JSON

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

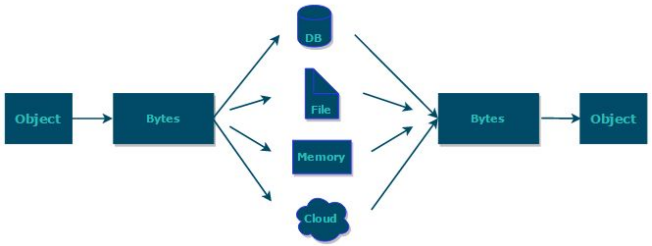
- Serialização
- Tipos Javascript
- instance of
- Objetos Javascript
- JSON - JavaScript Object Notation
- Chaves
- stringify
- toJSON
- parse
- Armazenando e recuperando dados
- Validação sintática
- Validação sintática + semântica
- JSON Schema
- Bibliotecas para JSON em python

Serialização

- Processo de capturar uma estrutura de dados de maneira a permitir que ela seja armazenada, transmitida e reconstruída novamente em uma estrutura de dados no futuro.

- Serialização em JSON é o processo de converter objetos ou estruturas de dados em uma string no formato JSON (JavaScript Object Notation), um padrão leve e legível usado para transmitir informações entre sistemas.

- É amplamente utilizado em APIs, permitindo que dados sejam facilmente armazenados, compartilhados e interpretados por diferentes linguagens de programação.



Tipos Javascript

- JavaScript possui tipos primitivos e tipos não primitivos que definem como os dados são manipulados e armazenados.

- **Primitivos:** São os tipos básicos e imutáveis, representando valores simples.
  - números, strings, booleanos, undefined, null.

- **Tipos Não Primitivos:** São objetos e estruturas mais complexas, usados para organizar dados e comportamentos:
  - Object, Array e Function

Tipos Javascript

- **Literais:** São representações fixas de valores nos códigos:
  - Números: 42
  - Strings: "Texto"
  - Arrays: [1, 2, 3]
  - Objetos: { chave: "valor" }

- **Operadores para tipos**
  - **typeof:** Retorna o tipo de dado de uma variável
  - **instanceof:** Verifica se um objeto é instância de um tipo específico

Tipos Javascript

- **Notações**
  - **Ponto:** Para acessar propriedades ou métodos de um objeto:

```
const obj = { nome: "Victor" };
console.log(obj.nome); // "Victor"
```

  - **Colchetes:** Para acessar propriedades usando strings ou variáveis:

```
const obj = { nome: "Victor" };
console.log(obj["nome"]); // "Victor"
```

instance of

- Verificação de instâncias:
  - **mycar instanceof Car:** Confirma que mycar foi criado pela função Car (retorna true).
  - **mycar instanceof Object:** Verifica que mycar é um objeto JavaScript (retorna true).

```
function Car(make, model, year) {
    this.make = make;
    this.model = model;
    this.year = year;
}

// Criando um novo carro
var mycar = new Car('Honda', 'Accord', 1998);

// Verificando as instâncias
var a = mycar instanceof Car; // retorna true
var b = mycar instanceof Object; // retorna true
```

instance of

- Verificação de instâncias:
  - **o instanceof C:** Retorna true porque o protótipo de o é C.prototype.
  - **o instanceof D:** Retorna false porque D.prototype não está na cadeia de protótipos de o.
  - **o instanceof Object:** Retorna true porque todos os objetos herdam de Object.
  - **Verificação de protótipo:** C.prototype instanceof Object retorna true porque C.prototype também é um objeto.

```
// Definindo construtores
function C() {}
function D() {}

// Criando uma instância do construtor C
var o = new C();

// Verificando as instâncias
console.log(o instanceof C); // true, pois
Object.getPrototypeOf(o) === C.prototype
console.log(o instanceof D); // false, porque D.prototype não está
na cadeia de protótipos de o
console.log(o instanceof Object); // true, porque todos os objetos
derivam de Object

// Verificando a relação entre C.prototype e Object
console.log(C.prototype instanceof Object); // true
```

Array Javascript

- JavaScript oferece estruturas flexíveis para armazenar e manipular dados, sendo os arrays e objetos as mais comuns.

