**金融行业开源软件研究评测报告——JSON组件**

**背景**

JSON是一种轻量级的数据交换格式，采用完全独立于编程语言的文本格式来存储和表示数据，简洁和清晰的层次结构使得JSON成为理想的数据交换语言。在业务系统中，JSON组件作为数据转换和处理的重要环节，对整个系统的数据传输效率起决定作用。

结合金融企业对JSON组件的应用需求，我们对C/C++语言的三个主流解析工具（cjson、libjson、rapidjson）进行了评测，旨在为开发者选取JSON组件时提供参考和指导。

**JSON组件评测模型**

JSON组件评测模型基于金融行业开源软件成熟度评测整体模型进行建立。整体模型充分结合了开源软件的特性、系统工程领域对于软件产品质量的要求以及金融行业对于开源软件的使用需求。**整体模型涵盖开源许可证、行业认可度、产品活力、服务支持、安全性、兼容性、可维护性、可扩展性、功能性、可靠性、易用性、性能效率等12 大类评估属性，117 个评测指标。**

JSON组件代码量较小、功能简单、相对比较独立。开发人员一般在开发过程中直接引入，调用相应接口完成相关功能。因此，JSON组件在行业使用情况、服务支持等方面的信息非常少，对企业来说无需评估。同时，JSON作为独立的小型组件，在兼容性、可维护性、可扩展性、可靠性和易用性也不是企业关注的内容。

**JSON评测模型重点围绕企业关注的特性以及可评测的指标进行建立，涵盖整体模型的5个方面，包括开源许可证、产品活力、安全性、功能性、性能效率。**JSON评测模型具体详见已发布的**金融业开源软件研究评测（二）——JSON组件评测模型。**

**评测环境**

|  |  |  |
| --- | --- | --- |
| **CPU/内存** | **主频** | **软件环境（操作系统、应用软件）** |
| 4核/8G | 2.0ghz | Ubuntu 14.04 |

每个JSON组件选取稳定版本，在一台Ubuntu虚拟机分别进行评测，JSON组件版本信息见下表。

|  |  |  |
| --- | --- | --- |
| **软件名称** | **版本号** | **源码地址** |
| cjson | 1.5.3 | <https://github.com/DaveGamble/cJSON> |
| libjson | 0.8 | <https://github.com/vincenthz/libjson> |
| rapidjson | 1.1.0 | <https://github.com/miloyip/rapidjson> |

**开源许可证对比**

开源许可证是对开源软件的授权许可，对企业在应用开源软件过程中提出了关于知识产权、版权等方面的要求。一般来说，企业拿来作为内部使用，一般不会产生影响。但是，如果企业要进行软件商业化，那么不同的开源许可证对企业的要求将千差万别，尤其是会产生潜在的法律纠纷。

评测的JSON组件相关许可证信息如下表。

|  |  |
| --- | --- |
| **软件名称** | **许可证类别** |
| cjson | MIT |
| libjson | LGPLv2.1 或LGPLv3.0 |
| rapidjson | MIT |

cjson和rapidjson遵循MIT开源许可证协议，企业在发布商业版本时可以闭源。libjson遵循LGPLv2.1或LGPLv3.0开源许可证协议，企业在修改源码后发布商业版本时不可以闭源，必须严格遵循LGPL协议进行开源。

cjson、libjson、rapidjson开源许可证具体使用要求如下表。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **软件名称** | **修改源码后发布商业版本时是否允许闭源** | **再发布是否需要遵守原开源许可证** | **与企业内部其他开源产品是否存在潜在冲突** | **是否可以基于开源产品提供服务** | **是否允许申请专利** | **新增代码的产品版权是否属于企业** |
| cjson | 允许 | 不需要 | 不存在 | 可以 | 允许 | 衍生代码需要企业明确标出，新增代码的产品版权方可属于企业 |
| rapidjson | 允许 | 不需要 | 不存在 | 可以 | 允许 |
| libjson | 不允许 | 需要 | 存在 | 可以 | 允许 |

从上表可以看出，**libjson与cjson、rapidjson相比，存在修改源码后发布商业版本时不允许闭源（通过类库引用发布商业版本时允许闭源）、再发布仍然需要遵守原开源许可证、与企业内部其他开源产品存在潜在冲突（开源许可证协议条款之间不兼容）三个方面的问题。**因此，企业如果考虑发布涵盖JSON工具的闭源商业版本，优先选择cjson和rapidjson。

**产品活力对比**

开源社区参与贡献者数量、代码提交更新情况以及版本发布情况能够反映项目的持续发展情况，同时使用者关注、使用等情况能够体现开源代码在业界的使用情况，从而能够反映产品的整体活力。

