Sérialisation

La sérialisation (ou marshalling en anglais) est un processus fondamental en informatique qui consiste à convertir des données ou des objets en une séquence de bits, de façon à pouvoir les transmettre ou les stocker de manière efficace. Cette technique permet d'encoder une structure de données complexe en un format compact, souvent sous forme de texte (comme le JSON ou XML) ou de binaire.

Prenons un exemple concret.

Nous avons un modèle de données pour des séries TV:

En C#, la classe correspondante à l'acteur est:

```
public class Actor
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime BirthDate { get; set; }
    public string Country { get; set; }
    public bool IsAlive { get; set; }
}
```

instancions-là dans un programme:

```
Actor Jen = new Actor
{
    FirstName = "Jennifer",
    LastName = "Aniston",
    BirthDate = new DateTime(1969, 2, 11),
    Country = "USA",
    IsAlive = true
};
```

Inspectons ses valeurs grâce au debugger:

Maintenant que notre objet est en mémoire, les diverses parties de notre programme vont pouvoir y accéder puisqu'elles partagent la même mémoire.

Nous voulons maintenant retrouver ce même objet dans la mémoire d'un autre programme (soit sur la même machine, soit ailleurs).

Nous allons donc commencer par **sérialiser** notre objet (binaire) en JSON (texte):

```
{
  "FirstName": "Jennifer",
  "LastName": "Aniston",
  "BirthDate": "1969-02-11T00:00:00",
  "Country": "USA",
  "IsAlive": true
}
```

Ce format texte est beaucoup plus facile à transmettre que le binaire. Nous pouvons le stocker dans un fichier facilement lisible et éditable, ou nous pouvons l'envoyer par réseau (http, TCP/IP) à une autre machine.

Le récepteur n'a plus qu'à **désérialiser** ce texte pour retrouver l'objet binaire avec lequel il pourra travailler dans sa mémoire à lui.

Pourquoi utiliser la sérialisation ?

La sérialisation est essentielle dans plusieurs contextes, notamment dans les systèmes distribués, le stockage de données, et la persistance d'état. Dans les applications distribuées, où différents composants de l'application sont répartis sur différents serveurs ou machines, il est crucial de pouvoir transférer des données structurées sans perdre d'information.

Par exemple, dans une architecture client-serveur, lorsqu'un client souhaite envoyer un objet complexe au serveur (comme une commande d'achat avec des informations détaillées sur le client, les produits, etc.), la sérialisation permet d'encoder cet objet pour le transférer via un réseau en utilisant des protocoles comme HTTP ou TCP/IP. Le serveur peut ensuite désérialiser l'objet pour en extraire les données et les traiter.

Quels sont les formats les plus connus ?

La sérialisation repose sur des formats d'encodage standardisés. JSON (JavaScript Object Notation) et XML (Extensible Markup Language) sont des formats texte populaires pour la sérialisation de données.

JSON est largement utilisé pour les communications réseau, notamment dans les API RESTful, en raison de sa simplicité et de sa légèreté par rapport à XML.

Les formats binaires comme Protocol Buffers (de Google) et Avro (d'Apache) sont également populaires pour la sérialisation, surtout lorsque la performance est cruciale, car ils produisent des fichiers plus compacts et plus rapides à lire et écrire que les formats texte.

Binaire VS Texte

Il y a plusieurs raisons pour lesquelles on pourrait choisir de ne pas utiliser la sérialisation en texte (JSON, XML) :

- La sérialisation de texte prend plus de place en mémoire que la sérialisation binaire, ce qui peut être un problème si vous avez besoin de stocker ou de transmettre de grandes quantités de données.
- La sérialisation de texte peut être plus lente à sérialiser et désérialiser que la sérialisation binaire, ce qui peut être un problème si vous avez besoin de traiter rapidement de grandes quantités de données.

- La sérialisation de texte peut ne pas être aussi efficace pour les données complexes ou volumineuses, car elle peut nécessiter plus de temps et de ressources pour être traitée.
- La sérialisation de texte n'est pas lisible par l'homme, ce qui peut rendre le débogage et la maintenance plus difficiles.
- La sérialisation de texte peut être plus difficile à transférer entre différentes plateformes et systèmes d'exploitation, car elle peut nécessiter des conversions de format ou de codage des caractères.

Elle a donc surtout l'avantage d'être **lisible par les humains**.

Comment fonctionne la sérialisation ?

La sérialisation prend un objet, le « parcourt » pour en extraire ses attributs et les traduit en un format de données conforme au schéma de sérialisation choisi. Par exemple, un objet Java peut être converti en une chaîne JSON contenant ses attributs et leurs valeurs correspondantes. En sens inverse, la désérialisation (ou unmarshalling) permet de recréer l'objet initial en lisant le flux sérialisé.

Avantages de la sérialisation

- Transport des données : La sérialisation facilite l'envoi de données sur les réseaux, particulièrement dans les applications Web et les API, permettant ainsi l'interopérabilité entre différents systèmes.
- Persistances des objets : Elle permet de sauvegarder l'état d'un objet pour le restaurer plus tard. Par exemple, un programme peut sauvegarder l’état de ses objets en les sérialisant dans un fichier, pour récupérer cet état lors d’une exécution ultérieure.
- Compatibilité multiplateforme : Les formats de sérialisation comme JSON et XML sont lisibles par de nombreux langages de programmation, ce qui facilite l’échange de données entre des systèmes hétérogènes.

Défis et limites de la sérialisation

Malgré ses avantages, la sérialisation présente aussi des défis.

- Les dates et heures. Les formats varient d'un pays à l'autre, d'une culture à l'autre. Chaque programmeur a envie de travailler avec un format qui lui est familier. Il est capital de se mettre d'accord sur un format au sein d'un système distribué. Le format reconnu mondialement est ISO 8601
- La sécurité est un problème majeur, car la désérialisation de données provenant de sources externes peut introduire des vulnérabilités, notamment des attaques d'injection.
- La performance peut être un problème si les objets sont trop complexes ou si le volume de données est important, en particulier avec les formats texte qui génèrent de plus grands volumes de données par rapport aux formats binaires.
- La sérialisation peut également poser des problèmes de compatibilité des versions : si la structure de l'objet est modifiée entre la sérialisation et la désérialisation, il peut être difficile de reconstruire l’objet correctement.