- **Array**
  - É uma **lista ordenada** de valores, indexados por números inteiros começando do 0.
  - Representado por **colchetes** ([]).
  - Pode conter qualquer tipo de dado, inclusive outros arrays ou objetos.

```
const lista = [1, "texto", true, { chave: "valor" }];
console.log(lista[1]); // "texto"
```

Objetos Javascript

- **Objeto**
  - É uma **coleção de pares chave/valor**, onde cada chave é única.
  - Representado por **chaves** ({}).
  - As chaves (ou propriedades) podem ser strings, e os valores podem ser de qualquer tipo.

```
const pessoa = {
    nome: "Pedro",
    idade: 30,
    ativo: true
  };
  console.log(pessoa.nome); // "Pedro"
```

Coleções Javascript

- **Coleções Não Ordenadas**
  - Objetos em JavaScript são não ordenados, ou seja, a ordem das chaves não é garantida.
  - Isso significa que, ao iterar sobre as chaves de um objeto, a sequência pode variar dependendo do ambiente ou versão do JavaScript.

```
const objeto = { b: 2, a: 1, c: 3 };
for (let chave in objeto) {
  console.log(chave); // Pode imprimir: "b", "a", "c" (ordem não garantida)
}
```

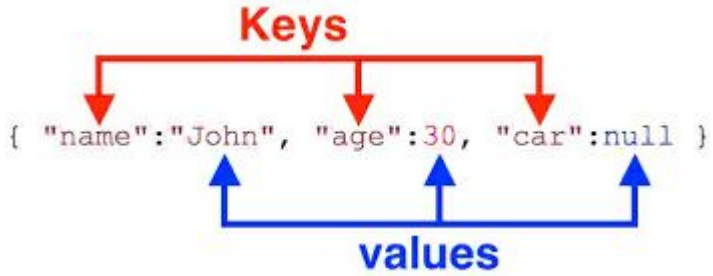
JSON - JavaScript Object Notation

- Formato leve usado para armazenamento e troca de dados.
- Auto-descritivo e de fácil entendimento.
- Independente de linguagem.
  - Usa sintaxe Javascript, mas é um formato texto que pode ser lido e usado por qualquer linguagem.
- Tipos:
  - Array, Boolean, null, Number, Object, String.

Chaves

- Chaves JSON devem ser strings entre aspas duplas.
  - { "name":"John" }

- Em JavaScript chaves podem ser strings, números ou nomes de identificadores.
  - { name:"John" }



stringify

- O método JSON.stringify() converte um valor JavaScript em uma string no formato JSON, ideal para armazenamento ou transmissão de dados.

```
JSON.stringify(value[, replacer[, space]]);
```

- **value:** O valor a ser convertido (geralmente um objeto ou array).
- **replacer (opcional):** Uma função ou array que filtra quais propriedades devem ser incluídas na string JSON.
- **space (opcional):** Um número ou string usado para inserir espaçamento na saída (para fins de formatação legível).

stringify - exemplo básico

- Observações
  - **Propriedades ignoradas:** Valores undefined, funções ou símbolos não são incluídos na string JSON.
  - **Recursão circular:** Objetos que referenciam a si mesmos geram erro ao serializar.
  - **Datas:** São convertidas para strings ISO 8601.

```
var myObj = { "name": "John", "age": 31, "city": "New York" };
var myJSON = JSON.stringify(myObj);

console.log(myJSON);
// Saída: {"name":"John","age":31,"city":"New York"}
```

toJSON

- Usado para serialização quando stringify for usado.

```
var obj = {
    foo: 'foo',
    toJSON: function() {
        return 'bar';
    }
};

// Serialização do objeto usando JSON.stringify
console.log(JSON.stringify(obj)); // '"bar"'
// Quando o objeto é parte de outro, o método toJSON ainda é utilizado
console.log(JSON.stringify({ x: obj })); // '{"x":"bar"}'
```

- O objeto `obj` possui uma propriedade `foo` e um método `toJSON` que define como ele deve ser serializado.
- **Serialização simples**: `JSON.stringify(obj)` retorna `"bar"`, pois o método `toJSON` substitui a saída padrão.
- **Parte de outro objeto**: Mesmo quando `obj` está dentro de outro objeto, como em `{ x: obj }`, o método `toJSON` é utilizado, resultando em `{"x":"bar"}`.

parse

- O método JSON.parse() é usado para converter uma string JSON válida em um objeto JavaScript.
- Ele analisa a string fornecida como entrada e constrói um objeto correspondente.
- A string deve estar em formato JSON válido, caso contrário, será lançada uma exceção (SyntaxError).

