Serialização de objetos

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

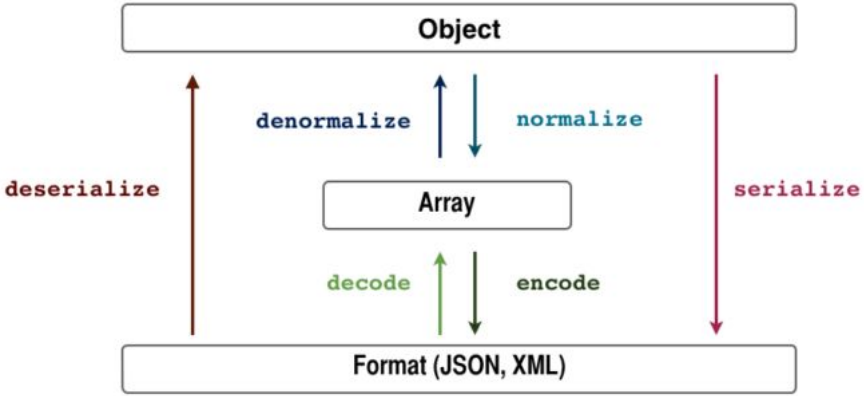
Slides baseados no material do Professor Régis Magalhães

Agenda

- Serialização
- Serialização x Desserialização
- Exemplo de Serialização e Desserialização com pickle
- Exemplo de Serialização e Desserialização com JSON
- Exemplo de Serialização e Desserialização com YAML
- Exemplo de Serialização e Desserialização com CSV
- Exemplo de Serialização e Desserialização com TOML
- Resumo das Diferenças

Serialização

- Processo de converter uma estrutura de dados ou objeto (como um dicionário, lista, ou classe) em um formato que pode ser facilmente armazenado em um arquivo ou transmitido pela rede. Esse formato pode ser JSON, XML, YAML, ou mesmo binário.



Serialização

- A serialização é o processo de converter um objeto Python em um formato que pode ser facilmente salvo em um arquivo, transmitido pela rede ou armazenado em um banco de dados. Esse processo é útil quando você precisa:
  - Salvar o estado de um objeto em um arquivo.
  - Enviar dados entre diferentes partes de um sistema.
  - Transferir dados entre sistemas distintos ou aplicativos.

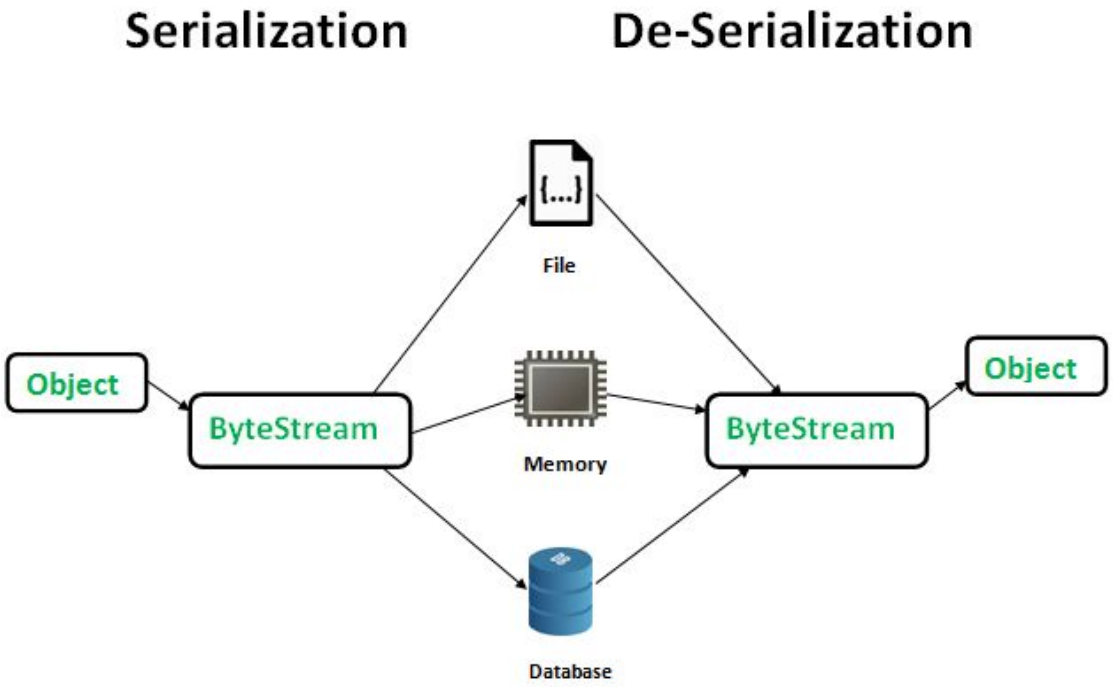
- Em Python, a serialização é frequentemente feita usando os módulos **pickle**, **json** e **yaml**.

