**AnalysisDrive 1.1.1**

**Reference Guide**

**2015/07/06**

[**1.** **概要** 3](#_Toc423947153)

[**2.** **ライセンス** 3](#_Toc423947154)

[**3.** **ディレクトリ構成** 6](#_Toc423947155)

[**4.** **導入方法(Visual Studio 2013)** 7](#_Toc423947156)

[**5.** **機能紹介と使用方法** 10](#_Toc423947157)

[**5.1.** **コマンドライン引数のパース** 10](#_Toc423947158)

[**5.1.1.** **概要** 10](#_Toc423947159)

[**5.1.2.** **使用方法** 10](#_Toc423947160)

[**5.1.3.** **エラーメッセージ** 14](#_Toc423947161)

[**5.1.4.** **その他の機能** 14](#_Toc423947162)

[**5.2.** **JSONのパース** 15](#_Toc423947163)

[**5.2.1.** **概要** 15](#_Toc423947164)

[**5.2.2.** **使用方法** 16](#_Toc423947165)

[**5.2.3.** **エラーメッセージ** 22](#_Toc423947166)

[**5.2.4.** **その他の機能** 23](#_Toc423947167)

[**5.2.5.** **Jsonオブジェクト作成方法のまとめ** 24](#_Toc423947168)

[**5.3.** **MessagePackのパース** 25](#_Toc423947169)

[**5.3.1.** **概要** 25](#_Toc423947170)

[**5.3.2.** **使用方法** 25](#_Toc423947171)

[**5.3.3.** **エラーメッセージ** 26](#_Toc423947172)

[**6.** **リリースノート** 27](#_Toc423947173)

# **概要**

本ライブラリは, 数値解析プログラムを作成する上で必要になるパーサーなどの補助的な機能を集めたものである。

# **ライセンス**

本ライブラリは, 次の三条項BSDライセンスに従う。

Copyright (C) 2015 kyo   
All rights reserved.

Redistribution and use in source and binary forms, with or without   
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice,   
   this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice,   
   this list of conditions and the following disclaimer in the documentation   
   and/or other materials provided with the distribution.
3. Neither the name of the <organization> nor the names of its contributors   
   may be used to endorse or promote products derived from this software   
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND   
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,   
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF   
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE   
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR   
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,   
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT   
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;   
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)   
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN   
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR   
OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,   
EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

参考までに, 日本語訳を下に示す。

Copyright (C) 2015 kyo   
All rights reserved.

ソースコード形式であれバイナリ形式であれ, 変更の有無に関わらず,   
以下の条件を満たす限りにおいて, 再配布および利用を許可します.

1. ソースコード形式で再配布する場合, 上記著作権表示, 本条件一覧,   
   および下記免責事項を必ず含めてください.
2. バイナリ形式で再配布する場合, 上記著作権表示, 本条件一覧および下記免責事項を,   
   配布物とともに提供される文書 および/または 他の資料に必ず含めてください.
3. 派生したソースコードおよびバイナリを普及させるために, 著作権者の名前を勝手に   
   使用しないでください. 使用したい場合は事前に書面による許可を得てください.

本ソフトウェアは, 著作権者およびコントリビューターによって「現状のまま」提供されて   
おり, 明示黙示を問わず, 商業的な使用可能性, および特定の目的に対する適合性に   
関する暗黙の保証も含め, またそれに限定されない, いかなる保証もありません. 著作   
権者もコントリビューターも, 事由のいかんを問わず, 損害発生の原因いかんを問わ   
ず, かつ責任の根拠が契約であるか厳格責任であるか（過失その他の）不法行為で   
あるかを問わず, 仮にそのような損害が発生する可能性を知らされていたとしても, 本ソフ   
トウェアの使用によって発生した（代替品または代用サービスの調達, 使用の喪失,   
データの喪失, 利益の喪失, 業務の中断も含め, またそれに限定されない）直接損害,   
間接損害, 偶発的な損害, 特別損害, 懲罰的損害, または結果損害について, 一切責   
任を負わないものとします.