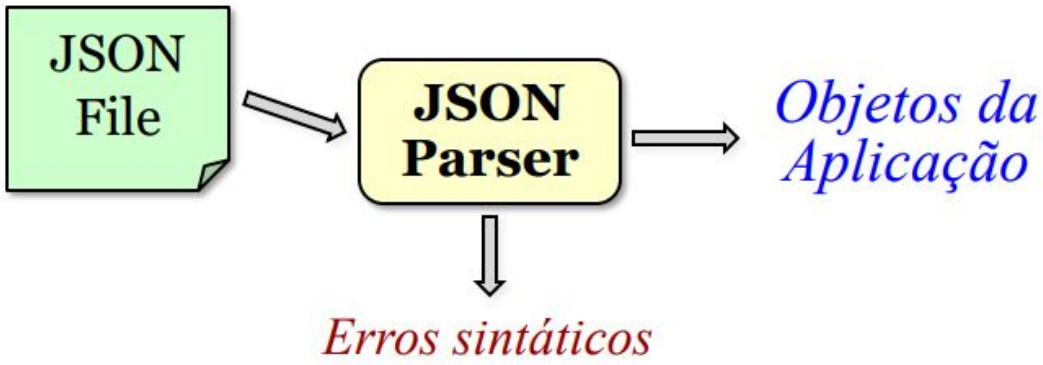
```
// Definindo um JSON como string
var myJSON = '{ "name": "John", "age": 31, "city": "New York" }';

// Convertendo o JSON para um objeto JavaScript
var myObj = JSON.parse(myJSON);

// Exibindo a propriedade 'name' no elemento com id 'demo'
document.getElementById("demo").innerHTML = myObj.name;
```

Armazenando e recuperando dados

```javascript
// Criando um objeto JavaScript
var myObj = {
    name: "John",
    age: 31,
    city: "New York"
};

// Convertendo o objeto em uma string JSON
var myJSON = JSON.stringify(myObj);

// Salvando a string JSON no localStorage
localStorage.setItem("testJSON", myJSON);

// Recuperando a string JSON do localStorage
var text = localStorage.getItem("testJSON");

// Convertendo a string JSON de volta para um objeto JavaScript
var obj = JSON.parse(text);

// Exibindo a propriedade 'name' no elemento com id 'demo'
document.getElementById("demo").innerHTML = obj.name;
```

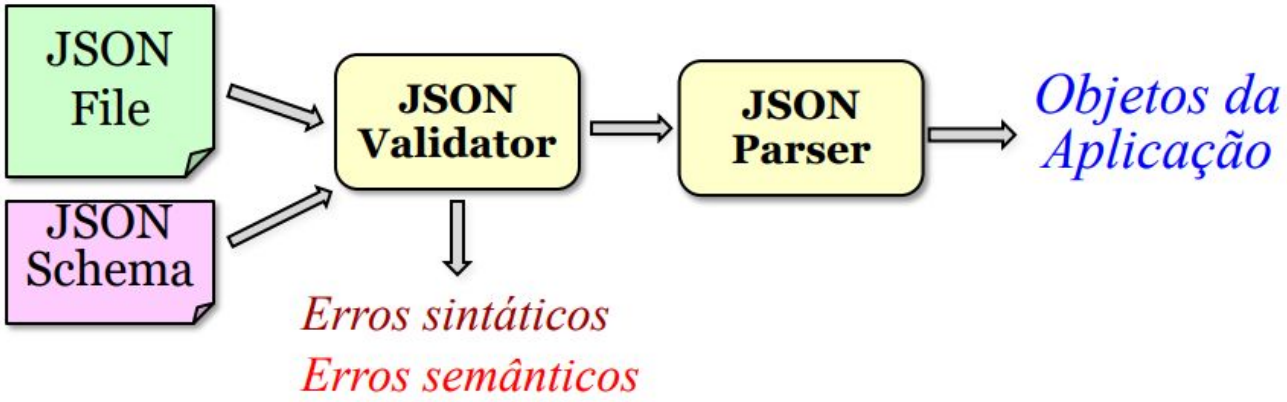
- **Objeto JavaScript**: `myObj` é um objeto com propriedades `name`, `age` e `city`.
- **Serialização**: `JSON.stringify(myObj)` converte o objeto em uma string JSON para armazená-lo no `localStorage`.
- **Armazenamento**: A string JSON é salva no `localStorage` com a chave `"testJSON"`.
- **Recuperação**: A string JSON é recuperada do `localStorage` com `getItem`.
- **Deserialização**: `JSON.parse(text)` transforma a string JSON de volta em um objeto.
- **Exibição**: O valor da propriedade `name` do objeto é exibido no elemento HTML com `id="demo"`.

