Introduction à la bibliothèque `cpr` pour la gestion des requêtes HTTPS en C++

31 mars 2025

Plan

Introduction aux commandes curl et http

La bibliothèque cpr en C++

Exemples d'utilisation

Introduction aux commandes `curl` et `http`

- `curl` est un outil en ligne de commande permettant de transférer des données via divers protocoles (HTTP, HTTPS, FTP, etc.).
- `http` (HTTPie) offre une interface plus conviviale pour tester et interagir avec des API HTTP.
- Exemple d'une requête GET avec `curl` :

```
curl -X GET https://www.example.com
```

Les méthodes HTTP : GET, POST, PUT, DELETE, PATCH

- `GET` : Récupère des informations depuis le serveur sans modifier d'état.
- `POST` : Envoie des données au serveur, généralement pour créer une nouvelle ressource.
- `PUT` : Met à jour une ressource existante ou en crée une nouvelle si elle n'existe pas.
- `DELETE` : Supprime une ressource du serveur.
- `PATCH` : Applique des modifications partielles à une ressource existante.

Ces méthodes constituent les bases de la communication HTTP/HTTPS pour interagir avec des API web.

Exemple de requête GET avec `curl`

Pour récupérer des données d'une API, utilisez une requête `GET` avec `curl`. Par exemple :

```
curl -X GET https://api.example.com/data -H "Accept:
    application/json"
```

Cette commande envoie une requête `GET` et affiche la réponse au format JSON.

Exemple de requête POST avec `curl`

Pour envoyer des données à une API, utilisez la méthode `POST` avec `curl` :

```
curl -X POST https://api.example.com/data \
     -H "Content-Type: application/json" \
     -d '{"nom": "exemple", "valeur": 123}'
```

Cette commande envoie un objet JSON pour créer ou mettre à jour une ressource.

Le format JSON

JSON (JavaScript Object Notation) est un format léger d'échange de données, facile à lire et à écrire pour les humains, et simple à analyser pour les machines.

- Utilise des paires clé-valeur pour représenter des objets.
- Les tableaux permettent de représenter des listes de données.
- Très utilisé dans les API web pour l'échange de données.

Exemple de JSON :

```
{
  "nom": "exemple",
  "age": 30,
  "interets": ["programmation", "lecture", "musique"]
}
```

Options avancées avec `curl`

- `-location` : Suivre automatiquement les redirections.
- `-verbose` : Afficher les détails de la communication.
- `-k` ou `-insecure` : Ignorer la vérification SSL (utiliser avec prudence).

Exemple avec redirection et verbosité :

```
curl --location --verbose https://api.example.com/data
```

Présentation de la bibliothèque `cpr`

- `cpr` est une bibliothèque C++ moderne simplifiant l'utilisation des requêtes HTTP/HTTPS.
- Inspirée de `curl`, elle permet d'interagir facilement avec des API web.
- Compatible avec les standards modernes (C++11 et plus, y compris C++17).

Installation locale via compilation des sources

- Installer les dépendances : `libcurl4-openssl-dev`, `cmake`, `build-essential` et `git`.
- Cloner le dépôt `cpr` depuis GitHub.
- Créer un répertoire de compilation et configurer avec CMake en précisant le préfixe d'installation (par exemple `$HOME/local`).
- Compiler et installer la bibliothèque.

```
sudo apt-get update
sudo apt-get install libcurl4-openssl-dev cmake build-
    essential git
git clone https://github.com/libcpr/cpr.git
cd cpr
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/local -
    DCMAKE_BUILD_TYPE=Release
make -j$(nproc)
make install
```

Intégration dans un projet CMake I

Pour intégrer cpr installé localement (par exemple dans `$HOME/local`) dans votre projet CMake, procédez comme suit :

- Ajoutez le chemin d'installation à la variable `CMAKE_PREFIX_PATH`.
- Utilisez la commande `find_package(cpr REQUIRED)` pour rechercher la bibliothèque.
- Liez cpr à votre cible avec `target_link_libraries`.

```
cmake_minimum_required(VERSION 3.10)
project(MonProjet)

# Ajout du chemin d'installation de cpr
set(CMAKE_PREFIX_PATH "$ENV{HOME}/local" ${
    CMAKE_PREFIX_PATH})

find_package(cpr REQUIRED)

add_executable(mon_executable main.cpp)
target_link_libraries(mon_executable PRIVATE cpr::cpr)
```

Exemple simple d'une requête GET

```
#include <cpr/cpr.h>
#include <iostream>

int main() {
    // Exemple de requête GET vers une URL sécurisée
    auto r = cpr::Get(cpr::Url{"https://www.example.com"});
    std::cout << r.text << std::endl;
    return 0;
}
```

Exemple d'une requête POST

```
#include <cpr/cpr.h>
#include <iostream>

int main() {
    // requête POST avec des paramètres
    auto r = cpr::Post(cpr::Url{"https://www.example.com"},
             cpr::Payload{{"param1", "valeur1"},
             {"param2", "valeur2"}});
    std::cout << r.text << std::endl;
    return 0;
}
```

Utilisation des fonctionnalités modernes (C++17)

```cpp
#include <cpr/cpr.h>
#include <string_view>
#include <iostream>

int main() {
    // Utilisation de std::string_view (C++17) pour définir
        l'URL
    std::string_view url = "https://www.example.com";
    auto r = cpr::Get(cpr::Url{std::string(url)});
    std::cout << r.text << std::endl;
    return 0;
}
```

Présentation de `llama-server`

- `llama-server` est un serveur d'inférence basé sur le modèle Llama.
- L'API permet d'interroger le modèle via HTTP, en envoyant un prompt et en récupérant une réponse générée.
- L'endpoint de l'API est, par exemple, `http://llmcode.math.unistra.fr:8090/completion`.

