C++ HttpServer 和 RapidJSON调研

孙权 2018-08-15

# 调研 C++ HttpServer

## 背景

由于要和XONE Java的服务交互， C++ 后台服务考虑引入一个Restful 类型的HttpServer 第三方组件。

## 现状

目前C++ 开源的Http Server有以下几类：第一种是基于libuv/libevent+http parser 实现的异步http库；第二类是基于Boost.Asio实现的http库(比如Beast, pion)；第三类是自成体系，比较重量的http库（比如facebook 的proxygen）。

开源项目列表如下：

轻量级的：

* mongoose 异步, 支持Restful，超轻量(仅2个文件)的HTTP库

Single-threaded, asynchronous, non-blocking core with simple event-based API

<https://github.com/cesanta/mongoose>

* mongoose-cpp通过C++包装，更容易使用，并且支持JsonCPP的绑定（这点和Spring-boot的feign 很像）

<https://github.com/Gregwar/mongoose-cpp>

* w 基于Libevent 简单的web框架，写法看起看起来有点酷

<https://github.com/simonask/w>

* cpp-netlib 纯头文件，跨平台，符合标准的网络库。

<http://cpp-netlib.org>

* pion基于boost的轻量级HTTP库

<https://github.com/splunk/pion>

* libhttpserver 用于创建内嵌的Rest http服务

<https://github.com/etr/libhttpserver>

* rest\_rpc 腾讯开源团队做的modern C++11 rpc framework 基于 ASIO

<https://github.com/qicosmos/rest_rpc>

* uv\_http\_t 基于libuv的http server 实现 (还在实验阶段,功能也不完整)

重量级的：

* proxygen <https://github.com/facebook/proxygen>

### Github Star 对比

|  |  |
| --- | --- |
| [proxygen](https://github.com/facebook/proxygen) |  |
| [mongoose](https://github.com/cesanta/mongoose/) |  |
| [pion](https://github.com/splunk/pion) |  |
| [libhttpserver](https://github.com/etr/libhttpserver) |  |
| [cpp-netlib](https://github.com/glynos/cpp-netlib) |  |
| [w](https://github.com/simonask/w) |  |
| [rest\_rpc](https://github.com/qicosmos/rest_rpc) |  |

具体选型哪一个？

个人感觉mongoose 相对轻量，支持WebSocket , MQTT ，SSL/TLS 等

uv\_http\_t 是基于libuv 的，作者自己实现了uv\_link\_t，借鉴了node.js 集成uv 的思想，但是做的还不够完善，不适合上生产。

# RapidJSON使用

## RapidJSON简介

**特点：高效的 C++ JSON 解析／生成器，提供 SAX 及 DOM 风格 API**

RapidJSON 是一个 C++ 的 JSON 解析器及生成器。它的灵感来自 [RapidXml](http://rapidxml.sourceforge.net/)。

* RapidJSON 小而全。它同时支持 SAX 和 DOM 风格的 API。SAX 解析器只有约 500 行代码。
* RapidJSON 快。它的性能可与 strlen() 相比。可支持 SSE2/SSE4.2 加速。
* RapidJSON 独立。 不需要依赖 STL、BOOST 等。
* RapidJSON 对内存友好。在大部分 32/64 位机器上，每个 JSON 值只占 16 字节（除字符串外）。它预设使用一个快速的内存分配器，令分析器可以紧凑地分配内存。
* RapidJSON 对 Unicode 友好。它支持 UTF-8、UTF-16、UTF-32 (大端序/小端序)，并内部支持这些编码的检测、校验及转码。例如，RapidJSON 可以在分析一个 UTF-8 文件至 DOM 时，把当中的 JSON 字符串转码至 UTF-16。它也支持代理对（surrogate pair）及 "\u0000"（空字符）。
* RapidJSON 是一个纯头文件的库。
* RapidJSON 没使用 C++ 异常、RTTI。

## 在Centos7.5上安装

$ yum install rapidjson

当前默认安装的版本：**v1.1.0** (也是官方Release的最新版本20160825)。

## 官方教程

官方教程：<http://rapidjson.org/zh-cn/md_doc_tutorial_8zh-cn.html>

#include "rapidjson/document.h"

#include "rapidjson/writer.h"

#include "rapidjson/stringbuffer.h"

#include <iostream>

using namespace rapidjson;

int main() {

// 1. Parse a JSON string into DOM.

const char\* json = "{\"project\":\"rapidjson\",\"stars\":10}";

Document d;

d.Parse(json);

// 2. Modify it by DOM.