ただし, 本ライブラリはApache License, Version 2.0 のライセンスで配布されている成果物を含んでいる。該当する機能と対応する成果物の関係は次の通りである。

|  |  |
| --- | --- |
| 該当する機能 | Apache License, Version 2.0で配布されている成果物 |
| MsgpackParserクラス | Msgpack Copyright (C) 2008-2014 FURUHASHI Sadayuki |
| JsonクラスのDumpMsgpack関数 |

Apache License, Version 2.0については, <http://www.apache.org/licenses/LICENSE-2.0>を参照されたい。

# **ディレクトリ構成**

ディレクトリ構成を下に示す。

AnalysisDrive

doc … ドキュメント(indexというショートカットをクリックすると閲覧できる)

AnalysisDrive 本ライブラリと必要なヘッダーが入ったフォルダ

このフォルダがあれば本ライブラリを使用できる

src … 本ライブラリのソースコード(.cpp)

改変する場合のみ必要

inc … 本ライブラリのソースコード(.h)

改変する場合のみ必要

libs… 本ライブラリで使用している他のライブラリ

改変する場合のみ必要

Setting … Visual Studio 2013 の設定データ(.vssettings)が入ったフォルダ

特に必要ない

AnalysisDrive.sln … Visual Studio 2013 のソリューションファイル

改変する場合のみ必要

LICENSE.txt … ライセンスが記載されたテキストファイル

再配布時はこのファイルを使用すれば良い

# **導入方法(Visual Studio 2013)**

Visual Studio 2013での導入方法について説明する.

|  |
| --- |
| 1. プロジェクト作成後, メニューから [プロジェクト] ⇒ [\*\*\*のプロジェクト] でプロパティページを開く |
|  |

|  |
| --- |
| 1. 左上の構成を[すべての構成]にし, [C/C++] ⇒ [全般] と進んで追加のインクルードディレクトリに AnalysisDrive/AnalysisDriveのパスを追加する(このフォルダだけ必要). |
| C:\Users\kyo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\無題.png |

|  |
| --- |
| 1. [リンカー] ⇒ [全般] で追加のライブラリディレクトリに AnalysisDrive/AnalysisDriveのパスを追加する. |
|  |

|  |
| --- |
| 1. 左上の構成を[Debug]にし, [C/C++] ⇒ [コード生成] と進んでランタイムライブラリを/MTdにする. |
|  |

|  |
| --- |
| 1. 左上の構成を[Release]にし, [C/C++] ⇒ [コード生成] と進んでランタイムライブラリを/MTにする. |
| C:\Users\kyo\AppData\Local\Microsoft\Windows\INetCache\Content.Word\無題.png |

|  |
| --- |
| 1. <AnalysisDrive.h>をインクルードして使用する. |
|  |

# **機能紹介と使用方法**

# **コマンドライン引数のパース**

# **概要**

本機能を使用すると, 次のような形式のコマンド内容をパースできる(接頭辞や引数の数を設定できる).

hoge.exe /wave wave1.txt wave2.txt /stopPost hoge1.txt hoge2.txt

実行ファイル

prefix(接頭辞)

引数を2つもつ

/wave の引数

prefixなしの引数

prefix(接頭辞)

引数を持たない

接頭辞の文字, 接頭辞の後ろの引数の数を定義することができる.

# **使用方法**

コマンドライン引数のパースをするには次の3つのクラスを使用する.

