Redis 入門

Redis の公式サイトを開くと、最初の一文には、Redis はオープンソースのインメモリ型データ構造ストレージで、データベースやキャッシュとしてよく使われると書かれています。Redis は非常に一般的です。

Redis のインストール

公式サイトから Redis をインストールできます。SQLite と同様です。インストールが完了したら、Python で Redis をどのように使用するのでしょうか。

```
pip install redis
```

```
>>> import redis
>>> r = redis.Redis(host='localhost', port=6379, db=0)
>>> r.set('foo', 'bar')
True
>>> r.get('foo')
b'bar'
```

このコードは、Python で Redis データベースに接続し、キーと値のペアを設定・取得する基本的な操作を示しています。以下に各ステップの説明を日本語で記載します。

1. `import redis`: Redis ライブラリをインポートします。
2. `r = redis.Redis(host='localhost', port=6379, db=0)`: Redis サーバーに接続します。`host` はサーバーのアドレス、`port` は接続ポート、`db` はデータベース番号を指定します。
3. `r.set('foo', 'bar')`: キー `'foo'` に値 `'bar'` を設定します。成功すると `True` が返されます。
4. `r.get('foo')`: キー `'foo'` に対応する値を取得します。値はバイト文字列として返されるため、`b'bar'` と表示されます。

Python のドキュメントにはいくつかの例が記載されています。ここで pip のようなものが登場します。pip はパッケージ管理ツールです。パッケージ管理ツールとは何かについては、「プログラミング環境に慣れる」の章を参照してください。pip は Python にとって、Homebrew が macOS システムにとっての存在に相当します。

pip は通常、Python をインストールする際に既に含まれています。もしコンピュータに複数のバージョンの Python と Pip が存在する場合、~/.bash_profile に以下の 2 行を追加することができます：

```
alias python=/usr/local/Cellar/python@3.9/3.9.1_6/bin/python3
alias pip=/usr/local/Cellar/python@3.9/3.9.1_6/bin/pip3
```

このコードは、シェルのエイリアスを設定しています。具体的には、python と pip コマンドを、指定されたパスの Python 3.9.1 の実行ファイルにリンクしています。これにより、ターミナルで python や pip と入力したときに、指定されたバージョンの Python と pip が使用されるようになります。

指定されたバージョンの python と pip をインストールする方法の一つとして、Homebrew を使用することができます。また、ソースコードからビルドしてインストールすることも可能です。

```
make
make test
make install
```

(注：上記のコードブロックはシェルコマンドであり、翻訳の必要はありません。そのままの形で使用されます。)

```
$ redis-server
87684:C 2021年3月10日 14:46:06.056 # oOOOoOOOoOOOo Redisが起動しています oOOOoOOOoOOOo
87684:C 2021年3月10日 14:46:06.056 # Redisバージョン=6.2.1，ビット=64，コミット=00000000，変更=0，PID=87684
87684:C 2021年3月10日 14:46:06.056 # 警告: 設定ファイルが指定されていません。デフォルトの設定を使用します。設定
87684:M 2021年3月10日 14:46:06.057 * 開くファイルの最大数を10032に増やしました（元々は4864に設定されていました
87684:M 2021年3月10日 14:46:06.057 * 単調増加クロック: POSIX clock_gettime
...
Redis 6.2.1 (00000000/0) 64ビット
...
87684:M 2021年3月10日 14:46:06.058 # サーバーが初期化されました
87684:M 2021年3月10日 14:46:06.058 * 接続を受け入れる準備ができました
```

以下に一部の内容を抜粋しました。これでインストールが完了したことがわかります。バージョンは 6.2.1 で、公式サイトの最新版です。別のターミナルウィンドウを開いて、試しに操作してみましょう：

```
$ redis-cli
127.0.0.1:6379> set a 2
OK
127.0.0.1:6379> get a
"2"
```

以下のコードを実行してください。

```
import redis

r = redis.Redis(host='localhost', port=6379, db=0)
r.set('foo', 'bar')
print(r.get('foo'))
```

出力：

```
$ python fib_redis.py
b'bar'
```

Redis キャッシュの例

以下是一个使用 Redis 实现斐波那契数列的示例。我们将使用 Redis 来存储已经计算过的斐波那契数，以避免重复计算。

1. 安装 Redis

首先，确保你已经安装了 Redis。你可以通过以下命令安装 Redis：

```
sudo apt-get install redis-server
```

2. 安装 Redis 客户端

我们将使用 Python 来与 Redis 进行交互。首先，安装 `redis` Python 包：

```
pip install redis
```

3. 实现斐波那契数列

接下来，我们编写一个 Python 脚本来实现斐波那契数列，并使用 Redis 来缓存结果。

```
import redis

# 连接到 Redis
```

```
r = redis.Redis(host='localhost', port=6379, db=0)

def fibonacci(n):
    # 检查 Redis 中是否已经缓存了结果
    cached_result = r.get(n)
    if cached_result:
        print(f"Cache hit for {n}")
        return int(cached_result)

    # 如果 n 小于等于 1，直接返回 n
    if n <= 1:
        result = n
    else:
        # 递归计算斐波那契数列
        result = fibonacci(n-1) + fibonacci(n-2)

    # 将结果存入 Redis
    r.set(n, result)
    print(f"Cache miss for {n}, calculated result: {result}")
    return result

# 测试
if __name__ == "__main__":
    n = 10
    print(f"Fibonacci({n}) = {fibonacci(n)}")
```

4. 运行脚本

运行上述脚本，你将看到类似以下的输出：

```
Cache miss for 0, calculated result: 0
Cache miss for 1, calculated result: 1
Cache miss for 2, calculated result: 1
Cache miss for 3, calculated result: 2
Cache miss for 4, calculated result: 3
Cache miss for 5, calculated result: 5
```

```
Cache miss for 6, calculated result: 8
Cache miss for 7, calculated result: 13
Cache miss for 8, calculated result: 21
Cache miss for 9, calculated result: 34
Cache miss for 10, calculated result: 55
Fibonacci(10) = 55
```

5. 解释

- **Redis 缓存**：我们使用 Redis 来存储已经计算过的斐波那契数。每次计算前，我们首先检查 Redis 中是否已经存在该结果。如果存在，则直接返回缓存的结果，避免重复计算。
- **递归计算**：如果 Redis 中没有缓存结果，则通过递归计算斐波那契数，并将结果存入 Redis。

6. 性能优化

通过使用 Redis 缓存，我们可以显著减少重复计算的时间，尤其是在计算较大的斐波那契数时。

7. 总结

通过结合 Redis 和 Python，我们实现了一个高效的斐波那契数列计算器。Redis 的缓存机制帮助我们避免了重复计算，从而提高了性能。

```python
import redis

r = redis.Redis(host='localhost', port=6379, db=0)

def f(n):
    nr = r.get(n)
    if nr is not None:
        return int(nr)
    res_n = 0
    if n < 2:
        res_n = n
    else:
        res_n = f(n-1) + f(n-2)
```

```
    r.set(n, res_n)
    return res_n

print(f(10))
```

これで完了です。