Value& s = d["stars"];

s.SetInt(s.GetInt() + 1);

// 3. Stringify the DOM

StringBuffer buffer;

Writer<StringBuffer> writer(buffer);

d.Accept(writer);

// Output {"project":"rapidjson","stars":11}

std::cout << buffer.GetString() << std::endl;

return 0;

}

## 性能测试

来源于作者miloyip 性能测试<http://miloyip.com/2016/nativejson>

### RapidJSON和 JsonCpp 对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 对比项 | RapidJSON | JsonCpp | 备注 |
| 0 | OverAll | 93% | 85% | 综合标准符合程度 |
| 1 | Parse (ms) | 8 | 166 | 解析时间 |
| 2 | Parse (byte) | 4,833,344 | 24,560,400 | 解析后的内存分配大小 |
| 3 | Stringify (ms) | 11 | 94 | Stringify 时间 |
| 4 | Prettify (ms) | 17 | *未统计* | Prettify 时间 |
| 5 | Code\_size(byte) | 31,248 | 243,478 | 代码大小(or文件字节) |

### 评测方法

标准符合程度（conformance）：

1. 使用 [JSON\_checker](http://www.json.org/JSON_checker/) 判断是否能分辨正确及错误的 JSON（parse validation）。
2. 解析字符串类型 JSON，如正确结果比对（parse string）。
3. 解析数字类型 JSON，如正确结果比对（parse double）。
4. 生成 DOM，再用 DOM 生成 JSON。判断来回往返（roundtrip）的字符串是否相同。

有些库可能没有生成 JSON 的功能，那么就不能完成来回往返检测。

性能检测：

1. 解析 JSON 至 DOM（parse）。
2. 把 DOM 转换至 JSON（stringify）。
3. 美化转换（prettify）。
4. 遍历 DOM 统计各种 JSON 类型的数量及字符串长度（statistics）。
5. 使用 SAX 做来回往返（sax roundtrip）。
6. 使用 SAX 做统计 (sax statistics)。
7. 生成一个程序，检测可执行档的大小。该程序能解析 JSON 至 DOM 并把统计结果打印出来。