Validação sintática



- A validação sintática no contexto do código acima é verificar se as entradas que passam por serialização (JSON.stringify) e desserialização (JSON.parse) estão em formato JSON válido e não contêm erros de estrutura.

Validação sintática + semântica

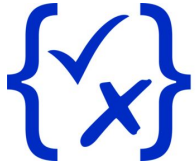


- A validação sintática e semântica é essencial para garantir que um JSON seja não apenas bem-formado (sintaticamente válido) mas também contenha valores que atendam aos requisitos esperados pelo sistema (semântica correta).

JSON Schema

- O JSON Schema é uma especificação para validar e definir a estrutura de documentos JSON. Ele é utilizado para descrever como um objeto JSON deve ser estruturado, incluindo tipos de dados, valores permitidos, formatos e restrições.

- Principais Elementos de um JSON Schema
  - **$schema:** Indica a versão do JSON Schema utilizada.
  - **type:** Define o tipo do dado (ex.: object, array, string, number).
  - **properties:** Descreve as propriedades de um objeto JSON e seus tipos.
  - **required:** Lista as propriedades obrigatórias.
  - **enum:** Restringe os valores permitidos a uma lista específica.
  - **items:** Define os tipos de dados dentro de um array.
  - **format:** Valida formatos específicos, como e-mails, URLs e datas.
  - **additionalProperties:** Controla se propriedades extras são permitidas.

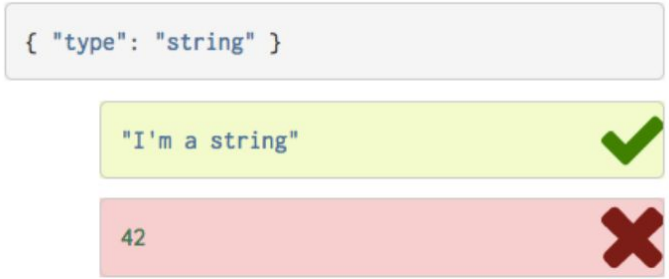


JSON Schema

Exemplo Completo de um JSON Schema

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "type": "object",
    "properties": {
      "id": { "type": "integer" },
      "name": { "type": "string" },
      "email": { "type": "string", "format": "email" },
      "roles": {
        "type": "array",
        "items": { "type": "string", "enum": ["admin", "user",
"guest"] }
      },
      "isActive": { "type": "boolean" }
    },
    "required": ["id", "name", "email"]
  }
```

- Definindo a estrutura para um objeto JSON representando um usuário.
- JSON Schema é um vocabulário que permite anotar e validar documentos JSON



Validação com JSON Schema

```
const Ajv = require("ajv");
const ajv = new Ajv();
const validate = ajv.compile(schema);

const data = { id: 1, name: "John", email:
"john@example.com", roles: ["user"], isActive: true
};
const valid = validate(data);

if (!valid) console.log(validate.errors);
else console.log("Dados válidos!");
```

- Ferramentas como AJV ou JSON Schema Validator podem ser usadas para validar objetos JSON contra um esquema.

- Benefícios do JSON Schema
  - Validação: Garante que o JSON segue um formato pré-definido.
  - Documentação: Serve como contrato entre sistemas.
  - Automação: Facilita a geração de formulários e APIs baseados em esquemas.

Bibliotecas para JSON em python

- A biblioteca nativa do Python para manipulação de JSON.
  - Principais funcionalidades:
    - Serialização: json.dumps()
    - Desserialização: json.loads()
    - Manipulação de arquivos JSON: json.dump() e json.load()

