# Recording information sent between 4D Client and 4D Server

By Jean-Yves Fock-Hoon, QA Manager, 4D, Inc.

TN 06-03

## Overview

このテクニカルノートでは、4D Client と 4D Server の間で交わされたリクエスト情報を解析し、記録する方法を紹介しています。

## Purpose

4D バージョン 2004 では、SET DATABASE PARAMETER コマンドがだいぶ拡張されました。その中には ID#28 番(4D Server Log Recording)も含まれます。このオプションは、クライアント-サーバ環境で使用することができ、付加的なパラメータとして、数値をひとつ受け取ります。

- 渡された数値が 0 であれば、4D Server はログの記録を中止します。
- 渡された数値が 0 よりも大きな値であれば、その値が作成されるリクエストログファイルのファイル名に追加されます。4D Server は受け取ったリクエストがどんなものであっても、関係する情報とともに、このリクエストログファイルにその内容を記録します。ファイルは、テキスト形式で、ストラクチャファイルと同じ階層のフォルダ内に作成されます。

ファイルサイズが 10MB に達すると、そのファイルは閉じられ、通し番号を付けられた新しいファイルが作成されます。同じ名前のファイルがすでに存在する場合は上書きされます。

リクエストログファイルをエディタなどで開いたとしても、ほとんどの人は、その内容をよく理解できないはずです。このテクニカルノートでは、リクエストログファイルを解析できるように説明したいと思います。

## Structure of the file

ファイルは、テキストエディタで開くと、次のようになっています。

```
1 nth log opened on Tuesday, January 10, 2006 03:17:09 PM
time pid uid cid request bytes in bytes out duration request name
01/10/06, 15:17:09 4 26268080 26291080 189 24 11 0
01/10/06, 15:17:09 4 26268080 26291080 189 24 11 0
01/10/06, 15:17:09 4 26268080 26291080 189 24 11 0
01/10/06, 15:17:09 4 26268080 26291080 189 24 11 0
```

最初の行は、ファイルの作成された日時を示しています。次の行は、それぞれの列のタイトルに相当します。残る行には、入ってきたリクエストの情報が記録されています。前述したように、列の意味が分からないと、このファイルはなかなか理解できません。はじめに、それぞれの列のタイトルについて概観することにしましょう。

最初の列は日付で表わされています。これはリクエストが発行された日付です。上記の例では、MM/JJ/YYフォーマットで、カンマに続き、24時間表記による時間が記述されています。

**PID**は、プロセスIDを意味します。これは内部的なプロセスの識別番号です。4Dの内部的な情報なので、その実体をデベロッパがたどることはできません。このIDは、インタフェースやランタイムエクスプローラに表示されるプロセス番号とは異なります。

**UID**は、ユーザIDを意味します。これも4D Serverの内部的な情報です。サーバに接続しているユーザには、すべて固有のIDが付けられています。このIDは、Designer、Administratorといった4Dのログインユーザアカウントではなく、On Server Open Connectionデータベースメソッドの$1に返される値と同じものです。

**CID**は、接続IDを意味します。これもまた4D Serverの内部的な情報です。4D Serverはこの値によって、プロセスがどの接続に属しているのかを判別しています。同じ値は、On Server Open Connectionデータベースメソッドの$2に返されます。

**Request**は、リクエストの目的内容を示すリクエストIDです。この値を調べることによって、4D Clientが一体何を4D Serverに対してリクエストしたのかを知ることができます。IDは独自の規格によるもので、一瞥して意味を推測できないようになっています。このテクニカルノートの後半で、その意味について解説しています。

**Bytes In**は、受け取ったリクエストのサイズ、つまり受け取ったパケットのサイズです。4D Clientが送るパケットのサイズは、リクエストの内容と密接に関連しています。簡単なリクエストは小さなパケットであるのに対し、レコードのアップロードは大きなパケットになります。

**Bytes Out**は、リクエストに対する応答として送られたパケットのサイズです。4D Serverは、あらゆるリクエストに対して必ず返事を返し、この値はそのサイズを表わしています。
普通、このサイズはクライアントに伝送する情報量が多いほど大きくなり、大きなレコードを転送する場合には、それに応じたサイズとなります。

**Duration**は、ミリ秒単位で表わしたリクエストの処理時間です。時間の計測は、パラメータを含めてリクエストを受理した瞬間に開始され、サーバから応答が送信された時点で終了します。大抵のリクエストは、1ミリ秒以内で処理されるものなので、この値は0になるはずです。処理時間が長くかかるのは、プロセス数が多すぎる場合、ネットワークが詰まっている場合、大量のSELECTION TO ARRAYを実行した場合などです。

**Request name**は、デフォルトで空です。4D Serverのリソースファイルに、特定のリソースがインストールされていれば、ここからリクエスト名を知ることができます。4D Serverのリソースは、アップグレードのたびに更新されるので、現実には、リソースではなく、ストラクチャファイルまたはデータファイルでリクエスト名を管理したほうが楽です。このテクニカルノートのAppendix1には、リクエストIDとリクエスト名の対応表が掲載されています。

## How to use the file

ファイルの内容はこれで理解できるようになりましたが、それだけでは、ファイルに価値はありません。リクエストログファイルは、本来、4D開発者がコマンドの最適化をするために使用していたものでした。しかしながら、4Dデベロッパも、クライアント-サーバ間の交信量を最少限に抑え、自身のアプリケーションによって交わされる通信を監視するためにこのファイルを活用することができます。