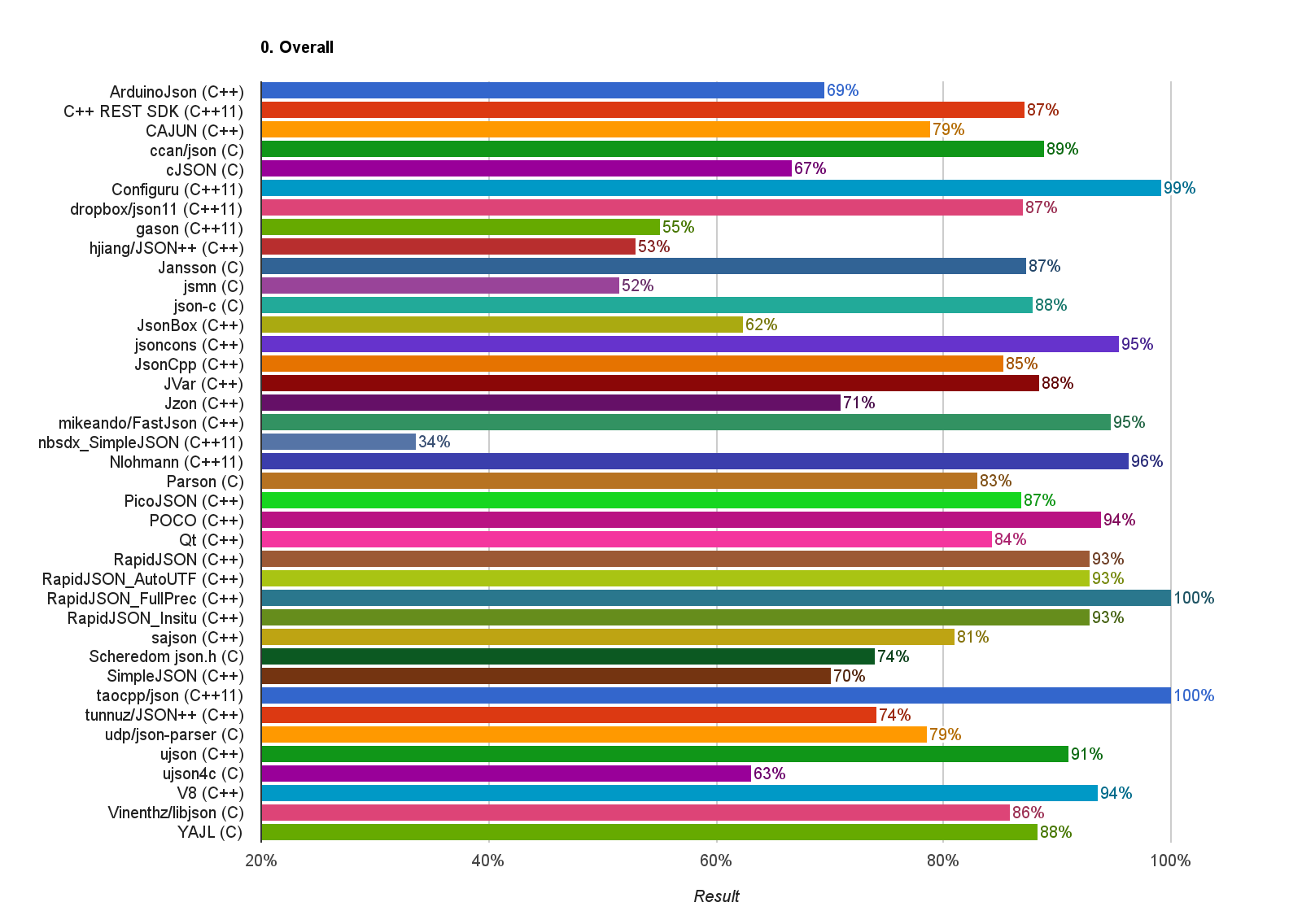
现时有 [3 个性能测试用的 JSON](https://github.com/miloyip/nativejson-benchmark/tree/master/data)，合共大小是 4.6MB。

加入此评测的库，最低限度要能实现 parse 和 statistics。其他检测是可选的。

### 结果

以下是 2016 年 9 月 9 日，在 MacBook Pro (Retina, 15-inch, Mid 2015, Corei7-4980HQ 2.80GHz)、clang 7.0 64-bit 的执行结果。也可用[互动版本观看](https://rawgit.com/miloyip/nativejson-benchmark/master/sample/conformance.html)。