|  |  |  |  |
| --- | --- | --- | --- |
| class adlib::CommandLineSetting (コマンドラインの読込形式を定義するクラス) | | | |
| メンバ関数 | 内容 | 引数 | 返り値 |
| CommandLineSetting | コンストラクタ | [第1引数]  接頭辞(空文字は禁止)  [第2引数]  接頭辞の後ろの引数の数  [第3引数]  ヘルプ表示 | なし |
| (static) ShowHelp | ヘルプを表示する | [第1引数]  本クラスが任意数格納されたコンテナ(std::vector)  [第2引数]  接頭辞なしの引数に関するヘルプ表示  [第3引数]  出力先のストリーム(std::ostream\*) | なし |

|  |  |  |  |
| --- | --- | --- | --- |
| class adlib::CommandLineParser (コマンドラインの内容をパースするクラス) | | | |
| メンバ関数 | 内容 | 引数 | 返り値 |
| Parse | コマンドライン引数をパースする | [第1引数]  コマンドラインの読み込み形式定義  (adlib::CommandLineSetting)が  格納されたコンテナ(std::vector)  [第2引数]  main関数の第1引数(argc)  [第3引数]  main関数の第2引数(argv) | パース結果(adlib::Command)が格納されたコンテナ(std::map)   * mapのキーはコマンドラインの接頭辞 |
| GetCommandArguments | 前回Parseに渡したコマンドラインの内容を繋げて文字列にして返す | なし | 前回Parseに渡したコマンドラインの内容  (std::string) |
| ErrorMessage | エラーメッセージを取得する | なし‘ | エラーメッセージが格納されたコンテナ(std::vector) |

|  |  |  |  |
| --- | --- | --- | --- |
| class adlib::Command (パース結果) | | | |
| メンバ関数 | 内容 | 引数 | 返り値 |
| Prefix | コマンドラインに入力された  接頭辞を取得する | なし | 接頭辞(const std::string)  “”であれば接頭辞なしの引数 |
| Args | 引数が格納されたコンテナの参照を取得する | なし | 引数が格納されたコンテナの参照  (const std::vector<std::string>&) |

まず, CommandLineSettingクラスでコマンドラインの内容(接頭辞, 接頭辞の後ろの引数の数, ヘルプ表示)を定義する. 1つのCommandLineSettingクラスで1つの接頭辞を定義し, 接頭辞の数だけコンテナ(vector)に格納する. 空の接頭辞””は使用してはならない(接頭辞なしの引数は定義しなくても取得できる).

// コマンドラインの定義(接頭辞, 接頭辞の後ろの引数の数, ヘルプ表示)

std::vector<adlib::CommandLineSetting> settings =

{

adlib::CommandLineSetting(“/?”, 0, “ヘルプを表示するフラグ”),

adlib::CommandLineSetting(“/wave”, 2, “入力波のファイルパス”),

adlib::CommandLineSetting(“/stopAnalysis”, 0, “解析を行わないフラグ”),

adlib::CommandLineSetting(“/stopPost”, 0, “ポスト処理を行わないフラグ”)

};

次に, CommandLineParserクラスのParse関数でコマンドライン引数をパースする. その際, コマンドラインの定義のコンテナを第1引数にセットする. Parse関数はパース結果としてCommandクラスのコンテナ(map)を返す. 例を示す.

入力されたコマンドライン(実行ファイル名をhoge.exeとする)

)

hoge.exe /wave wave1.txt wave2.txt /stopPost hoge1.txt hoge2.txt

値を取得するコード

int main(int argc, const char\* argv[])

{

adlib::CommandLineParser parser;

// コマンドライン引数のパース(Commandクラスのmapが返される)

auto commands = parser.Parse(settings, argc, argv);

// “/wave”のチェック(mapクラスのcountにより入力されているか確認する)

if (commands.count(“/wave”) > 0)

{

// CommandクラスのNumberOfArguments関数で引数の数を確認してループ

for (int i = 0; i < commands[“/wave”].Args().size(); ++i)

{

// []演算子でインデックスを指定すれば引数を参照できる

std::cout << commands[“/wave”].Args()[i] <<std::endl;

}

}

// 接頭辞なしの引数は“”で参照できる

if (commands.count(“”) > 0)

{

for (int i = 0; i < commands[“”].Args().size(); ++i)

{

std::cout << commands[“”].Args()[i] <<std::endl;

