ログファイル

ウェブサーバを効果的に管理するためには、サーバの活動やパフォーマンス、 今発生しているかもしれない問題に関するフィードバックを得ることが必要です。 Apache HTTP サーバには非常に包括的で柔軟なロギング機能があります。 この文書はロギング機能の設定の仕方と、ログに何が書かれているかを 理解するための方法を説明します。

トピック

セキュリティに関する警告

Apache がログファイルを書いているディレクトリに書き込める人は、 ほぼ確実にサーバが起動された uid へのアクセスを手に入れることができます。 そして、それは通常は root ユーザです。 ちゃんと結果を考えることなく、そのディレクトリへの 書き込み権限を与えないでください。詳しくは セキュリティのこつ [1]の文書を 読んでください。

加えて、ログファイルにはクライアントからの情報がそのまま、 エスケープされることなく書かれています。ですから、悪意のある クライアントがログファイルに制御文字を挿入することができます。 生のログを扱うときは注意してください。

エラーログ

関連モジュール	関連ディレクティブ
	ErrorLog
	LogLevel

ErrorLog ディレクティブにより 名前と場所が決まるサーバのエラーログは、一番重要なログファイルです。 Apache の診断情報はここに送られ、リクエストを処理しているときに発生したエラーはすべてここに記録されます。サーバを起動したときや、 サーバの動作に問題が起こったときは、一番最初に調べるべき ところです。間違いの詳細や修正方法がそこに書かれていることが よくあります。

エラーログは普通はファイルに書かれます (通常 unix システムでは error_log、Windows と OS/2 では error.log)。 Unix システムではエラーを syslog や パイプでプログラムに送る ことができます。

エラーログの書式は比較的自由度の高いもので、説明的に書かれています。 ただし、いくつかの情報はほとんどのエラーログのエントリにあります。 例えば、代表的なものに次のようなメッセージがあります。

```
[Wed Oct 11 14:32:52 2000] [error] [client 127.0.0.1] client denied by server
configuration: /export/home/live/ap/htdocs/test
```

ログエントリの最初の項目はメッセージの日付と時刻です。 二つめの項目は報告されているエラーの重要度です。 LogLevel で重要度のレベルを 制限することによりエラーログに送られるエラーの種類を制御することが できます。三つ目の項目はエラーを発生させたクライアントの IP アドレス です。残りはメッセージで、この場合はサーバがクライアントのアクセスを 拒否するように設定されている、ということを示しています。 サーバはリクエストされた文書の（ウェブのパスではなく）ファイルシステムの パスを報告します。

非常に広範囲のメッセージがエラーログに現れます。たいていのものは 上の例のような感じです。エラーログには CGI スクリプトのデバッグ 出力も書かれます。CGI スクリプトが stderr に書いた すべての情報は直接エラーログにコピーされます。

情報を追加したり削除したりしてエラーログをカスタマイズすることは できません。しかし、リクエストに対するエラーログのエントリは、 対応するエントリがアクセスログにあります。 例えば、上の例のエントリはアクセスログのステータスコード 403 の エントリに対応します。アクセスログはカスタマイズ可能ですので、 そちらを使うことによりエラーの状況に関する情報をより多く 手に入れることができます。

テストの最中は、問題が発生しているかどうかを見るために、 常にエラーログを監視するのが役に立つ場合がよくあります。 Unix システムでは、次のものを使うことができます。

```
tail -f error_log
```

アクセスログ

関連モジュール	関連ディレクティブ
mod_log_config	CustomLog
mod_setenvif	LogFormat
	SetEnvIf

サーバアクセスログはサーバが処理をしたすべてのリクエストを 記録します。アクセスログの場所と内容は CustomLog ディレクティブにより決まります。ログの内容の選択を簡潔にするために LogFormat ディレクティブを使用することができます。このセクションはアクセスログに 情報を記録するためのサーバの設定方法を説明します。