综合标准符合程度（越高越好）：



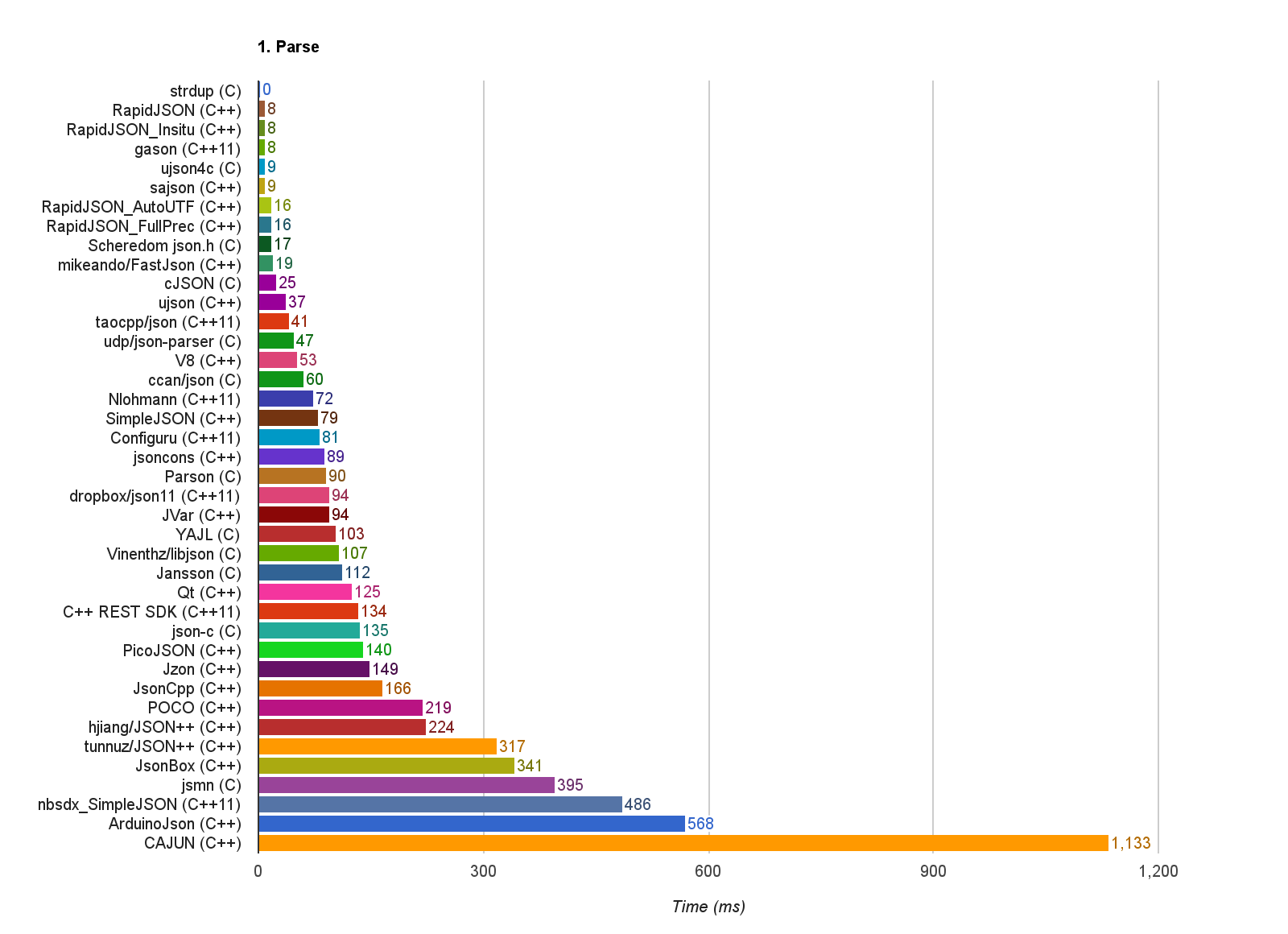
目前只有 RapidJSON 的全精度版本及 [The Art of C++ / JSON](https://github.com/taocpp/json) (taocpp/json) 能获得 100% 完美得分。

其实这个检测中，部分数字解析及相关的来回往返测试是过于严格的，超越了标准所需的程度。有大约 80%-90% 已经可以说完全符合标准。之后会调整检测，把一些不属于标准必须的部分仅作参考，不算进整体分数。

