Integrazione con moduli esterni

C, C++ e Java

Docente: Marino Speciale

Indice della Lezione

- Perché integrare Python con C/C++/Java
- Integrazione con C: ctypes
- Integrazione con C++: ctypes e pybind11
- Integrazione con Java: JPype e JNI

Perché integrare Python con altri linguaggi?

- **Performance**: C/C++ sono 10-100x più veloci per calcoli intensivi
- **Librerie esistenti**: riusare codice legacy senza riscrivere tutto
- **Ecosistemi specifici**: accedere a JVM libraries da Python
- **Best of both worlds**: scripting Python + velocità nativa

ctypes: Integrazione con C/C++

- è una libreria Python standard (built-in)
- Permette di chiamare funzioni da DLL/shared libraries (.so, .dll)
- Fornisce tipi di dato compatibili con C
- Nessuna compilazione Python necessaria

Esempio: Creare libreria C

File mathlib.c :

```
// mathlib.c
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

double multiply(double x, double y) {
    return x * y;
}
Compilazione:

gcc -shared -o mathlib.so -fPIC mathlib.c
```

Compilazione:

```
gcc -shared -o mathlib.so -fPIC mathlib.c
```

Usare la libreria C da Python

```
import ctypes

# Carica la libreria condivisa
lib = ctypes.CDLL('./mathlib.so')

# Chiamata funzione add
result_add = lib.add(10, 20)
print(f"10 + 20 = {result_add}")  # 30

# Per funzioni con float/double, specifica i tipi
lib.multiply.argtypes = [ctypes.c_double, ctypes.c_double]
lib.multiply.restype = ctypes.c_double

result_mul = lib.multiply(3.5, 2.0)
print(f"3.5 * 2.0 = {result_mul}")  # 7.0
```

ctypes: Tipi di dato comuni

- c_int → int
- c_float → float
- c_double → double
- c_char_p → char*
- c_void_p → void*
- POINTER() → puntatori
- Structure → struct C

Integrazione con C++

- `ctypes`: funziona se C++ esporta C API (`extern "C"`)
- `pybind11`: soluzione più potente e moderna
- `pybind11` gestisce automaticamente conversioni di tipo
- Supporta classi, template, STL containers

C++ con extern "C" per ctypes

```
// mathlib.cpp
#include <cmath>

extern "C" {
    double calculate_distance(double x1, double y1,
                              double x2, double y2) {
        double dx = x2 - x1;
        double dy = y2 - y1;
        return sqrt(dx*dx + dy*dy);
    }
}
```

Compilazione:

```
g++ -shared -o mathlib.so -fPIC mathlib.cpp
```

JPype: Python ↔ Java Bridge

- Permette di usare classi Java direttamente da Python
- Bridge nativo tramite JNI (Java Native Interface)
- Accesso completo a librerie Java e JVM
- Installazione: pip install JPype1

JPype: Esempio di base

```
import jpype
import jpype.imports

# 1. Avvia la JVM
jpype.startJVM()

# 2. Importa classi Java
from java.lang import String, System

# 3. Usa oggetti Java
java_string = String("Hello from Java!")
print(java_string.toUpperCase())  # HELLO FROM JAVA!

System.out.println("Chiamata diretta a System.out")

# 4. Chiudi JVM alla fine
jpype.shutdownJVM()
```

JPype: Usare classi Java custom

File Java :

```
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
}
```

Compilazione e uso:

```
javac Calculator.java

import jpype
jpype.startJVM(classpath=['.'])
Calculator = jpype.JClass('Calculator')
calc = Calculator()
print(calc.add(10, 20))  # 30
```

Altre opzioni Python-Java

- **JPype**: bridge nativo, più usato e mantenuto
- **Jython**: Python su JVM (Python 2.7, progetto fermo)
- **py4j**: comunicazione tramite socket (overhead maggiore)
- **jni** (package): wrapper Python su JNI C API

Performance: Python vs C vs Java

Python puro

Baseline: 1x (più lento)

C/C++ con ctypes

10-100x più veloce

Java con JPype

5-50x più veloce

Best Practices

- Usa ctypes per funzioni C semplici e veloci
- Considera pybind11 per progetti C++ complessi
- JPype quando hai già librerie Java da riusare
- Documenta tipi di dato e conversioni

Quando NON usare integrazione nativa

- Se NumPy/Pandas già risolvono il problema
- Overhead di chiamata può superare il guadagno
- Debugging più complesso (mix linguaggi)
- Deployment più difficile (dipendenze native)

Risorse e Documentazione

- ctypes: `docs.python.org/3/library/ctypes.html`
- pybind11: `pybind11.readthedocs.io`
- JPype: `jpype.readthedocs.io`
- JNI package: `pypi.org/project/jni/`

Grazie!

Domande?

Python + C/C++/Java = potenza e flessibilità