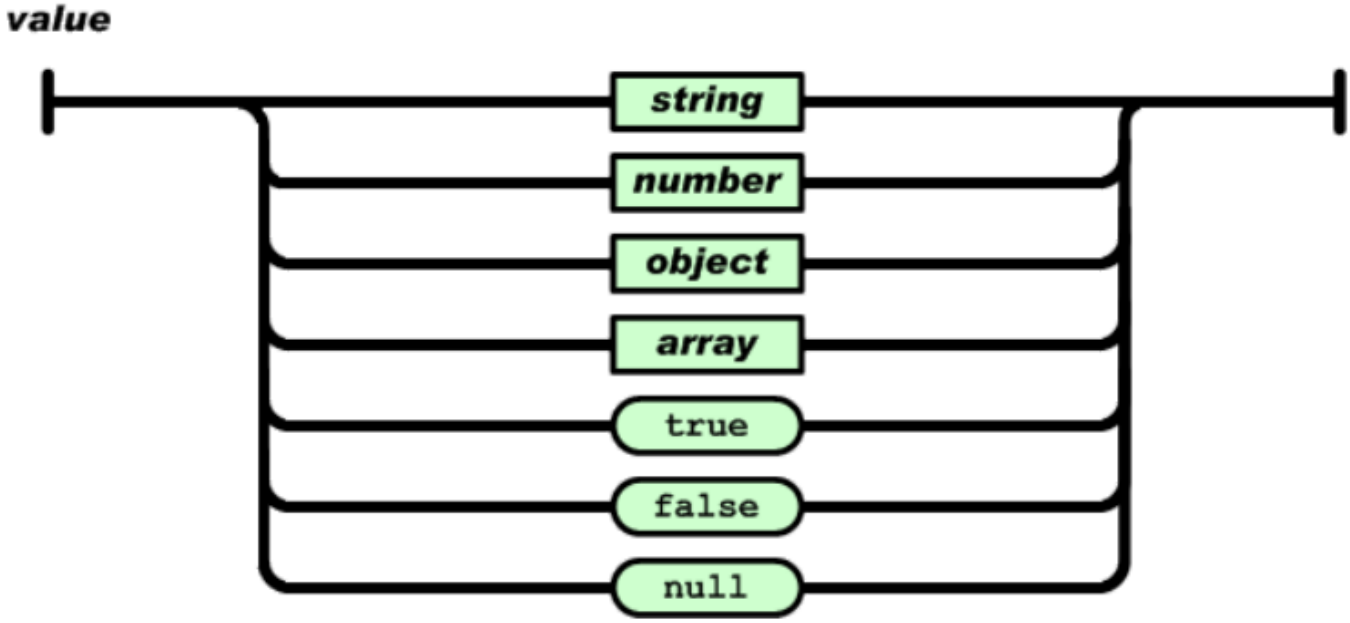

Рис. 3: Синтаксическая диаграмма понятия ”Значение”

Строка - коллекция нуля или больше символов Unicode, заключенная в двойные кавычки, используя \ (*обратную косую черту*) в качестве символа экранирования. Символ представляется как односимвольная строка. Похожий синтаксис используется в C и Java.