有些库未能处理 ‘“\u0000”’ 和代理对（surrogate pair），尚可接受。有些库连 "\n" 这类转义符都不处理，就有点太过分了。

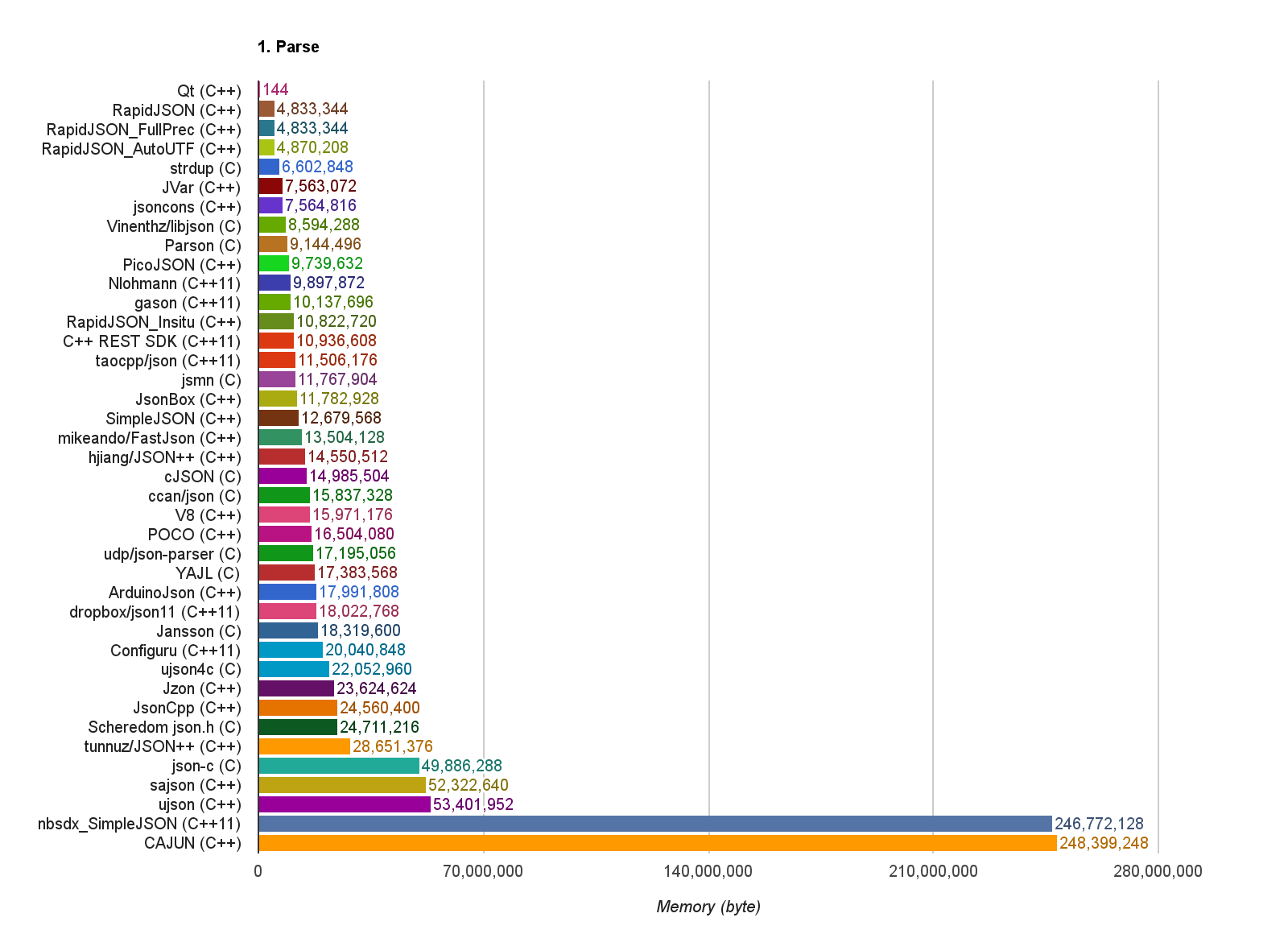
另一个常见问题是，一个 JSON 值结束后只能含有空白字符，例如[1, 2, 3] excited就是不合法的。有些库没做这个检测。