评测从**贡献者数量，代码提交次数， Github Stars、Watch、Fork数， Commits、Pull Requests数，代码生命周期**等几个方面进行产品活力对比，具体数据如下：

**（一）贡献者数量：较多的开发者投身于rapidjson**

从贡献者数量来看, rapidjson明显高于cjson和libjson,并且每个季度都有稳定数量的贡献者，libjson的贡献停留在2014年和2015年。（由于libjson2016年和2017年不再有贡献量，所以只显示cjson和rapidjson的对比）

**（二）代码提交次数：cjson和rapidjson持续更新，发展良好**

cjson 和rapidjson提交数量稳定，随着软件趋于稳定，代码提交次数呈现减少趋势。libjson提交次数较少，2016年几乎不再有提交。

**（三）Stars、Watch、Fork数量：rapidjson远高于cjson和libjson**

**rapidjson大约是libjson和cjson数量的30倍和5倍**，cjson的数量要高于libjson， 而对于Commits、Pull Requests数，rapidjson的数量远高于libjson和cjson。

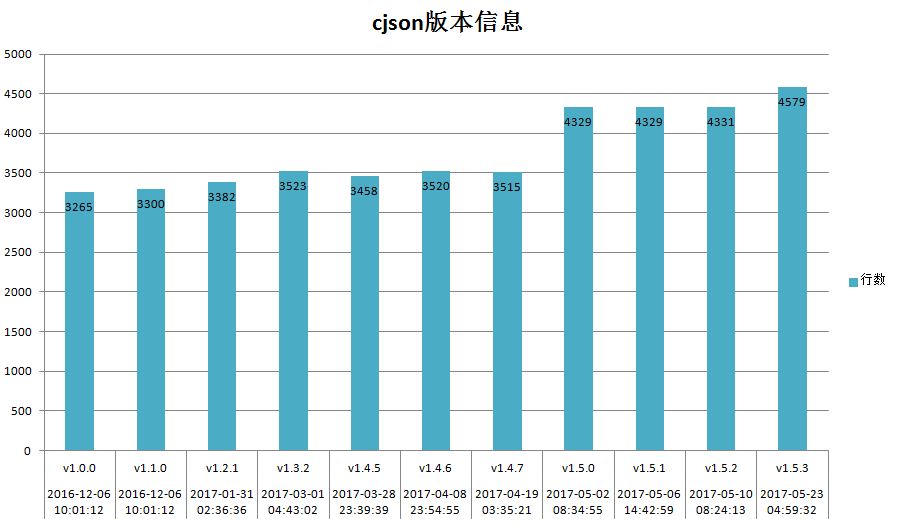
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **软件名称** | **Star** | **Folk** | **Watch** | **Commit** | **Pull requests** |
| cjson | 807 | 392 | 92 | 73 | 79 |
| libjson | 168 | 66 | 34 | 81 | 12 |
| rapidjson | 4639 | 1412 | 454 | 1939 | 409 |

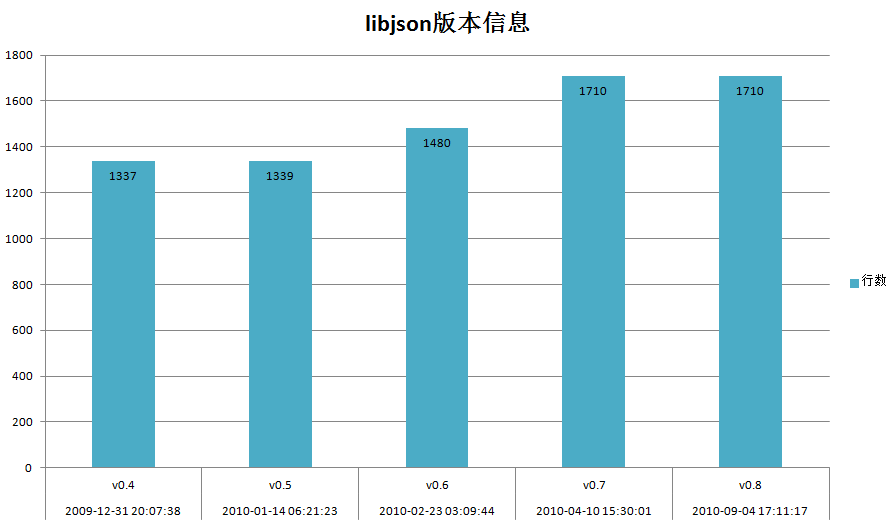
**（四）版本更新方面：rapidjson内容持续更新，cjon修改问题频繁**

