大きなデータ取り扱いに関する機能の作成に対して取り組んだ内容

このシステムでは、音声ファイルや画像ファイルといったデータ量が多くなるデータを大量に取り扱います。
その際に必ず気を付けなければならない点が、メモリリークへの対処です。

この資料は、メモリリークが発生しないように、大きなデータをどのように取り扱っているかを説明する
プレゼン資料となります。

目次

開発中に頻発したメモリリークの内容

開発中に、メモリリークが頻発した場面としては、以下のような状況になります。

- ユーザーからの圧縮ファイルの読み取り時
- ユーザーから渡された、音声データなどのバイナリデータの受け取り時
- ファイルサーバーからのバイナリデータの読み取り時

Javaの仮想環境である「ＪＶＭ」の設定を変えない状態では、およそ2ＧＢを超えるデータの取り扱い時に、メモリが枯渇し、「OutOfMemoryError」が発生して処理が中断されます。

なお、メモリリークへの対処は、コーディングの改善をしなくとも以下の内容で対応が可能になりますが、これでは根本的な解決にならず、今後のシステムの拡張性にも影響してしまうため見送りました。

・JVMにあてがわれるメモリ量を、設定によって増やす
・そもそも、大きなデータを取り扱わない要件定義にする

そのため、このシステムでは設定を極力変えず、「コーディングの改善」と「一時ローカルファイルの活用」によって、メモリリークを回避しております。

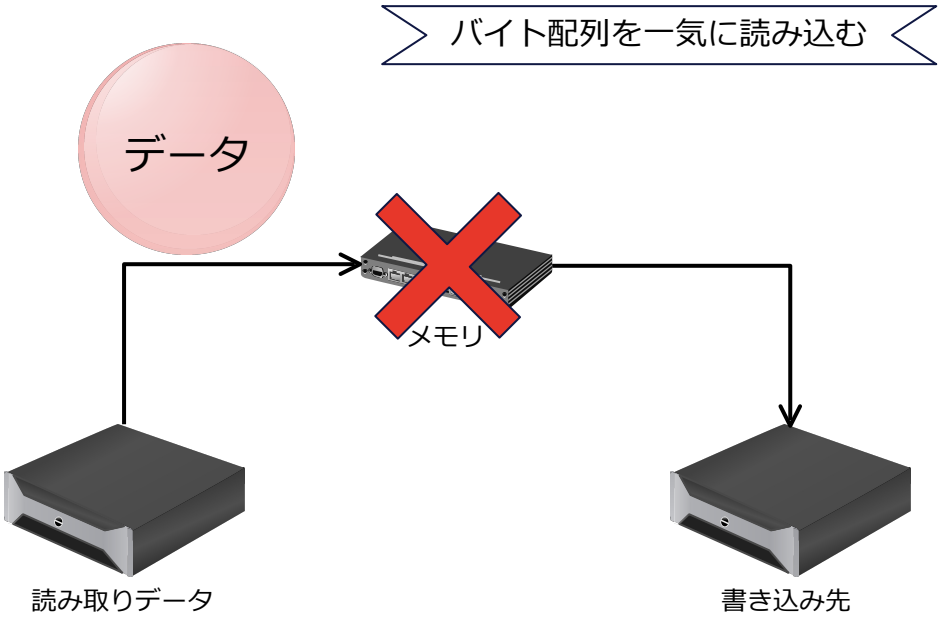
ストリームの使用の徹底

Javaでのバイナリデータの読み込みや書き込みには、大まかに分けて二つの方法があります。

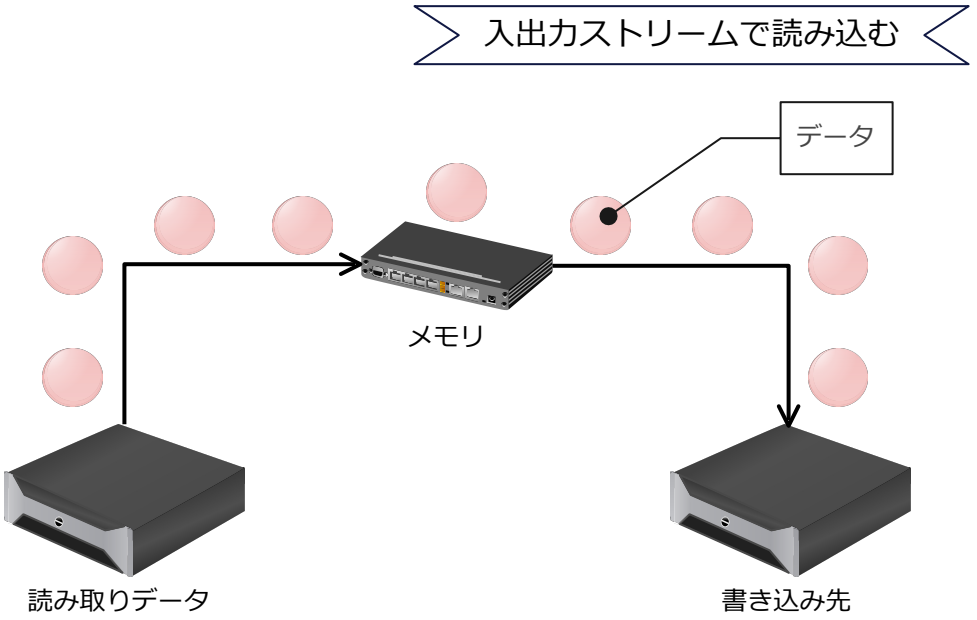
・データのバイト配列を一気にメモリに読み取って取り扱う
（例：ByteArrayInputStreamの、toByteArray()関数）