解析时间（越低越好）：



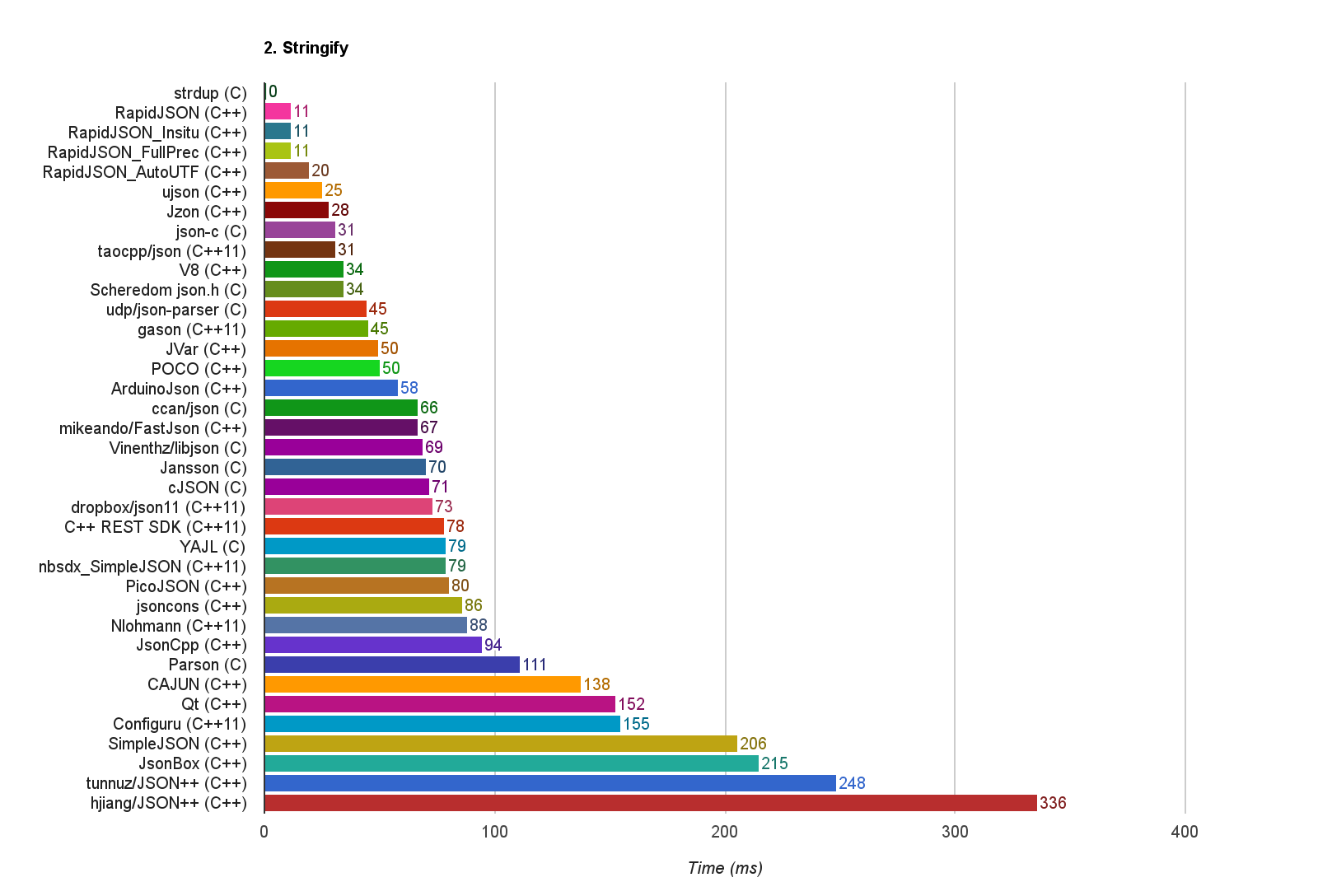
前排是 RapidJSON、[gason](https://github.com/vivkin/gason)、[ujson4c](https://github.com/esnme/ujson4c)、[sajson](https://github.com/chadaustin/sajson)。然而，后三者的标准符合程度都较低。现时最快的 RapidJSON 以 7.9ms 解析 4.6MB 的 JSON 文本，比最后一名快 140 倍以上。