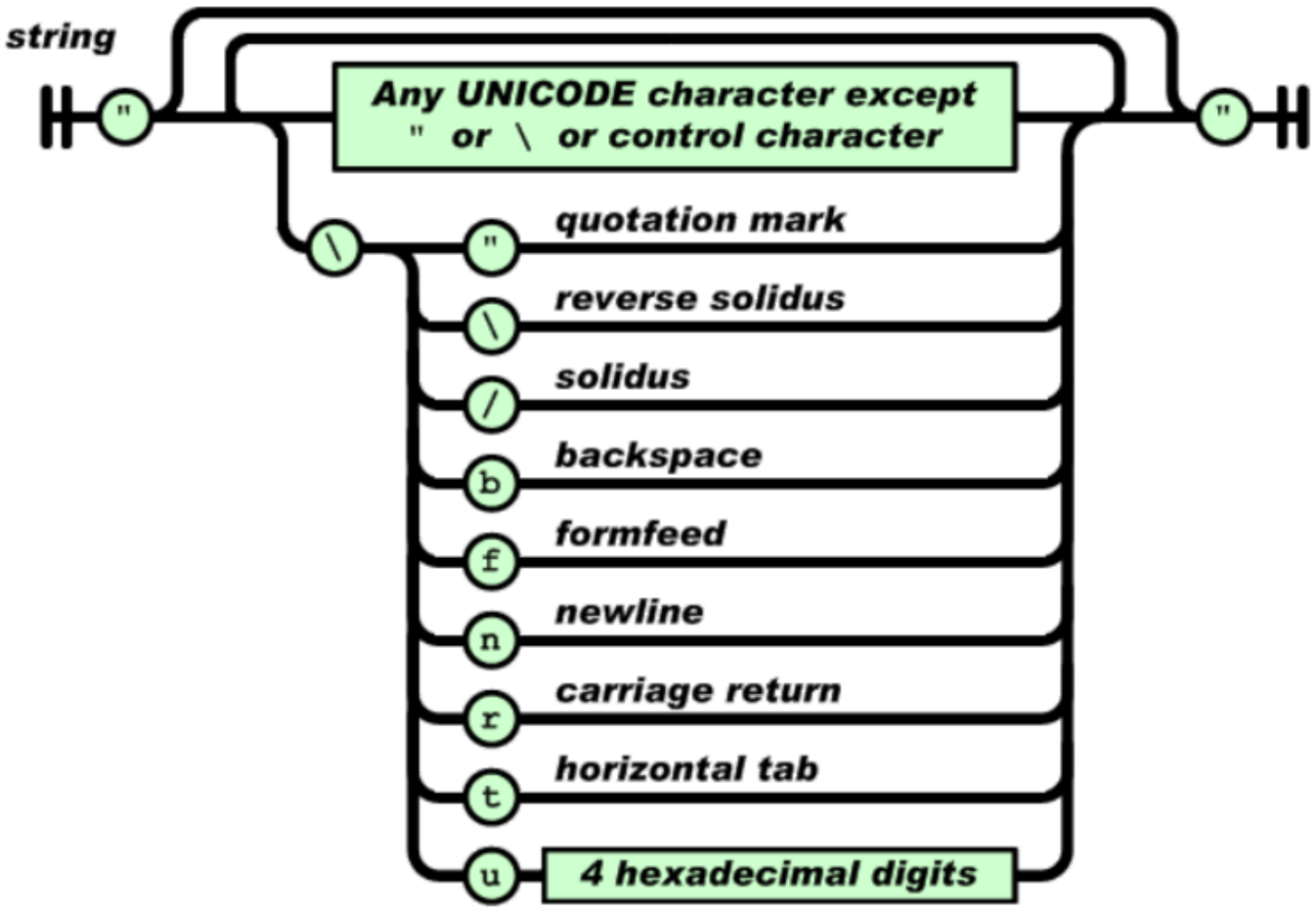

Рис. 4: Синтаксическая диаграмма понятия ”Строка”

Число - представляется так же, как в C или Java, кроме того, что используется толко десятичная система счисления.

