

言語の違いを意識することなく C言語通信ライブラリを利用可能とするラッパーの提案と実装

清水 一輝 †* 鈴木 秀和 † 内藤 克浩 ‡ 渡邊 晃 †

† 名城大学 ‡ 愛知工業大学

1 はじめに

大規模なプログラミングにおいて、ライブラリは処理速度や移植性、他言語との連携といった観点から、C言語で実装されることが多い。しかし、アプリケーションはC言語よりも抽象度の高い高級言語にて開発されることが一般的である。そのため、アプリケーションがC言語ライブラリを使用する場合は、一般的に呼び出し元の言語に応じたラッパーを作成し、ラッパーを経由してC言語ライブラリを使用する。

ラッパーを生成するツールとして SWIG (Simplified Wrapper and Interface Generator) [1] が存在する。しかし、SWIGにより生成されたラッパーは、C/C++ライブラリのAPIを意識して使用する必要がある。すなわち、C/C++ライブラリを使用したことのない開発者がC/C++ライブラリを意識する必要があるため、開発負荷が高くなる。そのため通信用APIといった、呼び出し元の標準APIにも同様な機能が用意されている場合は、呼び出し元の標準APIと同じ使用方法でライブラリを使用することが望ましい。

そこで本稿では、C言語ライブラリのAPIを意識する必要のあるこれまでの使用方法に加え、呼び出し元のプログラミング言語の標準APIと同じ使用方法でライブラリを利用可能とする、通信ライブラリを対象としたラッパーを提案する。提案方式を NTMfw (NTMobile framework library)[2] というC言語通信ライブラリに適用し、Java経由で動作確認及び処理時間の測定を行った。

2 既存技術：SWIG

SWIGはC/C++ライブラリのヘッダファイルを読み込み、呼び出し元の言語と連携を行うためのラッパーを生成する。連携にはFFI (Foreign Function Interface) と呼ばれる、C/C++にて実装された関数などを他のプログラミング言語から利用するための仕組みが用いられている。呼び出し元の言語は、自動生成されたラッ

パーを使用することにより、C/C++ライブラリを使用できる。しかし、ラッパーを介してC/C++ライブラリを使用する際には、C/C++ライブラリのAPIと同じ使用方法になる。呼び出し元の言語に新たな機能を提供する場合は、このラッパーを使用することに問題はない。しかし通信ライブラリのように、既に呼び出し元の言語にある機能と同様な機能を提供するライブラリの場合は適さない。

3 提案方式

3.1 目的と構成

本提案は、C言語通信ライブラリが提供するAPIの使用法に加え、呼び出し元のプログラミング言語の標準APIと同じ使用方法でも利用できるようにすることを目的とする。この目的を達成するために、C言語通信ライブラリとの連携と、言語間の違いの除去という2段階でラップを行う。図1に提案方式のモジュール構成を示す。連携用ラッパーはC言語通信ライブラリとの連携、通信用ラッパーは言語間の違いの除去の役割を担う。連携用ラッパーはSWIGにより生成されたラッパーを使用することができる。通信用ラッパーは、連携用ラッパーを再度ラップすることで言語間の違いを除去する。アプリケーションは、通信ライブラリが提供するAPIと同じ使用方法で使いたい場合は連携用ラッパー、呼び出し元の標準APIと同じ使用方法で使いたい場合は通信用ラッパーを選択する。

3.2 連携用ラッパーの構成

連携用ラッパーは、C言語ライブラリに対応したSWIGのコードを記述することで自動生成できる。連携用ラッパーでは、C言語通信ライブラリのAPIや型を呼び出し元の言語と連携できるようにマッピングする。アプリケーションは連携用ラッパーを使用することで、C言語通信ライブラリに実装されたAPIをC言語通信ライブラリと同じ使用方法で使用可能になる。

3.3 通信用ラッパーの構成

通信用ラッパーは、連携用ラッパーが提供するAPIを名前や型、引数が呼び出し元の標準通信用APIと一

Proposal and Implementation of Wrapper that makes
C-language Communication Library available
without Consciousness of Language Differences

