Transpilador Python - javaScript

Caio Lucena Colaço - 2217678

Gustavo Nunes de Norões Brito - 2123438

Lucas Siqueira - 2123112

Pablo Jereissati Guedes - 2127945

13/06/2024

Introdução

A evolução das linguagens de programação e a interconectividade entre diferentes plataformas de desenvolvimento têm impulsionado a criação de ferramentas que facilitam a transposição de código de uma linguagem para outra. Entre essas ferramentas, destaca-se o transpilador, um componente essencial no ecossistema de desenvolvimento de software moderno. Neste contexto, um projeto de transpilador que converte código Python para JavaScript apresenta desafios e oportunidades significativas. Este relatório abordará os aspectos fundamentais e as motivações por trás do desenvolvimento de um transpilador, destacando a escolha das linguagens Python e JavaScript, bem como as implicações técnicas e práticas dessa transição.

Desafios Técnicos

O processo de transpilar código não é trivial e envolve diversos desafios técnicos. Primeiramente, as duas linguagens possuem paradigmas e sintaxes diferentes. Python é uma linguagem de tipagem dinâmica, orientada a objetos e funcional, com ênfase em legibilidade e simplicidade. JavaScript, embora também seja dinamicamente tipada e suporte paradigmas orientados a objetos e funcionais, possui um modelo de execução assíncrono e uma série de particularidades no gerenciamento de memória e escopo que não estão presentes em Python.

Indentação do código

Um dos desafios mais notáveis ao transpilar código Python para JavaScript é a maneira como Python utiliza a indentação para definir blocos de código. Ao contrário de muitas outras linguagens, que utilizam chaves ({}) ou palavras-chave para delimitar blocos de código, Python utiliza a indentação. Isso significa que a estrutura hierárquica do código depende crucialmente da correta indentação.

Python:

```python
def saudacao(nome):
    if nome:
        print(f"Olá, {nome}!")
    else:
        print("Olá, visitante!")
```

JavaScript:

```javascript
function saudacao(nome) {
    if (nome) {
        console.log(`Olá, ${nome}!`);
    } else {
        console.log("Olá, visitante!");
    }
}
```

Para um transpilador, isso significa que ele deve primeiro entender a estrutura do código Python através da indentação, o que pode ser um desafio significativo, especialmente em códigos complexos com múltiplos níveis de indentação. Erros na indentação em Python podem levar a problemas sérios na execução do código transpilado, já que a semântica do programa pode ser alterada.

Remoção de comentários

Outro desafio técnico importante é o manejo dos comentários. Comentários são essenciais para a documentação e manutenção do código, mas podem interferir no processo de transpilação se não forem tratados corretamente. Em Python, comentários são iniciados com o símbolo # e podem aparecer em qualquer lugar no código.

```python
def saudacao(nome):
    # Verifica se o nome foi fornecido
    if nome:
        print(f"Olá, {nome}!")  # Saúda o usuário pelo nome
    else:
        print("Olá, visitante!")  # Saúda um visitante anônimo
```

No processo de transpilação, é frequentemente necessário remover comentários para evitar complicações na análise e na geração do código JavaScript. Isso porque a manutenção de comentários pode exigir uma lógica adicional para posicioná-los corretamente no código gerado, o que aumenta a complexidade do transpilador.

Funções e Classes

As funções e classes representam um dos componentes mais complexos no processo de transpilar código de Python para JavaScript. Em Python, as funções podem ser definidas de maneira simples e suportam conceitos avançados como decoradores e geradores. Em JavaScript, embora as funções também sejam flexíveis, a sintaxe e os padrões de uso podem diferir substancialmente.

As classes em Python são definidas usando a palavra-chave `class` e seguem um modelo orientado a objetos convencional. JavaScript, por sua vez, introduziu classes de maneira mais formal no ES6, mas possui um histórico de orientação a objetos baseado em protótipos, o que pode complicar a transição direta.

Análise Assintótica

Analisando o código construído no presente trabalho, percebemos que existem diversas funções que desempenham papéis específicos ao longo da transpilação, contudo, vemos que uma delas se sobressai sobre as demais no quesito de complexidade.

Na classe Main, temos primeiro uma mera atribuição de variável, onde são armazenados os nomes do arquivos. Em seguida, inicia-se a execução de um loop que fará a execução de várias funções responsáveis pela tradução do código, conforme indicado abaixo:

- **convertCodeToString(fileName)**: Lê o conteúdo do arquivo. Se m é o tamanho do arquivo, a complexidade é O(m)O(m)O(m).
- **removeComents(code)**: Usa uma regex para remover comentários. A regex percorre todo o código, então a complexidade é O(m)O(m)O(m).
- **splitCode(code)**: Percorre o código uma vez para dividir em linhas e calcular indentação. A complexidade é O(m)O(m)O(m).

- **addBrackets(code)**: Itera sobre as linhas do código. A complexidade é O(m)O(m)O(m).
- **tradutor(code)**: Itera sobre as linhas do código e realiza diversas substituições. A complexidade é O(m·k)O(m.k)O(m.k), onde k é a quantidade de substituições (assumindo que k é constante, podemos simplificar para O(m)O(m)O(m)).
- **gerar_arquivo_js(code, fileName)**: Escreve as linhas do código em um novo arquivo. A complexidade é O(m)O(m)O(m).
- **for i in range(0, len(code))**: Itera sobre as linhas do código e imprime. A complexidade é O(m)O(m)O(m).

Por conseguinte, a função que contém a maior complexidade é **tradutor(code)**, especialmente devido às múltiplas substituições e transformações que podem aumentar a complexidade.

Para melhorar a eficiência do código, é essencial reduzir a quantidade de operações repetidas. Isso pode ser feito combinando algumas das transformações na função tradutor, diminuindo assim o número de passagens pelo código. Com menos passagens, o desempenho geral do programa será aprimorado.

Outra sugestão é otimizar as expressões regulares usadas nas funções removeComents e tradutor. Expressões regulares mais eficientes podem acelerar o processamento de texto, tornando o código mais rápido e responsivo. Pequenas mudanças na construção das regex podem ter um impacto significativo na performance.

Além disso, é importante evitar leitura e escrita desnecessárias. Nas funções convertCodeToString e gerar_arquivo_js, a utilização de métodos de leitura e escrita em bloco

pode melhorar a eficiência do programa. Esse ajuste minimiza o tempo de I/O, resultando em um processamento mais rápido e eficiente dos arquivos.

Link do Github: https://github.com/CaioLuColaco/py-js-transpiler/tree/main