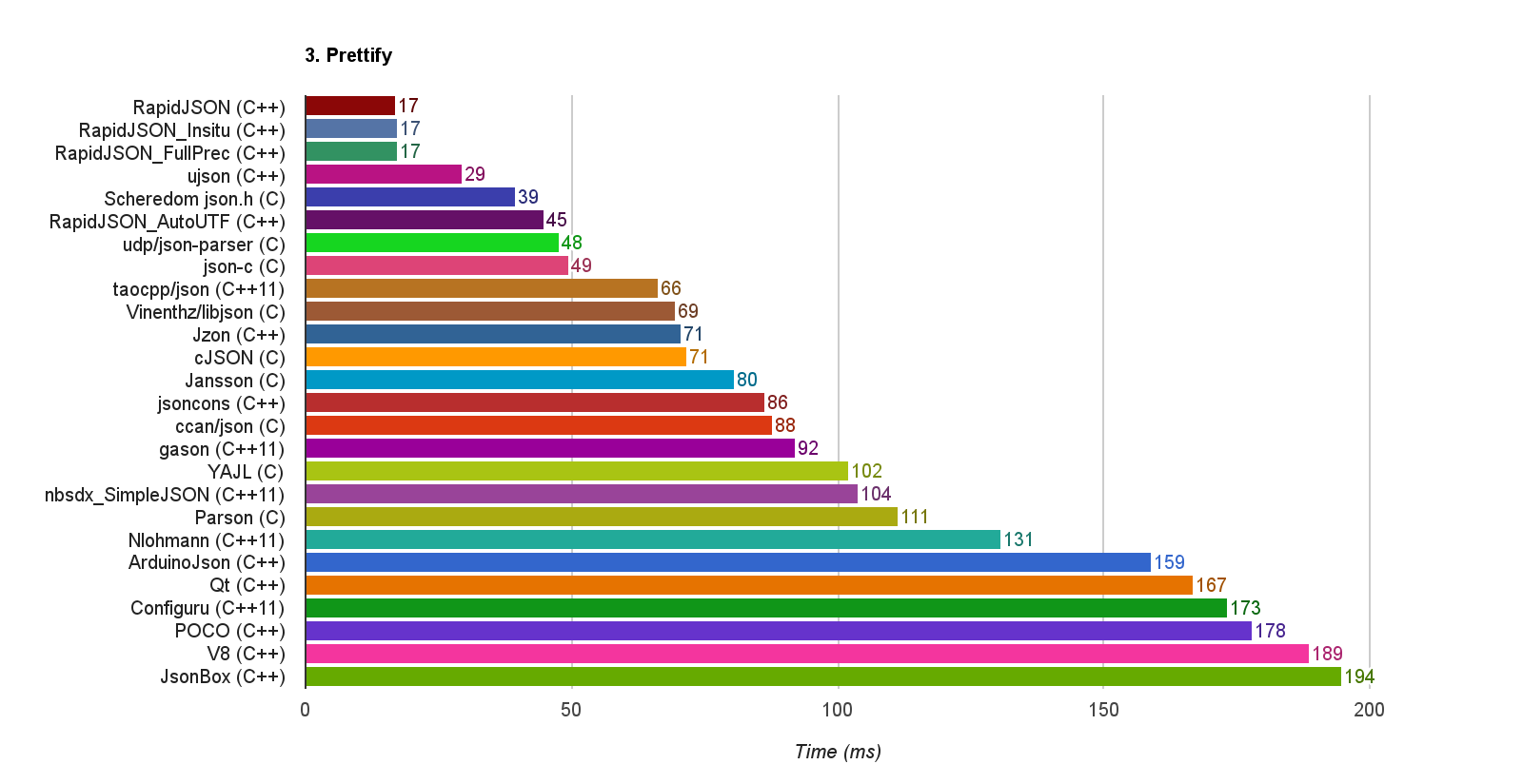
解析后的内存分配大小（越低越好）：



Qt 未能成功检测其内存消耗，请忽略。RapidJSON 暂无对手，它在 x64 架构下的内存总消耗是 (JSON 节点数量 ✕ 16 字节 + 长字符串字节大小)。长字符串是指超过 13 个字节的字符串。之前发现了一个比 RapidJSON 更省的 [jbson](https://github.com/chrismanning/jbson)，但因为在 travis 上崩溃暂未加进评测。