}

}

wave1.txt

wave2.txt

hoge1.txt

hoge2.txt

出力結果

# **エラーメッセージ**

CommandLineParserクラスのParse関数がコマンドライン引数のパースに失敗した場合, 空のmapを返す. その際のエラー内容は, CommandLineクラスのErrorMessage関数により, エラーメッセージの入ったコンテナ(std::vector<std::string>型)を取得することで分かる. このコンテナには, 前回のParse関数の結果が格納される. エラーが発生していなければサイズが0となる(vectorのsize関数で確認できる). エラーメッセージの内容は次の通りである.

|  |  |
| --- | --- |
| エラーメッセージ | エラー内容 |
| (Program Error) Prefix "\*\*\*" is Defined Double. | 設定された接頭辞が重複している (ライブラリ使用者のエラー). |
| (Program Error) There is no Commandline Contents. | コマンドライン引数の内容がない (実行ファイル名もない. ライブラリ使用者のエラー). |
| (Program Error) There is Prefix "\*\*\*" in Commandline Setting. | 設定された接頭辞に空文字””がある (ライブラリ使用者のエラー). |
| Prefix "\*\*\*" is Input Double. | 入力された接頭辞が重複している. |
| Parameter of Prefix "\*\*\*" is Insufficient. | 接頭辞の後ろの引数が不足している. |
| Parameter "\*\*\*" is Invalid. | 定義されていない接頭辞が入力されている. |

# **その他の機能**

その他に次の機能がある.

* CommandLineSettingクラスのstaticメンバであるShowHelp関数を使用すれば, ヘルプを表示できる(関数の仕様は上述の通り).
* CommandLineParserクラスのGetCommandArguments関数を使用すれば, 前回Parseした際のコマンドの内容を文字列で取得できる(関数の仕様は上述の通り).

# **JSONのパース**

# **概要**

JSONは次のように, name/valueのペアの集まりで記述されるデータ言語である。

{

“method” : “SMAC”,

“time” : {“duration” : 200000, “stepsize” : 0.001},

“particles”:

[

{“coord” : [0.0, 0.0, 0.0]},

{“coord” : [1.2, 0.0, 0.0]},

{“coord” : [1.2, 2.0, 0.0]}

]

}

**name**

**value**

valueには次の種類がある。

|  |  |  |
| --- | --- | --- |
| value | 内容 | 例 |
| string | “”で囲まれた文字列(UNICODEの並び) | “SMAC” |
| number | 数値(整数や浮動小数点数) | 200000 |
| object | {}で囲まれたname/valueペアのセット | {“duration” : 200000, “stepsize” : 0.001} |
| array | []で囲まれたvalueの集まり(配列) | [1.2, 2.0, 0.0] |
| bool | true or false | true |
| null | - | null |

トップレベルの要素として, objectかarrayを1つ記述する(上の例ではobjectとしている).

本ライブラリを使用すれば, JSONを分析し, valueを取得することができる. ただし, 文字コードセットはUTF-8に限定する(BOMの有無は問わない). ASCIIの範囲内(ローマ字, 数字など)であれば他の文字コードセットでもUTF-8と同様に扱える可能性がある(その場合はBOM無しの形式にするか, BOMを除外してからパースする必要がある).

# **使用方法**

詳しい仕様はdocフォルダ内のDoxygenドキュメントや, 実際のソースコードを参考にされたい. ここでは基本的な使用方法について説明する.

次のJsonParserクラスを使用すれば, JSON形式の文字列をパースすることができる.

|  |  |  |  |
| --- | --- | --- | --- |
| class adlib::JsonParser (JSON形式の文字列をパースするクラス) | | | |
| メンバ関数 | 内容 | 引数 | 返り値 |
| Parse | JSON形式の文字列をパースする | パースする文字列 | Jsonオブジェクト |
| ErrorMessage | エラーメッセージを取得する | なし‘ | エラーメッセージが格納され  たコンテナ(std::vector) |
| WarningMessage | ワーニングメッセージを取得する | なし‘ | ワーニングメッセージが格納され  たコンテナ(std::vector) |
| IsNameRepeated | 前回のParseでobjectのnameが重複していたか確認 | なし | 前回のParseでnameが重複していればtrueを返す |

Parse関数にJSON形式の文字列を渡すと, Jsonというクラスを返す.