```
import json

data = {"name": "John", "age": 30}
json_string = json.dumps(data)  # Serializar
print(json_string)  # {"name": "John", "age": 30}

parsed_data = json.loads(json_string)  # Desserializar
print(parsed_data["name"])  # John
```

Bibliotecas para JSON em python

- simplejson
  - Uma biblioteca externa, similar ao módulo json, mas com melhor desempenho e mais opções de configuração.

- Instalação:
  - pip install simplejson

```
import simplejson as json
data = {"name": "Alice", "age": 25}
json_string = json.dumps(data, indent=4)  # Serializar com indentação
```

Bibliotecas para JSON em python

- ujson (UltraJSON)
  - Uma alternativa de alto desempenho para trabalhar com JSON.
  - Benefícios: Muito mais rápido que a biblioteca padrão json.

- Instalação:
  - pip install ujson

```
import ujson
data = {"name": "Charlie", "age": 28}
json_string = ujson.dumps(data)
```

Bibliotecas para JSON em python

- jsonschema
  - Utilizada para validar objetos JSON contra esquemas JSON (JSON Schema).

- Instalação:
  - pip install jsonschema

```python
from jsonschema import validate

schema = {
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "age": {"type": "integer"}
    },
    "required": ["name", "age"]
}

data = {"name": "David", "age": 22}
validate(instance=data, schema=schema)  # Validação bem-sucedida
```

Bibliotecas para JSON em python

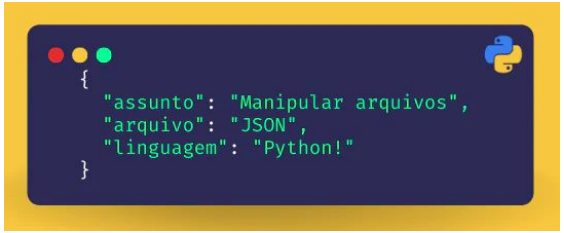
- marshmallow
  - Usada para serializar e desserializar objetos Python em JSON, além de validação e formatação.

- Instalação:
  - pip install marshmallow

```python
from marshmallow import Schema, fields

class UserSchema(Schema):
    name = fields.String(required=True)
    age = fields.Integer()

schema = UserSchema()
data = schema.dump({"name": "Frank", "age": 35})  # Serializar
print(data)
```

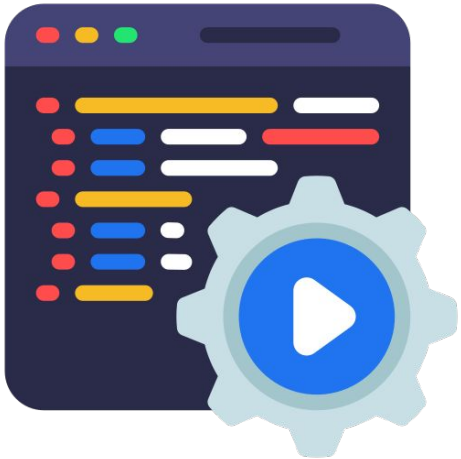
Bibliotecas para JSON em python

- Escolha da Biblioteca
  - Simples e padrão: Use json.
  - Desempenho: Prefira ujson ou orjson.
  - Validação: Utilize jsonschema.
  - Serialização avançada: Considere marshmallow.



Referências

- SMITH, Ben. JSON básico: conheça o formato de dados preferido da web. São Paulo: Novatec, 2015.
- https://en.wikipedia.org/wiki/JSON
- https://www.w3schools.com/js/js_json_intro.asp
- https://spacetelescope.github.io/understanding-json-schema/
- http://json-schema.org/
- https://github.com/FasterXML/jackson-databind
- http://www.mkyong.com/java/jackson-2-convert-java-object-to-fromjson/
- Jackson vs Gson (com exemplos)
    - http://www.baeldung.com/jackson-vs-gso



Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

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