Desserialização

- A desserialização é o processo inverso: ela pega um objeto serializado (armazenado em um formato específico) e o converte de volta ao seu estado original em Python. Assim, um arquivo ou um stream de dados que foi serializado pode ser carregado como um objeto Python.

- Principais Formatos de Serialização e Desserialização em Python
  - **pickle:** Serializa objetos Python complexos em um formato binário.
  - **json:** Serializa e desserializa objetos em formato JSON, ideal para dados simples e interoperabilidade com outras linguagens e sistemas.
  - **yaml:** Serializa e desserializa dados em formato YAML, que é fácil de ler e muito usado em configurações.

Serialização x Desserialização



Exemplo de Serialização com pickle

```python
import pickle

# Exemplo de objeto a ser serializado
dados = {"nome": "Alice", "idade": 25, "cursos": ["Python", "Data Science"]}

# Serializar o objeto e salvar em um arquivo
with open("dados.pkl", "wb") as file:
    pickle.dump(dados, file)
```

- Define um dicionário `dados` com informações (nome, idade, cursos).
- Abre o arquivo `dados.pkl` em modo de escrita binária (`wb`).
- Serializa o dicionário `dados` com `pickle.dump` e o salva no arquivo `dados.pkl`.
- O arquivo agora contém o objeto `dados` em formato binário para futura recuperação.

Exemplo de Desserialização com pickle

```
import pickle

# Ler o arquivo e desserializar o objeto
with open("dados.pkl", "rb") as file:
    dados_carregados = pickle.load(file)

print("Dados desserializados:", dados_carregados)
```

- Abre o arquivo `dados.pkl` em modo de leitura binária (`rb`).
- Usa `pickle.load(file)` para ler e desserializar o conteúdo do arquivo, restaurando o objeto `dados_carregados` ao seu estado original.
- Imprime o objeto carregado com `print("Dados desserializados:", dados_carregados)`.
- Assim, `dados_carregados` contém os dados do objeto salvo anteriormente em `dados.pkl`.

Exemplo de Serialização com JSON

```python
import json

dados = {"nome": "Alice", "idade": 25, "cursos": ["Python", "Data Science"]}

# Serializar para JSON e salvar em um arquivo
with open("dados.json", "w") as file:
    json.dump(dados, file)
```

- Cria um dicionário `dados` com informações (nome, idade e cursos).
- Abre o arquivo `dados.json` em modo de escrita (`w`).
- Usa `json.dump(dados, file)` para converter o dicionário `dados` para o formato JSON e salvá-lo no arquivo `dados.json`.
- Esse processo permite salvar os dados em um formato legível por humanos e amplamente compatível com outros sistemas.

Exemplo de Desserialização com JSON

```python
import json

# Ler o arquivo e desserializar os dados JSON
with open("dados.json", "r") as file:
    dados_carregados = json.load(file)

print("Dados desserializados do JSON:", dados_carregados)
```

- Abre o arquivo `dados.json` em modo de leitura (`r`).
- Usa `json.load(file)` para ler o conteúdo do arquivo e desserializar o JSON, convertendo-o de volta para um dicionário Python e armazenando-o em `dados_carregados`.
- Imprime o conteúdo do dicionário com `print("Dados desserializados do JSON:", dados_carregados)`.
- Esse processo restaura os dados salvos em JSON de volta ao formato original em Python.

Exemplo de Serialização com YAML

```
import yaml

dados = {"nome": "Alice", "idade": 25, "cursos": ["Python", "Data Science"]}

# Serializar para YAML e salvar em um arquivo
with open("dados.yaml", "w") as file:
    yaml.dump(dados, file)
```

- Define um dicionário `dados` com informações (nome, idade e cursos).
- Abre o arquivo `dados.yaml` em modo de escrita (`w`).
- Usa `yaml.dump(dados, file)` para converter o dicionário `dados` para o formato YAML e salvá-lo no arquivo `dados.yaml`.
- Esse processo armazena os dados em um formato legível por humanos, ideal para arquivos de configuração.

Exemplo de Desserialização com YAML

```
import yaml

# Ler o arquivo YAML e desserializar
with open("dados.yaml", "r") as file:
    dados_carregados = yaml.safe_load(file)

print("Dados desserializados do YAML:", dados_carregados)
```

- Abre o arquivo `dados.yaml` em modo de leitura (`r`).
- Usa `yaml.safe_load(file)` para ler o conteúdo do arquivo e desserializar o YAML, convertendo-o de volta para um dicionário Python e armazenando-o em `dados_carregados`.
- Imprime o conteúdo do dicionário com `print("Dados desserializados do YAML:", dados_carregados)`.
- Esse processo restaura os dados armazenados em YAML para o formato original em Python, permitindo seu uso posterior.