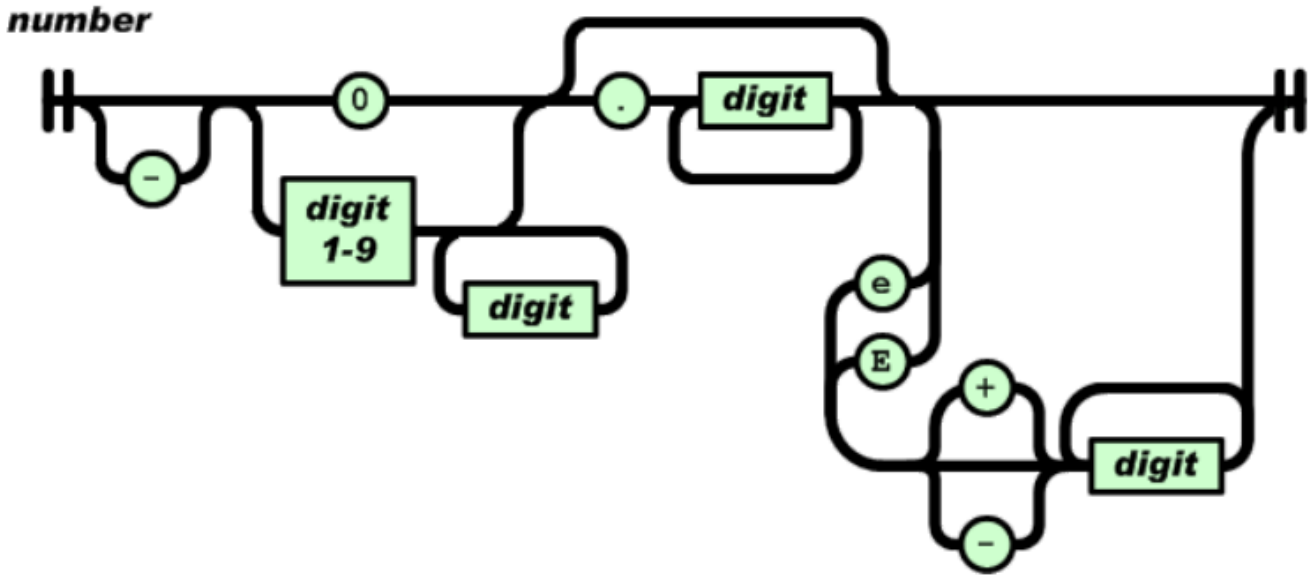

Рис. 5: Синтаксическая диаграмма понятия ”Число”

3 Сохранение данных в файле

Для того, что бы сохранить данные в файле (или передать их каким-либо другим способом) необходимо создать объект *JSON*, содержащий все эти данные, а затем выполнить сериализацию этого объекта в файл.

3.1 Создание простых объектов

3.1.1 Строка

Функции для создания строковых объектов

```
struct json_object* json_object_new_string(const char *s)
struct json_object* json_object_new_string_len(const char *s, int len)
```

Создаёт новый пустой объект `json_object` типа `json_type_string`. Выделяет память для данных `json_object` и копирует значение строки. Если задан параметр `len`, проверяет длину строки.

Параметры s Строка.

Возвращает Объект `json_object` типа `json_type_string`.

3.1.2 Целое

Функции для создания объектов целого типа

```
struct json_object* json_object_new_int(int32_t i)
struct json_object* json_object_new_int64(int64_t i)
```

Создаёт новый пустой объет `json_object` типа `json_type_int`. Необходимо отметить, что значение сохраняется во внутренней 64-битной переменной. Для того, что бы быть уверенным в том, используется действительно 64-битное значение, лучше использовать `json_object_new_int64`.

Параметры i целое

Возвращает Объект `json_object` типа `json_type_int`

3.1.3 Вещественное

Функции для создания объектов вещественного типа

```
struct json_object* json_object_new_double(double  d)
struct json_object* json_object_new_double(double  d, const char * ds))
```

Создаёт новый объект `json_object` типа `json_type_double`, используя точное сериализованное представлегние.

Это позволяет числа, которые отображаются неэффективно (например: 12.3 ⇒ ”12.300000000000001”) сохранять в более подходящем виде.

Замечание
Это используется в `json_tokener_parse_ex()` позволяя выполнять точную десериализацию объекта.

Эквивалентная последовательность вызовов

```
jso = json_object_new_double(d);
json_object_set_serializer(d, json_object_userdata_to_json_string,
       strdup(ds), json_object_free_userdata)
```

Параметры

- d Числовое значение типа double.
- ds Строковое представление этого числа. Оно и будет скопировано.

Возвращает Объект `json_object` типа `json_type_double`

3.1.4 Булевское

Функции для создания объектов типа boolean

```
struct json_object* json_object_new_boolean(json_bool b)
```

Создаёт новый пустой объект `json_object` типа `json_type_boolean`.

Параметр b Значение типа `json_bool` TRUE или FALSE (0 или 1)

Возврвщает a Объект `json_object` типа `json_type_boolean`

3.2 Создание массивов

Функция для создания массива

```
struct json_object* json_object_new_array(void)
```

Создаёт новый объект `empty json_object` типа `json_type_array`.

Параметры Нет

Возвращает Объект `json_object` типа `json_type_array`

Функция для добавления элемента в массив

```
int json_object_array_add(struct json_object *obj,
                          struct json_object *val);
```

Добавляет элемент в конец объекта json_object типа json_type_array. Счётчик ссылок при этом **не** увеличивается. Это позволяет сделать операцию добавления поля в объект более компактной. Если Вы желаете сохранить ссылку на добавляемый объект, Вы должны окружить передаваемый объект `json_object_get`.

Параметр

- obj экземпляр `json_object`
- val добавляемый объект `json_object`

Возвращает

- Ноль при успешном добавлении элемента
- -1 При ошибке

Функция для вставки элемента в заданную позицию

```
int json_object_array_put_idx(struct json_object *obj,
                              int idx,
                              struct json_object *val);
```

Вставляет или заменяет по указанному индексу эемент массива (объект json_object типа json_type_arra

Счётчик ссылок при этом **не** увеличивается. Это позволяет сделать операцию добавления поля в объект более компактной. Если Вы желаете сохранить ссылку на добавляемый объект, Вы должны окружить передаваемый объект `json_object_get`.