// JSONで書かれたファイルを読み込む

std::ifstream ifs;

ifs.open(“test.txt”);

// バッファにファイルの内容を格納

std::string buf(“”);

std::string tmp(“”);

while(std::getline(ifs, tmp)) buf += tmp;

// パーサーを用意

adlib::JsonParser parser;

// パースしてJsonオブジェクトを取得

adlib::Json json = parser.Parse(buf);

JsonParserはJSONの各valueを, Jsonというクラスに変換する. このJsonオブジェクトを通してvalueを取得できる. JSONのvalueの型によって, 取得できる値の型は決まっている. その対応は次の通りである(adlibは本ライブラリの名前空間).

|  |  |  |
| --- | --- | --- |
| value (JSON) | Jsonオブジェクトから取得できる値の型 | 取得に使用する関数 (Jsonのメンバ関数) |
| string | std::string | GetString() |
| number | int, double (選択可能) | intならGetNumberInt()  doubleならGetNumberDouble() |
| object | std::map<std::string, adlib::Json> | GetObject() |
| array | std::vector<adlib::Json> | GetArray() |
| true | bool | GetBool() |
| false | bool | GetBool() |
| null | なし (内部的にはstd::nullptr\_tで扱う) | IsNull() で true が返ったらnull |

尚, 前述のコードで取得しているJsonオブジェクト(変数名json)は, ファイルに記述されたトップレベルのobject(あるいはarray)である.

例を示す. 次のJSONをパースしたとき, その下に記載しているコードで値を取得できる.

読み込んだJSON

{

“method” : “SMAC”,

“time” : { “duration” : 200000, “stepsize” : 0.001 },

“particles”: [ {“coord” : [0.0, 0.0, 0.0]}, {“coord” : [1.2, 0.0, 0.0]}, {“coord” : [1.2, 2.0, 0.0]} ]

}

値を取得するコード

// “method”の値をstringとして取得

auto sMethod = json.GetObject()[“method”].GetString();

// “method”の値を表示

std::cout << sMethod << std::endl;

// “time”の中の”stepsize”の値をdoubleとして取得

auto dStepsize = json.GetObject()[“time”].GetObject()[“stepsize”].GetNumberDouble();

// “stepsize”の値を表示

std::cout << dStepsize << std::endl;

SMAC

0.001

出力結果

arrayはvectorとして取得できるため, 上例の”particles”の内容は次のように取得する.

値を取得するコード

// “particles”の値をvectorとして取得

auto vParticles = json.GetObject()[“particles”].GetArray();

// イテレータでvectorの成分にアクセス

for (auto p = vParticles.begin(); p != vParticles.end(); ++p)

{

// “coord”の成分数を取得

size\_t size = p->GetObject()[“coord”].GetArray().size();

// インデックスでvectorの成分にアクセス(上のfor文のようにイテレータでアクセスしてもよい)

for (size\_t i = 0; i < size; ++i)

{

auto x = p-> GetObject()[“coord”].GetArray()[i].GetNumberDouble();

std::cout << x << “ “ << std::endl;

}

std::cout << std::endl;

}

0 0 0

1.2 0 0

1.2 2 0

出力結果

ここまで簡単な例を示してきたが, 厳密には次のような仕様になっているため注意されたい.