・入出力ストリームで、メモリに一気に取り込まず取り扱う
（例：諸々の、InputStreamやOutputStreamの継承クラス）

このシステムでは、入出力ストリームの使用を徹底し、メモリにデータをため込まないようにコーディングしています。



メモリがデータを格納しきれず、
オーバーフローする



メモリの格納許容量を超えないように少しづつ
読み取るので、オーバーフローが発生しない

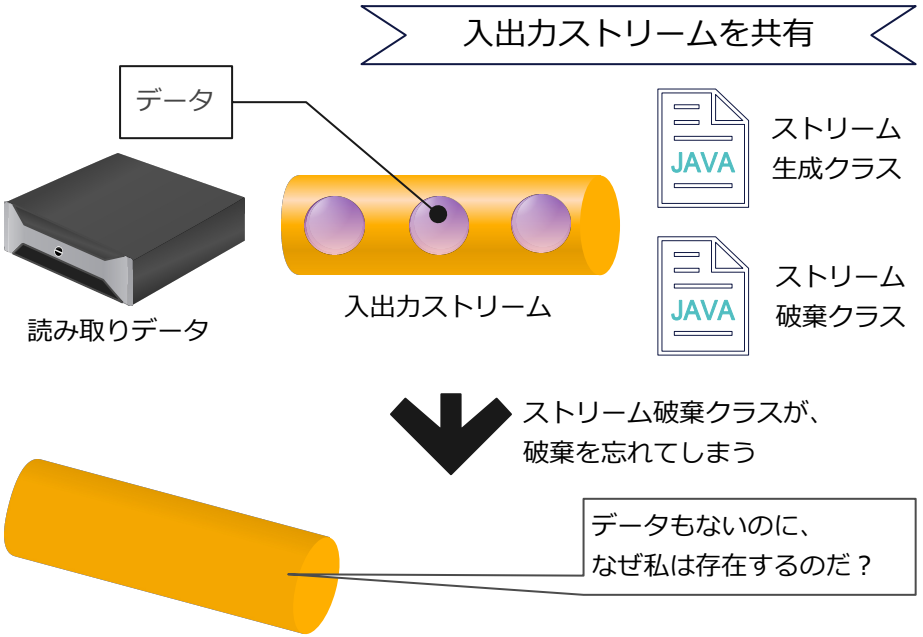
一時ローカルファイルの活用

先述した入出力ストリームでは、他のクラスや関数をまたぐデータのやり取りは、以下の理由で難しいです。

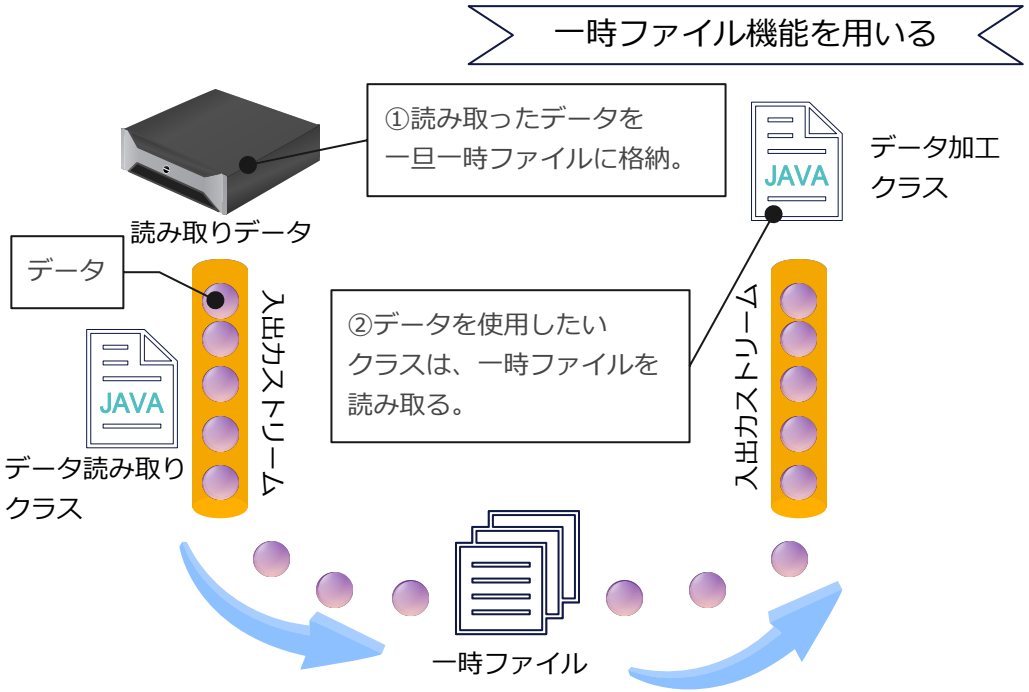
入出力ストリームの閉じ忘れによる、メモリの不足の恐れ

厳密には、入出力ストリーム自体を他のクラスや関数と共有して使うことは可能ですが、閉じ忘れが発生しやすくなることから、そのコーディング方法はアンチパターンとなっております。

