Concept

JSON-RPC 2.0 est une manière simple de faire communiquer une application avec un serveur.

Une application envoie au serveur le nom d'une méthode à exécuter, avec des paramètres.
Le serveur exécute cette méthode et renvoie soit un résultat, soit une erreur.

En pratique, cela ressemble à un appel de fonction, mais à distance.

Exemple très simple

Imaginons une application qui veut récupérer un utilisateur.

Elle envoie ceci au serveur :

```
{
  "jsonrpc": "2.0",
  "method": "user.get",
  "params": {
    "id": 42
  },
  "id": 1
}
```

Cela veut dire :

“Serveur, appelle la méthode `user.get` avec l'identifiant `42`.”

Le serveur peut répondre :

```
{
  "jsonrpc": "2.0",
  "result": {
    "id": 42,
    "name": "Lara"
  },
  "id": 1
```

```
}
```

Cela veut dire :

“Voici le résultat de la méthode demandée.”

Les éléments importants

Une requête JSON-RPC contient généralement quatre éléments :

Élément	Rôle
`jsonrpc`	Indique la version du protocole. En JSON-RPC 2.0, la valeur e `"2.0"`.
`method`	Le nom de la méthode que le serveur doit exécuter.
`params`	Les paramètres envoyés à cette méthode.
`id`	Un identifiant qui permet de relier la réponse à la bonne der

---

Exemple avec une erreur

Si l’application demande une méthode qui n’existe pas :

```
{
  "jsonrpc": "2.0",
  "method": "user.deleteEverything",
  "params": {},
  "id": 2
}
```

Le serveur peut répondre :

```
{
  "jsonrpc": "2.0",
  "error": {
    "code": -32601,
    "message": "Method not found"
```

```
  },
  "id": 2
}
```

Ici, le serveur dit simplement :

“Je ne connais pas cette méthode.”

Différence avec une API REST

Une API REST est organisée autour de ressources.

Par exemple :

```
GET /users/42
POST /users
DELETE /users/42
```

REST fonctionne très bien quand on manipule des objets classiques comme :

- des utilisateurs ;
- des articles ;
- des commandes ;
- des produits ;
- des documents.

REST dit plutôt :

“Voici une ressource, je veux la lire, la créer, la modifier ou la supprimer.”

JSON-RPC, lui, est organisé autour d’actions ou de méthodes.

Par exemple :

```
{
  "jsonrpc": "2.0",
  "method": "user.get",
  "params": {
    "id": 42
  },
  "id": 1
```

```
}
```

JSON-RPC dit plutôt :

“Exécute cette méthode avec ces paramètres.”

Exemple concret : transfert d’un objet

Imaginons une application d’inventaire.

Avec JSON-RPC, on peut envoyer :

```
{
  "jsonrpc": "2.0",
  "method": "inventory.transferItem",
  "params": {
    "itemId": 123,
    "toLocationId": 5
  },
  "id": 10
}
```

C’est très clair :

“Transfère l’objet 123 vers le lieu 5.”

Le serveur peut répondre :

```
{
  "jsonrpc": "2.0",
  "result": {
    "transferId": 987,
    "itemId": 123,
    "newLocationId": 5,
    "status": "completed"
  },
  "id": 10
```

```
}
```

Pourquoi utiliser JSON-RPC ?

JSON-RPC est intéressant quand une application a beaucoup d’actions métier.

Par exemple :

```
auth.login
auth.logout
user.getCurrent
inventory.search
inventory.transferItem
document.generatePdf
signature.signDocument
notification.markAsRead
```

Dans ce genre de cas, JSON-RPC peut être plus naturel qu’une API REST.

Au lieu d’inventer beaucoup d’URLs, on nomme directement les actions que le serveur sait faire.

Avantages

JSON-RPC a plusieurs avantages :

- il est simple à comprendre ;
- il utilise du JSON, donc il est facile à lire ;
- il fonctionne bien avec un frontend web ;
- il permet d’avoir une seule adresse d’API, par exemple `/rpc` ;
- il est pratique pour les applications internes ou les outils métier ;
- les réponses ont toujours une structure prévisible.

Une réponse réussie ressemble toujours à ceci :

```
{
  "jsonrpc": "2.0",
  "result": {},
```

```
  "id": 1
}
```

Une réponse en erreur ressemble toujours à ceci :

```
{
  "jsonrpc": "2.0",
  "error": {
    "code": 123,
    "message": "Something went wrong"
  },
  "id": 1
}
```

Limites

JSON-RPC n’est pas parfait.

Il est moins connu que REST pour les APIs publiques.
Il profite moins naturellement du cache HTTP.
Il demande aussi une bonne discipline dans le nommage des méthodes.

Par exemple, ces noms sont clairs :

```
inventory.search
inventory.transferItem
document.generatePdf
user.updateProfile
```

Ces noms sont mauvais :

```
doStuff
process
getData
handle
update
```

Si les méthodes sont mal nommées, l’API devient vite difficile à comprendre.

Comparaison rapide

Technologie	Idée principale	Quand c’est utile
REST	Manipuler des ressources avec des URLs	APIs publiques, CRUD classique, ressou
JSON-RPC	Appeler des méthodes à distance	Applications métier, actions précises, b
GraphQL	Le client choisit exactement les données qu’il veut	Gros frontends avec besoins très variab
gRPC	Appels rapides et typés entre services	Communication backend-to-backend
WebSocket	Connexion ouverte en temps réel	Chat, notifications instantanées, live up

---

Image mentale

REST ressemble à ceci :

“Je veux lire, créer, modifier ou supprimer une ressource.”

JSON-RPC ressemble à ceci :

“Je veux appeler cette fonction sur le serveur.”

Exemple mental :

```
Application → Serveur : appelle inventory.transferItem
Serveur → Application : transfert effectué
```

Conclusion

JSON-RPC 2.0 est un protocole simple pour appeler des méthodes sur un serveur avec du JSON.

Il est particulièrement pratique pour les applications métier où les actions sont nombreuses et importantes.

Pour une application interne, un tableau de bord, un outil d’administration ou une application de gestion, JSON-RPC peut être un très bon choix.

Il faut simplement rester strict sur trois points :

1. donner des noms de méthodes clairs ;
2. valider correctement les paramètres ;
3. documenter les résultats et les erreurs possibles.

Bien utilisé, JSON-RPC est simple, propre et efficace.

Revision #2
Created 2026-07-07 12:58:35 UTC by Larananas
Updated 2026-07-07 13:03:19 UTC by Larananas