* **JSONのvalueの型と取得に使用した関数が対応していない場合, 無意味な値が返される**例えばvalueがstringであるのにも関わらず, GetNumberIntでint値を取得しようとした場合, valueの文字に関わらず0が返る. 失敗時に返される値は次の通りである.

|  |  |
| --- | --- |
| Jsonのメンバ関数 | 失敗時に返される値 |
| GetString() | “” |
| GetNumberInt() | 0 |
| GetNumberDouble() | 0.0 |
| GetObject() | サイズ0のmap |
| GetArray() | サイズ0のvector |
| GetBool() | false |

そのため, valueが想定通りの型であるかを, あらかじめ調べておくことを推奨する. これには次の関数を使用する.

|  |  |
| --- | --- |
| 確認する JSONの型 | 調べるための関数 (Jsonのメンバ関数. 型があっていればtrueが返る) |
| string | IsString() |
| number | IsNumber()   * IsNumberInt(), IsNumberDouble()で 整数か浮動小数点数かを調べられる |
| object | IsObject() |
| array | IsArray() |
| true | IsBool() |
| false | IsBool() |
| null | IsNull() |

また, Jsonのメンバ関数GetValueTypeの返り値によって型を調べることもできる.

|  |  |
| --- | --- |
| Jsonのメンバ関数GetValueType()の返り値 | JSONの型 |
| Json::JSON\_VALUE\_TYPE::STRING | string |
| Json::JSON\_VALUE\_TYPE::NUMBER\_INT | number (整数) |
| Json::JSON\_VALUE\_TYPE::NUMBER\_DOUBLE | number (浮動小数点数) |
| Json::JSON\_VALUE\_TYPE::OBJECT | object |
| Json::JSON\_VALUE\_TYPE::ARRAY | array |
| Json::JSON\_VALUE\_TYPE::BOOL | bool (true または false) |
| Json::JSON\_VALUE\_TYPE::NUL | null |

* **GetObjectで返されるmapに存在しないnameでアクセスしたとき, nullのJsonオブジェクトがmapに追加される**

例を示す.

読み込んだJSON

{ “method” : “SMAC” }

値を取得するコード

null

出力結果

// “met”の値をstringとして取得

auto sMethod = json.GetObject()[“met”].GetString();

// “met”の値を表示

std::cout << sMethod << std::endl;

// nullかどうか調べる

if (json.GetObject()[“met”].IsNull()) std::cout << “null” << std::endl;

nullのJsonオブジェクトが追加されるため, 上の例のjson.GetObject()[“met”].GetString()のように存在しないnameからGetString()を呼び出してもエラーとはならない. しかし, 次のようにnameが存在するかどうかをあらかじめ調べてからアクセスすることを推奨する.

// “met”か存在するかどうか調べる

if (json.GetObject().count(“met”) > 0) std::cout << “exist” << std::endl;

上の例ではmapクラスのcountによりnameの数を調べているが, findでイテレータを取得し, nameの存在を調べても良い.

// “met”か存在するかどうか調べる

if (json.GetObject().find(“met”) != json.GetObject().end()) std::cout << “exist” << std::endl;

しかし, 1度でも存在しないnameでアクセスするとnullのJsonオブジェクトが追加されるため, その後では上記の方法で調べることができない.

nullのJsonオブジェクトが追加されることを禁止するには, json変数にconstを付ければ良い.

// パースしてJsonオブジェクトを取得

const adlib::Json& json = parser.Parse(buf);

// “method”の値をstringとして取得

auto sMethod = json.GetObject().at(“method”).GetString();

// “met”の値をstringとして取得(失敗するとmapから例外std::out\_of\_rangeが投げられる)

auto sMet = json.GetObject().at(“met”).GetString();

こうすれば, 少なくともjson変数には暗黙にnullのJsonオブジェクトが追加されることはない. ただし, mapの[]演算子が使用できないため, 上の例のようにat関数でアクセスする必要がある. この場合, 存在しないnameでアクセスすると例外std::out\_of\_rangeが投げられる.