一つの読み取りデータを複数の関数やクラスで共有したい場合は、Javaに標準実装されている
一時ファイル機能を用いて可能にしています。（try-with-resource文により、ストリームを自動削除）



try-with-resource文を使えないので、
ストリームが残り、メモリを食いつぶす



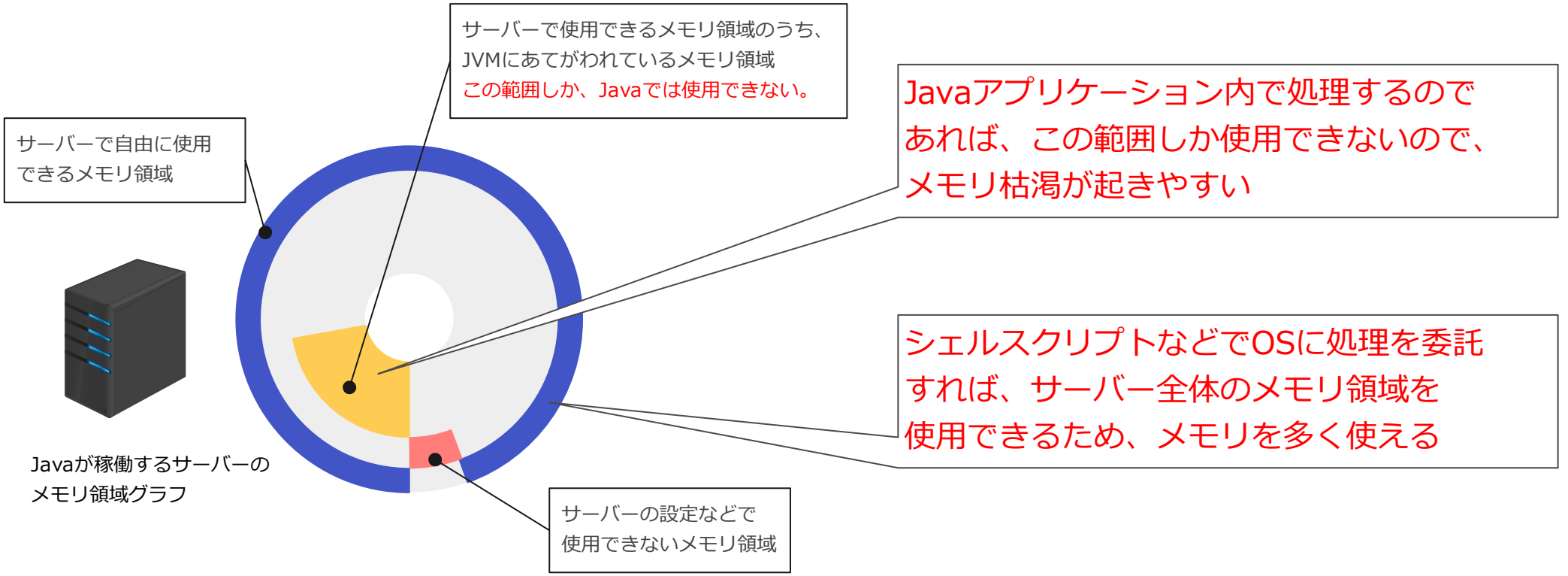
不必要なストリームが残らない

シェルスクリプトの活用

システム内の処理内容によっては、どうしてもメモリにデータをため込んだ上で処理しなくてはならない状況があります。（例：CSVファイルの管理番号の重複チェックなど）

その際は、シェルスクリプトによってLinux等のOSに処理を委託し、そちらで処理を行うようにしています。

JVMにあてがわれているメモリ量は限られておりますが、OSに処理を委託すれば使用できるメモリ量を大幅に多くして処理できるからです。



JVMのメモリ領域をあらかじめ増やしておくことも検討しましたが、ほとんど使用しないのにメモリ領域を確保しておく事は、無駄であると考えたため見送りました。