Если выполняется **замещение** объекта, счётчик ссылок замещённого объекта уменьшается.

Если индекс больше текущего размера массива, то размер массива будет автоматически расширен до значения индекса.

Параметры

- obj Экземпляр объекта `json_object`;
- idx Индекс, куда вставлять элемент;
- val Объект `json_object`, который должен быть добавлен.

Возвращает

- Ноль при успешном добавлении элемента
- -1 При ошибке

Функция сортировки мсассива

```
void json_object_array_sort(struct json_object *jso,
                            int(*sort_fn)(const void *, const void *)
                           );
```

Сортирует элементы объекта `jso` типа `json_type_array`.

В качестве обоих параметров функции `sort_fn` передаются указатели на `json_object`.

Параметры

- `obj` Экземпляр объекта `json_object`;
- `sort_fn` Функция сортировки.

Возвращает Ничего не возвращает.

3.3 Создание составных объектов

Функция для создания объекта, хранящего пары ключ:значение

```
struct json_object* json_object_new_object(void);
```

Создаёт новый пустой объект, со счётчиком ссылок равным 1. Тот, кто создал этот объект, будет его единственным владельцем. Надо помнить о том. что при вызове `json_object_object_add` или `json_object_array_put_idx` владельцем станент соответствующий массив/объект. Если вы желаете сохранить совместное владение или даже добавить этот объект как дочерний многих объектов или массивов, Вам нужно вызвать `json_object_get`. Любые права владения, которые Вы получили, но не передали, должны быть освобождены вызовом `json_object_put`.

Параметры Нет

Возвращает `json_object` объект типа `json_type_object`

Функция для добавления пары ключ:значение в составной объект

```
extern void json_object_object_add(struct json_object* obj, const char *key,
                                   struct json_object *val);
```

Добавляет поле объекта в `json_object` типа `json_type_object`.

Счётчик ссылок **не** будет инкрементирован. Это делает добавление поля в объект более компактным. Если Вы желаете сохранить ссылку на добавляемый объект, независимо от времени жизни `obj`, вы дожны окружить передаваемый объект с помощью вызовов `json_object_get`.

После вызова этой функции, права владения `val` передаются к `obj`. таким образом, Вы должны проверить. что Вы реально являетесь владельцем этого объекта. Например, `json_object_new_object` сделает Вас владельцем до тех пор, пока Вы не отдатите право собственности, вызвав `json_object_object_get`.

Параметры

- obj Экземпляр объекта `json_object`;
- key Имя поля объекта (будет сделана приватная копия)
- val Объект `json_object` или `NULL` связанный с этим полем.

3.4 Запись данных в файл

Объект JSON любого типа имеет метод *json_object_userdata_to_json_string*, превращающий его значение в строку символов. Этот универсальный метод вызывает, в зависмости от типа элемента, соответствующий конкретный преобразователь:

- json_object_object_to_json_string;
- json_object_boolean_to_json_string;
- json_object_int_to_json_string;
- json_object_double_to_json_string;
- json_object_string_to_json_string;
- json_object_array_to_json_string;

Для того, что бы сохранить строковое представление JSON объекта, необходимо просто записать этот текстовый буфер в файл:

Запомнить объект в файле fd

```
fprintf(fd, ”%s\n”, json_object_to_json_string(my_object));
```

4 Восстановление данных из файла

Для того, что бы восстановить данные из текстового файла (или принятого сообщения) необходимо провести синтаксический разбор текста, что бы преобразовать его в объект JSON, а затем разобрать каждое поле этого объекта (возможно - рекурсивно).

Типично, это выглядит таким образом:

Типичный алгоритм разбора текстового буфера

```
struct json_object_iterator it;
struct json_object_iterator itEnd;
struct json_object* obj;

obj = json_tokener_parse(”{’first’:’george’, ’age’:100}”); ❶

it = json_object_iter_begin(obj); ❷
itEnd = json_object_iter_end(obj);

while (!json_object_iter_equal(&it, &itEnd)) { ❸
    printf(”%s\n”,
            json_object_iter_peek_name(&it)); ❹
    json_object_iter_next(&it); ❺
}
```

❶ Текстовый буфер превращаем в объект

❷ Определяем начальное и конечное значения итератора

❸ Цикл до граничного значения итератора

❹ Выбираем очередной элемент в дереве объекта

❺ Переходим к следующему элементу.

5 Обработка ошибок

wwwwwwwwwww