# **エラーメッセージ**

JsonParserクラスのParse関数がJSONのパースに失敗した場合, nullのJsonオブジェクトを返す. その際のエラー内容は, JsonParserクラスのErrorMessage関数により, エラーメッセージの入ったコンテナ(std::vector<std::string>型)を取得することで分かる. このコンテナには, 前回のParse関数の結果が格納される. エラーが発生していなければサイズが0となる(vectorのsize関数で確認できる). エラーメッセージの内容は次の通りである.

|  |  |
| --- | --- |
| エラーメッセージ | エラー内容 |
| Parse Failure. | JSONのパース失敗(詳細不明) |
| There is '0' Prefixial Number. | 0接頭のnumberがある |
| There is Bad Word in Number. | numberに無効な文字が含まれている |
| "\*\*\*" is out of Integer Range. | number(整数)がint型の範囲外 |
| "\*\*\*" is out of Double Range. | number(浮動小数点数)がdouble型の範囲外 |
| There is not '0'~'9' Follower of '.' in Number. | numberの小数点の後ろに数値がない |
| There is not '0'~'9' Follower of 'e' or 'E' in Number. | numberの指数表記’e’, ‘E’の後ろに数値がない |
| There is not '"' end of "\*\*\*". | 文字列を閉じる’”’がない |
| Escape Character of "\*\*\*" is Failure. | 不正なエスケープシーケンスがある |
| There is code in the range U+D800 ~ U+DFFF (surrogate). | エスケープシーケンスでサロゲート領域のUNICODEが指定されている(本ライブラリはUTF-8に従うため, サロゲート領域は禁止) |
| There is not '}' in object. | objectを閉じる’}’がない |
| There is not ',' in object. | object内の2つ目以降のname/valueペアの前に”,”がない |
| There is not '"' in object. | objectに文字列の開始を表す’”’がない |
| There is not ':' in object. | objectに”:”がない |
| There is not ']' in array. | arrayを閉じる’]’がない |
| There is not ',' in array. | array内の2つ目以降の値の前に’,’がない |
| Value "\*\*\*" is Failure. | 不正な値がある |

同様に, WarningMessage関数によりワーニングメッセージの入ったコンテナを取得できる. ワーニングは注意すべき事がある場合に発生し, エラーさえ起きていなければ正常にJsonオブジェクトを返す. ワーニングメッセージの内容は次の通りである.

|  |  |
| --- | --- |
| ワーニングメッセージ | ワーニング内容 |
| Name "\*\*\*" Repeated, and Later Name is Given Priority. | objectのnameが重複している |

「Key "\*\*\*" Repeated, and Later Key is Given Priority.」は, object内のnameに重複があった場合に発生するワーニングメッセージである. JSONの仕様(RFC4627)において, nameはuniqueであるべきとされている. しかし, 重複した場合も, 最後のname/valueペアを優先することにして, そのまま処理を続行するソフトウェアは少なくない. 本ライブラリでもその方針ではあるが, このワーニングによって重複を検知することができる. また, JsonParserクラスのIsNameRepeated関数は, 前回のParseでnameが重複していればtrueを返す.

# **その他の機能**

その他に次の機能がある.

* JsonクラスのDamp関数を使用すれば, 引数のストリームにJSON形式の文字列を出力できる.
* JsonクラスのDampMsgpack関数を使用すれば, 引数のストリームにMessagePack形式のバイナリを出力できる.

# **Jsonオブジェクト作成方法のまとめ**

JSONとJsonクラスとの関係を下図に示す。JsonParserクラスでJsonオブジェクトを作成すれば、Get\*\*\*メソッドを通してC++の基本的な型としてあなたのプログラムで利用できる。逆に、Jsonクラスのコンストラクタの引数にC++の基本的な型を入力すれば、Jsonオブジェクトを作成できる(下図参照)。

MessagePack（バイナリ）

JSON（テキスト）

class Json

**Analysis Drive**

JsonParser

クラス

Damp

メソッド

MsgpackParser

クラス

DampMsgpack

メソッド

あなたのプログラム

Jsonクラスのコンストラクタ

Get\*\*\*メソッド

* std::string
* int
* double
* std::map<std::string, adlib::Json>
* std::vector<adlib::Json>
* bool
* std::nullptr\_t
* adlibは本ライブラリの名前空間

左記の型をJsonクラスの

コンストラクタの引数にすることで

Jsonオブジェクトを生成できる

[C++の基本的な型]

# **MessagePackのパース**

# **概要**

MessagePackとはJSONライクなバイナリフォーマットである(<http://msgpack.org/>)。本ライブラリのMessageParserクラスを使用すると, MessagePack形式(Version 0.5.9)のバイナリをパースし, Jsonオブジェクトを作成できる(Jsonオブジェクトについては5.2節を参照).