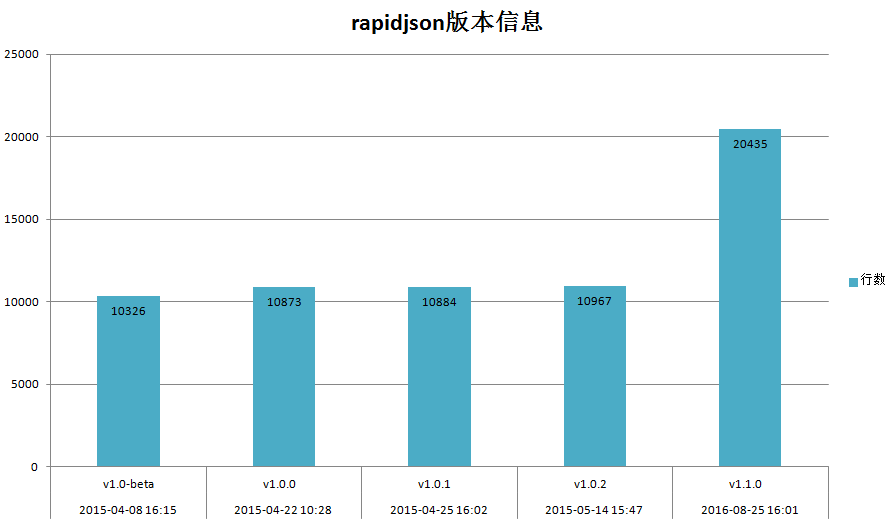
**libjson从2010年10月后，停止版本更新**。

cjson共发布11个版本，最新版本发布于2017年5月，通过对各个版本的更新日志研究发现，cjson发布的v1.0.0版本，已完成系统基本功能开发，后续**cjson版本的更新主要是修改问题、增加打印日志等，主体功能更新较少，代码量变化较小**。

rapidjson共发布5个版本**，**最新版本发布于2016年8月，通过对各个版本的更新日志研究发现，rapidjson发布的v1.0-beta版本，已完成系统基本功能开发，后续三个版本的更新主要是修改问题、接口组件化，**rapidjson v1.1.0版本更新主要是丰富功能和修改问题**，新增功能主要包括JSON指针、JSON模式、轻松JSON语法、降低内存消耗等。







**（五）论文、博客和专业论坛：rapidjson论文最多， rapidjson和cjson关注程度高**

在博客和专业论坛上，rapidjson和cjson的讨论要远高于libjson，说明**cjson和rapidjson的使用程度更高**，两者问题解决的参与程度更高。**rapidjson的论文数量最多**，内容集中在json的生成与解析，网络中数据压缩，数据管理和数据交换对json的使用。

**综合来看，rapidjson开发者和关注者的数量均远高于libjson和cjson，代码提交次数和版本迭代方面，rapidjson内容持续更新。因此，在产品活力方面，rapidjson活跃程度最高。**

**安全性对比**

随着信息系统和网络技术的不断发展，信息安全性成为人们关注的焦点。近年来发生的大量的网络攻击、信息泄露等事件，使得企业对软件的安全性更加重视。尤其是金融企业来说，更是如此。

**rapidjson和libjson尚未暴露安全漏洞。cjson在NVD软件库中已暴露6个安全漏洞，70%都为高危漏洞**，主要漏洞情况为远程攻击者利用json字符串中的字符导致服务器崩溃，越界内存访问，或者尾随某些字符串执行任意代码等。因此，**rapidjson和libjson相比cjson代码质量更高，相对来说更加安全。**

**功能性对比**

JSON组件的功能主要包括解析和解析生成。解析是将JSON格式数据转换为相应格式的数据。解析生成是将解析后再进行数据生成。

1. **解析： rapidjson和libjson擅长解析字符串，cjson擅长解析Double**

解析主要测试三种情况：解析有效和无效JSON成功率、解析Double成功率、解析字符串成功率，其中解析有效和无效JSON成功率有34个测试案例，解析Double成功率有66个测试案例，解析字符串成功率有9个测试案例，具体测试结果如下表。

|  |  |  |  |
| --- | --- | --- | --- |
| **软件名称** | **解析有效和无效JSON成功率** | **解析Double成功率** | **解析字符串成功率** |
| cjson | 91% | 97% | 89% |
| libjson | 100% | 88% | 100% |
| rapidjson | 100% | 73% | 100% |

