2024

PLOTS LUMINEUX

COMMUNIQUER EN ASYNCHRONE

PP LEFEBVRE Jordan - Lycée Raymond Queneau

Présentation :

La problématique de la communication réside dans le fait qu'elle ne s'effectue pas à la même vitesse qu'un échange de données interne à une application. En conséquence, il y a un délai (très court) entre l'envoi d'une requête et la réception de la réponse. Il est donc nécessaire de mettre en place une communication asynchrone pour gérer cette latence.

Classe d’interface de gestion des doonées :

On va modifier notreclasse data... en fonction de notre besoin.
Exemple : DataJoueurs
pour communiquer avec une API qui vous fournie une liste de joueur.
Il faut commencer par faire hériter la classe par QObject.

Envoyer une requête :

Pour transmettre une requête en HTTP, il faut utiliser la classe QNetworkAccessManager

Il faut donc Include cette classe et la déclarer.
Ici on la nomme manager.

Exemple d'utilisation de l'instance manager :

```
void DataJoueurs::getJoueur(const QString &id)
{
    connect(manager, &QNetworkAccessManager::finished,
            this, &DataJoueurs::lireDataGetJoueur);
    QNetworkRequest request(QUrl("http://192.168.190.4:38080/API/routeurJoueurs.php?id=" + id));
    manager->get(request); // récupérer le joueur spécifié par identifiant
}
```

La méthode get recoit 1 paramètre : La requête (URI de notre api et de notre ressource)

On est en fonctionnement asynchrone donc la fonction ne retourne pas la réponse de notre API.

Lire la réponse :

La classe QNetworkAccessManager envoie le signal finished lorsque la réponse de l'API est reçue.
Ce signal passe la réponse de l'API en paramètre.
Dans la méthode getJoueur(id), on a établit une connexion entre ce signal et une méthode spécifique de notre classe pour gérer la réponse. (Exemple : lireDataGetJoueur)

```
connect(manager, &QNetworkAccessManager::finished,
        this, &DataJoueurs::lireDataGetJoueur);
```

Dans notre cas, la fonction connect fait le lien entre la fin de la réception de la réponse http à notre requête et l'exécution de la méthode lireDataGetJoueur

Maintenant on créer cette méthode lireDataGetJoueur. Cette méthode sera appelée à chacune des réponses transmises par notre API.

```
void DataJoueurs::lireDataGetJoueur(QNetworkReply *reply)
{
    QString dataJson(reply->readAll());
    qDebug() << "Data Json Joueur : " << dataJson;
}
```

La méthode reçoit en paramètre la réponse de type QNetworkReply. Pour traiter la réponse, il faut utiliser la méthode readAll de l'instance reply.

Puis on affiche avec un qDebug afin de vérifier le fonctionnement de notre communication.

Décoder le json :

```
void DataResultats::lireDataCreateConfig(QNetworkReply *reply)
{
    disconnect(manager, &QNetworkAccessManager::finished, this, &DataResultats::lireDataCreateConfig);
    if (reply->error() == QNetworkReply::NoError) {
        QByteArray responseData = reply->readAll();
        qDebug() << "Réponse du serveur :" << responseData;


        QJsonParseError jsonError;
        QJsonDocument jsonResponse = QJsonDocument::fromJson(responseData, &jsonError);

        if (jsonResponse.isObject()) {
            QJsonObject jsonObject = jsonResponse.object();
            if (jsonObject.contains("statut")) {
                int statut = jsonObject["statut"].toInt();
                if (statut == 1) {

                    if (jsonObject.contains("configpartie") && jsonObject["configpartie"].isObject()) {
                        emit creationConfigReussie();
                        qDebug() << "Config créé avec succès.";
                    } else {
                        emit creationConfigEchouee();
                        qDebug() << "Erreur lors de la création de la config.";
                    }
                    return;
                }
            }
        }
    }
}
```

QJsonDocument est une chaine de caractère contenant du JSon.

Il faut maintenant décomposer les différents éléments se trouvant à l'intérieur.

En JSON, il existe :
- des objets : { 'nom' : 'toto' , 'prenom' : 'tata' }
- des tableaux : [{ 'nom' : 'toto' , 'prenom' : 'tata' },{ 'nom' : 'titi' , 'prenom' : 'tutu' }]

Pour décoder du JSON avec QT, il existe donc deux classes : - les objets : QJsonObject - les tableaux : QJsonArray

La méthode object() d'un élément json va retourner un tableau avec clé. La clé correspondante aux nom des objets.
La méthode toArray() permet de récupérer un tableau indexé. L'index étant l'ordre des objets dans le tableau.