# **使用方法**

次のMsgpackParserクラスを使用すれば, MessagePack形式のバイナリをパースすることができる.

|  |  |  |  |
| --- | --- | --- | --- |
| class adlib::MsgpackParser (MessagePack形式のバイナリをパースするクラス) | | | |
| メンバ関数 | 内容 | 引数 | 返り値 |
| Parse | MessagePack形式のバイナリをパースする | パースするバイナリ(文字列) | Jsonオブジェクト |
| ErrorMessage | エラーメッセージを取得する | なし‘ | エラーメッセージが格納され  たコンテナ(std::vector) |
| WarningMessage | ワーニングメッセージを取得する | なし‘ | ワーニングメッセージが格納され  たコンテナ(std::vector) |
| IsNameRepeated | 前回のParseでMapのnameが重複していたか確認 | なし | 前回のParseでnameが重複していればtrueを返す |

JSONをパースするJsonParserクラスと同様のインターフェイスであり, 使用方法も同様である(JsonParserクラスについては5.2節を参照). MessagePackと取得できるJSON(Jsonオブジェクト)の型の対応は次の通りである。

|  |  |
| --- | --- |
| MessagePackの型 | JSON(Jsonオブジェクト)の型 |
| Nil | null |
| Boolean | bool |
| Integer | number(整数) |
| Float | number(浮動小数点数) |
| String | string |
| Array | array |
| Map | object |
| Binary | 未対応(null) |
| Extension | 未対応(null) |

Binary型など未対応の型が検出された場合は, 代わりにnullが格納される.

# **エラーメッセージ**

MsgpackParserクラスのParse関数がMessagePackのパースに失敗した場合, nullのJsonオブジェクトを返す. その際のエラー内容は, MsgpackParserクラスのErrorMessage関数により, エラーメッセージの入ったコンテナ(std::vector<std::string>型)を取得することで分かる. このコンテナには, 前回のParse関数の結果が格納される. エラーが発生していなければサイズが0となる(vectorのsize関数で確認できる). エラーメッセージの内容は次の通りである.

|  |  |
| --- | --- |
| エラーメッセージ | エラー内容 |
| Unknown Error. | 予期せぬエラー(主にMsgpackライブラリ関連) |

同様に, WarningMessage関数によりワーニングメッセージの入ったコンテナを取得できる. ワーニングは注意すべき事がある場合に発生し, エラーさえ起きていなければ正常にJsonオブジェクトを返す. ワーニングメッセージの内容は次の通りである.

|  |  |
| --- | --- |
| ワーニングメッセージ | ワーニング内容 |
| Name "\*\*\*" Repeated, and Later Name is Given Priority. | Mapのnameが重複している |
| Binary Type is not Supported, and null is Given Value. | Binary型(未対応)が検出された |
| Extension Type is not Supported, and null is Given Value. | Extension型(未対応)が検出された |
| Unknown Type is Detected, and null is Given Value. | 未対応の型が検出された |

「Key "\*\*\*" Repeated, and Later Key is Given Priority.」は, Map内のnameに重複があった場合に発生するワーニングメッセージである. MsgpackParserクラスのIsNameRepeated関数は, 前回のParseでnameが重複していればtrueを返す.

# **リリースノート**

|  |  |  |
| --- | --- | --- |
| Version | 修正項目 | 追加機能 |
| 1.0.0 | なし | * コマンドラインパーサー * JSONパーサー |
| 1.1.0 | * JSONクラスにJSON形式でのダンプ機能を追加 * JSONクラスにMessagePack形式でのダンプ機能を追加 | * MessagePackパーサー |
| 1.1.1 | * デバッグ情報を生成しないように修正 (WARNINGが出てしまうため) | なし |