Exemplo de Serialização com CSV

```python
import csv
# Dados de exemplo
dados = [
    {"nome": "Alice", "idade": 25, "curso": "Python"},
    {"nome": "Bob", "idade": 30, "curso": "Data Science"},
    {"nome": "Charlie", "idade": 22, "curso": "Machine Learning"}
]

# Serializar os dados para CSV
with open("dados.csv", "w", newline="") as file:
    writer = csv.DictWriter(file, fieldnames=["nome", "idade", "curso"])
    writer.writeheader()  # Escreve os nomes das colunas
    writer.writerows(dados)  # Escreve cada linha de dados
```

- **Define os dados**: Cria uma lista de dicionários com informações de nome, idade e curso.
- **Abre o arquivo CSV**: Abre (ou cria) `dados.csv` para escrita.
- **Configura o escritor CSV**: Define as colunas (`fieldnames`) para `nome`, `idade`, `curso`.
- **Escreve o cabeçalho**: Grava os nomes das colunas no arquivo.
- **Escreve os dados**: Grava cada linha de dados no arquivo CSV.

Exemplo de Desserialização com CSV

```python
import csv

# Ler dados do arquivo CSV
with open("dados.csv", "r") as file:
    reader = csv.DictReader(file)
    dados = [row for row in reader]

print(dados)
```

- **Abre o arquivo CSV**: Abre `dados.csv` em modo de leitura.
- **Lê os dados**: Usa `csv.DictReader` para ler cada linha como um dicionário e armazena em `dados`.
- **Exibe os dados**: Imprime o conteúdo de `dados`.

Exemplo de Serialização com TOML

```
#pip install toml

import toml

# Dados de exemplo
dados = {
    "pessoa": {
        "nome": "Alice",
        "idade": 25,
        "cursos": ["Python", "Data Science"]
    },
    "configuracoes": {
        "tema": "escuro",
        "notificacoes": True
    }
}
```

```
# Serializar os dados para um arquivo TOML
with open("dados.toml", "w") as file:
    toml.dump(dados, file)
```

- **Define os dados**: Cria um dicionário com informações de pessoa e configurações.
- **Abre o arquivo TOML**: Abre (ou cria) `dados.toml` para escrita.
- **Serializa os dados**: Grava o dicionário `dados` em formato TOML no arquivo.

Exemplo de Desserialização com TOML

```python
import toml

# Ler dados do arquivo TOML
with open("dados.toml", "r") as file:
    dados = toml.load(file)

print(dados)
```

- **Abre o arquivo `dados.toml`** em modo de leitura.
- **Carrega o conteúdo do arquivo** em um dicionário Python.
- **Exibe o dicionário** no console com `print(dados)`.

Resumo das diferenças

- Vantagens de serialização e desserialização
  - **Serialização** permite persistência e transmissão de dados de forma estruturada.
  - **Desserialização** permite recuperar e reutilizar os dados em um formato Python nativo.

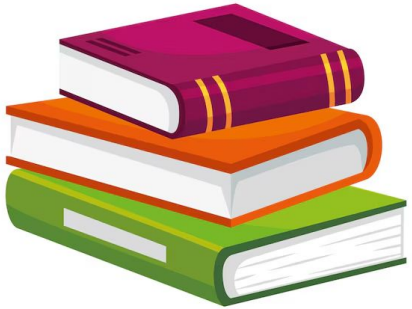
- Esses processos são essenciais para manipular dados em diferentes contextos, como armazenamento e comunicação entre sistemas.

Bibliografia Básica

- SADALAGE, P. J. E FOWLER, M. NoSQL Essencial. Editora Novatec, São Paulo, 2013.
- REDMOND, E.; WILSON, J. R. Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. 1ª edição, 2012. The Pragmatic Programmers.
- ULLMAN, J.D.; WIDOW, J. First Course in Database Systems. 3a edição, 2007. Prentice Hall.
- HAMBRICK, G. et al. Persistence in the Enterprise: A Guide to Persistence Technologies; 1ª edição, 2008. IBM Press.
- ELMASRI, R.; NAVATHE, S. B. Sistemas de banco de dados. 4ª edicao, 2009. Pearson/Addison-Wesley.

Bibliografia Complementar

- WHITE, Tom. Hadoop: the definitive guide. California: O´Reilly, 2009. xix, 501 p. ISBN 9780596521974 (broch.).
- AMBLER, S.W., SADALAGE, P.J. Refactoring Databases: Evolutionary Database Design. 1a edição, 2011. Addison Wesley.
- SILBERSCHATZ, A.; SUDARSHAN, S. Sistema de banco de dados. 2006. Campus.
- LYNN, B. Use a cabeça! SQL. 1ª edição, 2008. ALTA BOOKS.
- SMITH, Ben. JSON básico: conheça o formato de dados preferido da web. São Paulo: Novatec, 2015. 400 p. ISBN 9788575224366 (broch.).
- HITZLER, P., KRÖTZSCH, M., and RUDOLPH, S. (2009). Foundations of Semantic Web Technologies. Chapman & Hall/CRC.
- ANTONIOU, G. and HARMELEN, F. (2008). A Semantic Web Primer. Second Edition,Cambridge, MIT Press, Massachusetts.
- HEATH, T. and BIZER, C. (2011). Linked Data: Evolving the Web into a Global Data Space. Morgan & Claypool, 1st edition.



Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br