**当解析Double类型的数据时，cjson的成功率最高，rapidjson的成功率最小。因此在解析Double类型数据的情况下建议使用cjson**。

**当解析字符串类型的数据时，cjson的成功率最低，rapidjson和libjson的成功率达到100%。因此在解析字符串类型的数据情况下建议使用rapidjson和libjson。**

1. **解析生成：rapidjson成功率最高**

解析生成主要测试一种情况：解析生成成功率。解析生成成功率有27个测试案例，具体测试结果如下表。

|  |  |
| --- | --- |
| **软件名称** | **解析生成成功率** |
| cjson | 67% |
| libjson | 63% |
| rapidjson | 100% |

测试结果表明，**rapidjson的解析生成成功率最高**。

**综合来看，rapidjson解析和解析生成的成功率是最高的，cjson解析Double的成功率最高。**

**性能效率对比**

性能效率主要考察解析效率、生成效率、统计效率、内存四个方面。解析效率是指对JSON数据进行解析所消耗的时间。生成效率是指将DOM数据转换为JSON数据所消耗的时间。统计效率是指对数据进行遍历所消耗的时间。内存是指对JSON数据进行解析所消耗的内存大小。

1. **解析效率：rapidjson效率最高**

解析效率主要测试一种情况：解析时间。解析时间测试使用的三个数据源大小为2199KB、1687KB、617KB，计算对三个数据源进行解析消耗的总时间，具体测试结果如下表。

|  |  |
| --- | --- |
| **软件名称** | **解析时间（ms）** |
| cjson | 60 |
| libjson | 78 |
| rapidjson | 10 |

测试结果表明，**解析json对象消耗的时间方面， rapidjson大约是cjson、libjson时间的1/6、1/8，rapidjson比cjson和libjson更加高效**。

1. **生成效率：rapidjson速度最快**

生成效率主要测试两种种情况：DOM生成JSON、DOM生成优化的JSON。生成效率测试使用的三个数据源大小为2199KB、1687KB、617KB，计算对三个数据源进行生成消耗的总时间，具体测试结果如下表。

|  |  |  |
| --- | --- | --- |
| **软件名称** | **DOM生成JSON(ms)** | **DOM生成优化的JSON(ms)** |
| cjson | 260 | 258 |
| libjson | 122 | 122 |
| rapidjson | 11 | 13 |

测试结果表明，**rapidjson的生成耗时大约是cjson、libjson时间的1/10、1/20**，**生成效率最高**。

1. **统计效率：rapidjson效率最高**

统计效率主要测试三种情况：DOM遍历统计、SAX统计、解析生成统计。统计效率使用的三个数据源大小为2199KB、1687KB、617KB，计算对三个数据源进行统计消耗的总时间，具体测试结果如下表。

|  |  |  |  |
| --- | --- | --- | --- |
| **软件名称** | **DOM遍历统计(ms)** | **SAX统计(ms)** | **解析生成统计(ms)** |
| cjson | 3 | / | / |
| libjson | 2 | 26 | 35 |
| rapidjson | 1 | 8 | 20 |

测试结果表明，**rapidjson的统计耗时大约是cjson、libjson的1/2、1/3，统计效率最高**。

1. **内存：rapidjson内存消耗最低**

内存主要测试三种情况：内存消耗、内存峰值消耗、内存分配次数。内存测试使用的三个数据源大小为2199KB、1687KB、617KB，计算对三个数据源进行解析所消耗的总内存，具体测试结果如下表。

|  |  |  |  |
| --- | --- | --- | --- |
| **软件名称** | **内存消耗（KB）** | **内存峰值消耗（KB）** | **内存分配次数** |
| cjson | 16,621 | 16,621 | 263,589 |
| libjson | 9,608 | 9,693 | 560,910 |
| rapidjson | 4,661 | 4,996 | 115 |

测试结果表明，**内存和内存峰值的消耗方面， rapidjson大约是cjson、libjson的1/3、1/2，内存分配次数方面， rapidjson大约是cjson、libjson的1/2000、1/500**0。rapidjson明显优于cjson和libjson。

**综合来看，rapidjson解析效率、生成效率、统计效率最高，rapidjson内存消耗最低。**

**总结**

根据JSON组件评测模型，我们从开源许可证、产品活力、安全性、功能性、性能效率五个方面，完成了对rapidjson、libjson、cjson三种开源JSON组件的成熟度评测，对比总结情况如下表所示。

|  |  |
| --- | --- |
| **测试指标** | **测试结果** |
| 开源许可证 | rapidjson=cjson>libjson |
| 产品活力 | rapidjson>cjson>libjson |
| 安全性 | rapidjson=libjson>cjson |
| 功能性 | rapidjson>libjson>cjson |
| 性能效率 | rapidjson>libjson>cjson |

**总体来说，rapidjson在各方面均优于cjson和libjson，同时libjson在安全性、功能性和性能效率方面优于cjson**。**建议企业在应用中优先选择rapidjson。**