もちろん、アクセスログに情報を蓄積することはログ管理の 始まりに過ぎません。次の段階は有用な統計を取るためにこの情報を 解析することです。一般的なログ解析はこの文書の範囲外で、 ウェブサーバ自身の仕事というわけでもありません。この話や、 ログ解析を行なうアプリケーションの情報を得るには、 Open Directory [2] や Yahoo [3] を調べてください。

いろんなバージョンの Apache httpd が mod_log_config, mod_log_agent, TransferLog ディレクティブといった、 他のモジュールやディレクティブを使ってアクセスのロギングを制御してきました。今では、CustomLog がすべての古い ディレクティブの機能を含むようになっています。

アクセスログの書式は非常に柔軟な設定が可能です。 書式は C の printf(1) フォーマット文字列に非常に似た フォーマット文字列 により指定されます。いくつか次の節で例を示します。 フォーマット文字列に使用できる内容の一覧は mod_log_config の文書 [4] を見てく

ださい。

Common Log Format

アクセスログのよくある設定に以下のものがあります。

```
LogFormat "%h %l %u %t ¥"%r¥" %>s %b" common
CustomLog logs/access_log common
```

これは、ニックネーム common を定義し、 ログのフォーマット文字列の一つと関連付けます。フォーマット文字列は パーセントディレクティブからなり、それぞれのパーセントディレクティブは サーバにどの情報をロギングするかを指示します。フォーマット文字列に 文字をそのまま入れることもでき、それらはログの出力に直接コピーされます。 そこに引用文字 (") を書くときは、 フォーマット文字列の最後として解釈 されることを防ぐためにバックスラッシュでエスケープする必要があります。 フォーマット文字列には改行用の "¥n"、タブ用の "¥t" という特別な制御文字も含めることができます。

CustomLog ディレクティブは 既に定義された ニックネーム を使って新しいログファイルを設定します。 アクセスログのファイル名はスラッシュで始まらない限り、 ServerRoot からの相対パスとして 扱われます。

上の設定は Common Log Format (CLF) と呼ばれる形式で ログエントリを書きます。この標準の形式は異なるウェブサーバの多くが 生成することができ、多くのログ解析プログラムが読みこむことができます。 CLF により生成されたログファイルのエントリは以下のようになります:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
```

このログエントリのそれぞれの部分の意味は以下で説明します。

127.0.0.1 (%h)
: これはサーバへリクエストをしたクライアント (リモートホスト) の IP アドレスです。HostnameLookups が On の場合は、サーバはホスト名を調べて、 IP アドレスが書かれているところに記録します。しかし、この設定は サーバをかなり遅くするので、あまりお勧めできません。 そうではなく、logresolve [5] の ようなログの後処理を行なうプログラムでホスト名を調べるのが良いでしょう。 ここに報告される IP アドレスは必ずしもユーザが使っているマシンの ものであるとは限りません。ユーザとサーバの間にプロキシサーバが あれば、このアドレスは元のマシンのものではなく、プロキシの アドレスになります。

- (%l)
: 出力中の「ハイフン」は要求された情報が手に入らなかったということを 意味します。この場合、取得できなかった情報はクライアントのマシンの identd により決まる RFC 1413 のクライアントの アイデンティティです。この情報はあまり信用することができず、 しっかりと管理された内部ネットワークを除いては使うべきではありません。 Apache は IdentityCheck が On になっていない限り、この情報を得ようとすらしません。

frank (%u)
: これは HTTP 認証による、ドキュメントをリクエストした人の ユーザ ID です。CGI スクリプトには通常同じ値が REMOTE_USER 環境変数として与えられます。リクエスト

のステータスコード（以下を参照）が 401 であった場合は、ユーザは認証に失敗しているので、 この値は信用できません。ドキュメントがパスワードで保護されていない場合は、このエントリは前のものと同じように "-" に なります。

[10/Oct/2000:13:55:36 -0700] (%t)
サーバがリクエストの処理を終えた時刻です。書式は:

```
[day/month/year:hour:minute:second zone]
day = 2*digit
month = 3*letter
year = 4*digit
hour = 2*digit
minute = 2*digit
second = 2*digit
zone = (`+' | `-') 4*digit
```

ログのフォーマット文字列に %{format}t を 指定することで、別の形式で時刻を表示させることもできます。 このとき、format は C の標準ライブラリの strftime(3) の形式になります。

"GET /apache_pb.gif HTTP/1.0" (¥"%r¥")
クライアントからのリクエストが二重引用符の中に示されています。 リクエストには多くの有用な情報があります。まず、この場合クライアントが 使ったメソッドは GET です。次に、クライアントは リソース /apache_pb.gif を要求しました。そして、 クライアントはプロトコル HTTP/1.0 を使用しました。 リクエストの各部分を独立にログ収集することもできます。例えば、 フォーマット文字列 "%m %U%q %H" は メソッド、パス、クエリ文字列、プロトコルをログ収集し、 結局 "%r" とまったく同じ出力になります。

200 (%>s)
サーバがクライアントに送り返すステータスコードです。 この情報は、リクエストが成功応答（2 で始まるコード）であったか、 リダイレクション（3 で始まるコード）であったか、クライアントによる エラー（4 で始まるコード）であったか、サーバのエラー（5 で始まるコード）であったか、を現すので、非常に大切です。ステータスコードの 完全なリストは HTTP 規格[6]（RFC2616 第 10 節）にあります。

2326 (%b)
この最後のエントリはクライアントに送信されたオブジェクトの、 応答ヘッダを除いたサイズを現します。コンテントがクライアントに送られなかった 場合は、この値は "-" になります。コンテントが無い場合に "0" をログ収集するには、%b ではなく %B を使ってください。

Combined Log Format

もう一つのよく使われる書式は Combined Log Format と呼ばれています。 以下のようにして使うことができます。

```
LogFormat "%h %l %u %t ¥"%r¥" %>s %b ¥"%{Referer}i¥" ¥"%{User-agent}i¥"" combined
CustomLog log/acces_log combined
```

この書式の最初の方は Common Log Format とまったく同じで、最後に 二つ追加のエントリがあります。追加のエントリはパーセントディレクティブ %{header}i を使っています。ここで header は HTTP のリクエストヘッダのどれかです。この書式による アクセスログは

以下のような感じになります:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326
"http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"
```

追加のエントリは:

"http://www.example.com/start.html" (¥"%{Referer}i¥")
: "Referer" (意図的な綴り間違い) HTTP リクエストヘッダです。 これはクライアントが報告してくる参照元のサイトを表します。 (この場合は、/apache_pb.gif にリンクしているか、 それを含んでいるページです)。

"Mozilla/4.08 [en] (Win98; I ;Nav)" (¥"%{User-agent}i¥")
: User-Agent HTTP リクエストヘッダです。これはクライアントのブラウザが 自分自身のことを報告してくる情報です。

複数のアクセスログ

複数のアクセスログは単に設定ファイルに複数の CustomLog ディレクティブを書くことで作成されます。例えば、以下のディレクティブは 三つのアクセスログを作ります。最初のものは基本的な CLF の情報で、 二つ目と三つ目は referer とブラウザの情報です。最後二つの CustomLog は ReferLog ディレクティブと AgentLog ディレクティブの効果をまねる方法を示しています。

```
LogFormat "%h %l %u %t ¥"%r¥" %>s %b" common
CustomLog logs/access_log common
CustomLog logs/referer_log "%{Referer}i -> %U"
CustomLog logs/agent_log "%{User-agent}i"
```

この例は LogFormat で ニックネームを定義する必要がない、 ということも示しています。ニックネームの代わりに、 CustomLog ディレクティブに 直接ログの書式を指定することができます。