デベロッパが、ローレベルコミュニケーションを監視しなければならない理由が何でしょうか。ネットワークは、しばしばクライアント-サーバのパフォーマンスを落とす要因になります。例えば、ネットワークが細ければ、クライアントの反応は遅くなり、タイムアウトもより頻繁に生じるようになります。一方、LANなど、ネットワークが整備されているために、あまりに多くのクライアントが同時に接続し、パフォーマンスに支障が出る場合もあります。

ログの記録は、リクエストが発行されるたびに行われるため、リクエスト数が多くなると、ディスクアクセスが追いつかず、システム全体が遅くなってしまう危険があります。このようなわけで、常時、リクエストログオプションを使用することには慎重な態度を取ることが勧められています。

メソッドによる交信を解析すると、いろいろなことが分かります。それぞれのメソッドが、どれだけのパケット通信を引き起こすのかというのは有用な情報です。それにより、パケット数の少ない、最適のメソッドに書き換えることができます。とはいえ、パケット数と処理速度は、直接、関連があるものではありません。パケット数が少ないということは、使用するネットワークリソースが少なく、内部的な処理量も少ないということなので、速度が向上する可能性は

高いといえますが、結果として他のプロセスが多くの処理を行ない、また処理するリクエスト数も増えることになるので、わずかながら処理が遅くなるというケースもあり得ます。

多数のクライアントがサーバに接続する状況で、パフォーマンスの問題を憂慮している場合、一番良いのはすべてのリクエストをよく調べてみることです。プロセスIDと接続IDでリクエストを整理してゆけば、リクエスト名からそれぞれがどのプロセスなのかは大体分かります。そうすれば、その状況は最適化の余地があるのか、それともそれだけの処理をそれだけの回数こなしていることの必然的な結果なのかを見極めることができます。

## The Demonstration database

サンプルデータベースは、ログファイルを解析するための非常にシンプルなデータベースです。SET DATABASE PARAMETERコマンドにセレクタを渡すことによって、リクエストログファイルの管理が始まります。

作成されるのは、テキストファイルで、適当なエディタで開くことができます。問題は、このファイルにはリクエスト名という形式での記録がないということ、また全体的に統計値がないということです。そこでサンプルデータベースの出番となります。

1-サンプルデータベースを4D Serverで起動し、4D Clientで接続します。
2-カスタムモードに切り替えます。
3-DemonstrationメニューからRequest Log Fileを選択します。3個のボタンを配置された小さなダイアログ画面が表示されます。
Record:このボタンをクリックすることによって、リクエストログファイルの使用をONまたはOFFにすることができます。
Clear:このボタンをクリックすることによって、ストアドプロシージャが起動し、すべてのリクエストログファイルが削除されます。
View:作成したログファイルを閲覧するために使用します。

前述したように、リクエストログファイルは複数作成されます。IDには1を使用していますが、この値そのものは重要ではなく、任意の値を使用することができます。大事なのは、サイズが10MBに達した時点で新しいリクエストログファイルが作成され、結果として複数のリクエストログファイルが存在するようになるということです。

10MBのファイルデータを複数読み込んで、リストボックス表示するのはあまり賢明なことではありません。それで、サンプルではデータを[Requests]テーブルに読み込んでいます。

Viewボタンをクリックすると、タブコントールによって様々な情報を閲覧することができます。

Raw Data: このタブには、すべてのリクエストをみることのできるリストボックスが配置されています。数10MB分をまとめて表示する代わりに、1ページに表示される件数を指定するようになっています。ページ間の移動には、画面左下のナビゲーションボタンを使用します。ヘッダをクリックすることによって、ページ単位での並び替えが実行できます。命名セレクションを使用しているので、ページごとの並び順はナビゲーション後も保持されます。このダイアログは、リストボックスで大きなセレクションを表示する方法の例題ともなっています。

Mnemonics: このタブには、ニーモニック、つまりリクエスト名による統計データが表示されます。リクエストログファイルを開く際、リクエストIDはデータベースの[Mnemonics]テーブルに従い、リクエスト名に変換されます。ダイアログでは、統計データを表示するためにクイックレポートプラグインを利用しています。つまり*M_GenerateXMLStats*メソッドによってXMLファイルが生成され、それを解析した結果がリストボックスの配列に代入されています。画面には、リクエストごとにコールされた回数、処理時間の平均および合計、パケット数などが表示されます。こうした情報は、最適化を計る上での指針となるはずです。

Processes: このタブでは、基本的に上と同じ手法を使用しています。*M_GenerateXMLStats2*メソッドでは、クイックレポートプラグインを利用して処理時間とパケットサイズの合計を接続ごと、プロセスごとに算出し、XMLファイルとして出力しています。結果は解析されて配列に代入され、リストボックスとして表示されます。プロセスごとの処理時間、データ転送量が分かるので、不自然な値があれば、そのプロセスをもっとよく調べてみることができます。

Graphs: このタブでは、5種類のグラフを作成することができます。
Bytes per Request: リクエストタイプごとのパケットサイズの合計が2次元棒グラフで表示されます。
Bytes per time: 単位時間ごとのパケットサイズの合計が2次元線グラフで表示されます。
All Durations: 時間内の全リクエストの処理時間の合計が2次元線グラフで表示されます。
% per request: リクエストタイプの割合が円グラフで表示されます。
Duration per Request: リクエストタイプごとの処理時間の合計が2次元棒グラフで表示されます。

## Summary

このテクニカルノートでは、リクエストログ機能の使用方法や、その読み取り方について簡単に取り上げました。将来のテクニカルノートでは、統計結果の評価について論じる予定です。