†Kazuki Shimizu, †Hidekazu Suzuki,

‡Katsuhiro Naito, †Akira Watanabe

†Meijo University, ‡Aichi Institute of Technology

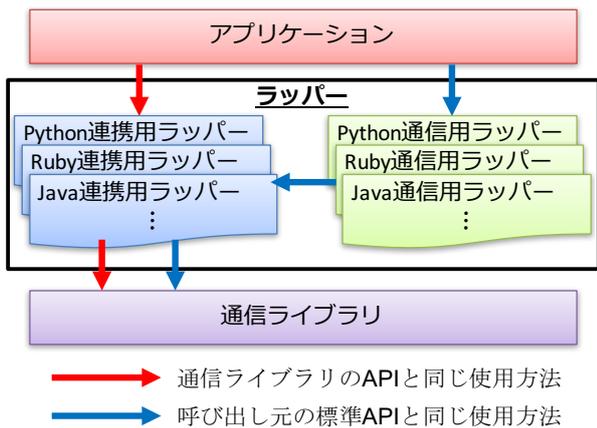


図 1: モジュール構成

致するようにラップする。ラップする際に、呼び出し元の標準通信用 API の引数で調整できないパラメータが存在する場合がある。この場合は、各言語の仕様に基づいた処理を行うように、パラメータをラッパー内で指定する。ラップした API は、API の内部で連携用ラッパーを経由することで通信ライブラリにアクセスする。アプリケーションは通信用ラッパーを使用することにより、呼び出し元の標準通信用 API と同じ使用方法で C 言語通信ライブラリを使用できる。

4 実装

提案方式の連携用ラッパーと通信用ラッパーを Java にて実装した。対象は、移動透過性を実現する C 言語通信ライブラリである NTMfw である。

4.1 連携用ラッパーの実装

SWIG により自動生成される連携用ラッパーではなく、代わりに JNA (Java Native Access) と呼ばれる C/C++ ライブラリと Java 間での連携を可能とするライブラリを使用し、NTMfw へのアクセスを可能にした。NTMfw の API を定義する NTMfw 用連携ラッパークラスを作成し、そこにマッピングした API を実装した。

4.2 通信用ラッパーの実装

Java 標準の通信用クラスを継承し、NTMfw による通信を行うように処理を実装した。NTMfw の API では細かくパラメータを調整できるようになっているが、Java 標準の通信用クラスの提供する API ではそのようなことを想定していない。例えばソケット生成用の API の場合、NTMfw の API ではソケット生成時に IPv4/v6 のどちらを使用するか指定できる。しかし、Java ではソケット生成時に IPv4/v6 を指定できず、IPv6 が使用

できる場合は IPv4 よりも IPv6 が優先されて使用される。そこで、このような API に対しては Java の仕様に基づいた処理を行うようにパラメータをラッパー内で指定した。

5 評価

5.1 動作確認

NTMfw の API を使用して通信している際に、ネットワークを切り替えることで移動透過性を実現できているか確認を行った。その結果、ネットワークを切り替えても通信を継続できたので、Java から C 言語通信ライブラリを使用できていることが確認できた。

5.2 処理時間の測定

連携用ラッパーと通信用ラッパーを使用する際の処理時間をそれぞれ測定した。測定対象は、UDP の送信用 API と受信用 API である。以下の表 1 に 100 回測定した結果の平均を示す。

表 1: UDP 送信/受信用 API の処理時間

使用ラッパー	送信 API[ms]	受信 API[ms]
連携用ラッパー	1.222	1.330
通信用ラッパー	1.277	1.363
差 (通信-連携用)	0.055	0.033

表 1 より、使用するラッパーの違いによって処理時間に与える影響は僅かであることが分かった。よって、使い方を呼び出し元の標準 API に合わせることは利便性といった観点から有用であると考えられる。

6 まとめ

本稿では、C 言語通信ライブラリを利用可能とする利便性の高いラッパーを提案した。提案方式により、プログラミング言語の違いを意識することなく、C 言語通信ライブラリを利用可能である。今後は、提案したラッパーを自動生成できるようにする予定である。

参考文献

- [1] SWIG developers : Simplified Wrapper and Interface Generator, <http://www.swig.org/>.
- [2] K.Naito, et al., "End-to-end IP mobility platform in application layer for iOS and Android OS", IEEE CCNC 2014, pp.276-281, 2014.