Interroger l'API avec `curl` : requête POST

```
curl -X POST http://llmcode.math.unistra.fr:8090/
    completion \
     -H "Content-Type: application/json" \
     -d '{"prompt": "In a few words: what is a template in
          c++ ?"}'
```

La commande envoie une requête POST avec un objet JSON contenant le prompt.

Analyse de la réponse avec curl

Vous pouvez utiliser jq pour formater la réponse JSON :

```
curl -X POST http://llmcode.math.unistra.fr:8090/
    completion \
     -H "Content-Type: application/json" \
     -d '{"prompt": "How to use templates in c++ ?"}' | jq
          .
```

Ceci affiche la réponse JSON de manière lisible.

Introduction au format JSON

JSON (JavaScript Object Notation) est un format léger d'échange de données qui est facile à lire pour les humains et simple à analyser pour les machines. Il est couramment utilisé pour l'échange de données entre serveurs et applications web.

- Basé sur des paires clé-valeur.
- Utilisé pour structurer des données, notamment dans les API Web.
- Facilement lisible et modifiable à la main, et peut être interprété par de nombreux langages.

Structures de données en JSON

JSON permet de structurer des données à l'aide de plusieurs types :

- **Objets** : Contiennent des paires clé-valeur, représentant des entités ou objets complexes.
- **Tableaux** : Listes ordonnées de valeurs, pouvant contenir des objets ou des valeurs primitives.
- **Types primitifs** : Chaînes de caractères, nombres, booléens et `null`.

Exemple de fichier JSON

Un fichier JSON est simplement un fichier texte contenant des données structurées en JSON. Voici un exemple de contenu d'un fichier JSON représentant une liste d'utilisateurs :

```
{
  "utilisateurs": [
      {
          "nom": "Alice",
          "age": 30,
          "email": "alice@example.com"
      },
      {
          "nom": "Bob",
          "age": 24,
          "email": "bob@example.com"
      }
  ]
}
```

Manipulations JSON en Python

Voici quelques exemples simples de manipulation de JSON en Python, avec la bibliothèque standard `json`.
Exemple de conversion d'un dictionnaire Python en JSON :

```
import json

data = {
    "nom": "Alice",
    "age": 25,
    "interets": ["musique", "lecture", "sport"]
}

json_data = json.dumps(data, indent=4)
print(json_data)
```

Exemple de lecture de données JSON depuis une chaîne :

```
json_str = '{"nom": "Alice", "age": 25, "interets": ["
    musique", "lecture"]}'
data = json.loads(json_str)
print(data["nom"], data["age"])
```

Interroger l'API avec cpr : requête POST

```
#include <cpr/cpr.h>
#include <iostream>

int main() {
    cpr::Response r = cpr::Post(
        cpr::Url{"http://llmcode.math.unistra.fr:8090/
            completion"},
        cpr::Body{"{\"prompt\": \"How to use templates in
            c++ ?\"}"},
        cpr::Header{{"Content-Type", "application/json"}}
    );
    std::cout << r.text << std::endl;
    return 0;
}
```

Gestion des erreurs avec cpr

```
#include <cpr/cpr.h>
#include <iostream>

int main() {
    auto r = cpr::Post(
        cpr::Url{"http://llmcode.math.unistra.fr:8090/
            completion"},
        cpr::Body{"{\"prompt\": \"How to use templates in
            c++ ?\"}"},
        cpr::Header{{"Content-Type", "application/json"}}
    );

    if (r.error) {
        std::cerr << "Erreur : " << r.error.message << std
            ::endl;
    } else {
        std::cout << "Réponse : " << r.text << std::endl;
    }
    return 0;
}
```

Cette gestion permet d'afficher un message d'erreur en cas d'échec de la requête.

Création et affichage d'un objet JSON

```
#include <nlohmann/json.hpp>
#include <iostream>
using json = nlohmann::json;

int main() {
    // Création d'un objet JSON avec quelques paires clé/
        valeur
    json j = {
        {"nom", "Alice"},
        {"age", 30},
        {"est_etudiant", false}
    };
    // Affichage du JSON avec une indentation de 4 espaces
    std::cout << j.dump(4) << std::endl;
    return 0;
}
```

Accès et modification de données JSON

```cpp
#include <nlohmann/json.hpp>
#include <iostream>
using json = nlohmann::json;

int main() {
    json j = { {"nom", "Bob"}, {"age", 25} };

    // Accès à une valeur
    std::cout << "Nom : " << j["nom"] << std::endl;

    // Modification d'une valeur
    j["age"] = 26;
    std::cout << "Nouvel âge : " << j["age"] << std::endl;

    // Ajout d'une nouvelle paire clé/valeur
    j["ville"] = "Paris";
    std::cout << j.dump(4) << std::endl;
    return 0;
}
```

Lecture d'un fichier JSON et parcours des éléments I

```
#include <nlohmann/json.hpp>
#include <iostream>
#include <fstream>
using json = nlohmann::json;

int main() {
    // Ouverture d'un fichier JSON nommé "data.json"
    std::ifstream file("data.json");
    if (!file) {
        std::cerr << "Erreur lors de l'ouverture du
            fichier." << std::endl;
        return 1;
    }

    // Lecture et parsing du fichier JSON
    json j;
    file >> j;

    // Parcours d'un tableau d'utilisateurs contenu dans
        le JSON
    for (const auto& utilisateur : j["utilisateurs"]) {
        std::cout << "Nom : " << utilisateur["nom"]
```

Lecture d'un fichier JSON et parcours des éléments II

```
                    << ", Âge : " << utilisateur["age"] <<
                        std::endl;
        }
        return 0;
    }
```