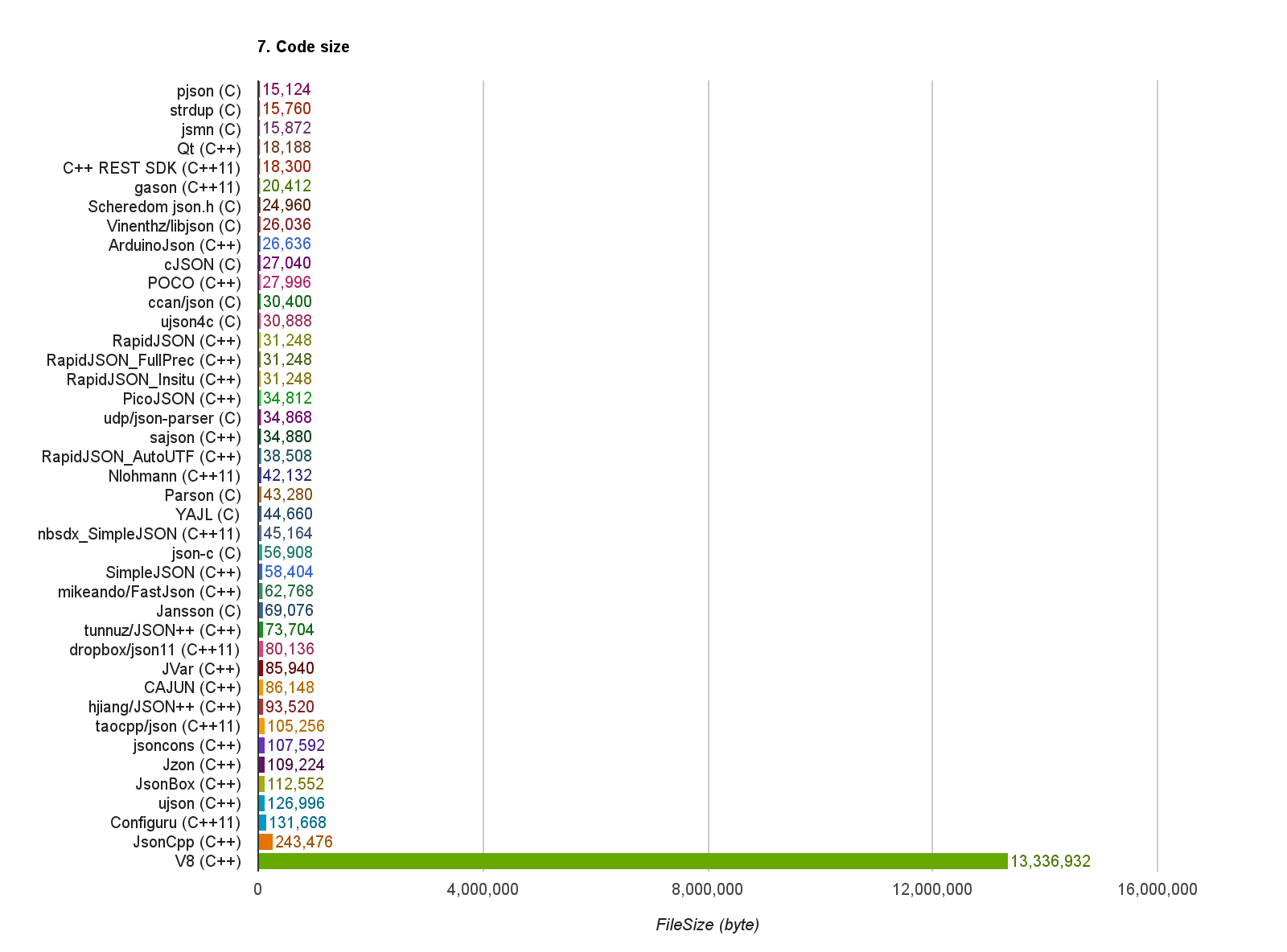
Stringify 和 Prettify 时间（越低越好）：





这两项都是 RapidJSON 完胜。这里有一个问题是测试的 JSON 数据中，最大的一个主要内容全为浮点数，而这类型的生成也是各类型中最耗时的，也许将来要再调整测试数据。

代码大小（越低越好）：



# 参考文档

1 RapidJSON <https://github.com/Tencent/rapidjson>

2 RapidJSON 性能测试 <https://github.com/miloyip/nativejson-benchmark>

3知乎jsoncpp和rapidjson哪个好用？ <https://www.zhihu.com/question/23654513>

4 RapidJSON 作者blog <http://miloyip.com/2016/nativejson>