条件付きログ

クライアントのリクエストの特徴に基づいてアクセスログにエントリの 一部をロギングしない方が便利なことがあります。これは 環境変数[7] の補助により簡単に実現できます。まず、 リクエストが何らかの条件に合うということを現すために環境変数が 設定される必要があります。これは通常は SetEnvIf により 行なわれます。そして、CustomLog ディレクティブの env= 節を使って環境変数が設定されているリクエストを 含めたり排除したりすることができます。いくつか例を挙げます:

```
# Mark requests from the loop-back interface
SetEnvIf Remote_Addr "127¥.0¥.0¥.1" dontlog
# Mark requests for the robots.txt file
SetEnvIf Request_URI "^/robots¥.txt$" dontlog
# Log what remains
CustomLog logs/access_log common env=!dontlog
```

他の例として、英語を話す人からのリクエストとそれ以外の人からのリクエストを 分けたい、という場合を考えてみてください。

```
SetEnvIf Accept-Language "en" english
CustomLog logs/english_log common env=english
CustomLog logs/non_english_log common env=!english
```

ここまででは条件付きロギングが非常に強力で柔軟であることを示してきましたが、 それがログの内容を制御する唯一の方法というわけではありません。ログファイルは サーバの活動の完全な記録である方がより役に立ちます。単純にログファイルを 後処理して、考慮したくないログを削除する方が簡単であることがよくあります。

ログの交替

普通の負荷のサーバでさえ、ログファイルに保存される情報の量は 膨大になります。アクセスログのファイルは普通 10,000 リクエスト毎に 1 MB 以上増えます。ですから、既存のログを移動したり、削除したりして、 定期的にログを交替させることが必要になります。これはサーバの実行中には 行なえません。というのは、Apache はファイルが open されている間は ずっと古いログファイルに書き続けるからです。 新しいログファイルを open できるように、ログファイルが移動されたり 削除された後に、サーバを再起動 [8]する 必要があります。

優雅な 再起動を行なうことで、サーバは既存のコネクションや 処理待ちのコネクションを失うことなく新しいログファイルを open させる ことができます。しかし、これを実現するために、サーバは古いリクエストを 扱っている間は古いログファイルに書き続ける必要があります。 ですから、再起動の後ではログファイルの処理を始める前に、しばらく待たなければ なりません。単にログを交替させて、ディスクの節約のために古いログを 圧縮する普通のシナリオは:

```
mv access_log access_log.old
mv error_log error_log.old
apachectl graceful
sleep 600
gzip access_log.old error_log.old
```

ログの交替をするもう一つの方法はパイプ経由のログを使うもので、次の節で説明されています。

パイプ経由のログ

Apache httpd はエラーログとアクセスログをファイルに直接書く代わりに、 パイプを通して別のプログラムに書き出すことができます。 この機能により、主サーバにコードを追加することなく ロギングの柔軟性が非常に高まっています。パイプにログを書くためには、単にファイル名をパイプ文字 "|" に置き換え、その続きに 標準入力からログのエントリを受けとる実行プログラムの名前を書くだけです。 Apache はパイプ経由のログ用のプロセスをサーバの起動時に実行し、 サーバの実行中にそのプログラムがクラッシュしたときはそれを再び 実行します。(この最後の機能がこの技術が「信頼性のあるパイプ経由のロギング」 と呼ばれている理由です。)

パイプ経由のログ用のプロセスは Apache httpd の親プロセスから起動され、 そのプロセスのユーザ ID を継承します。これは、これは、パイプ経由のログ用の プログラムは普通

root として実行されることを意味します。 ですから、プログラムを簡単で安全に保つことが非常に重要です。

パイプ経由のログの重要な利用法は、サーバの再起動なしでログの交替を することです。Apache HTTP サーバにはこのための rotatelogs [9] と呼ばれる簡単な プログラムが付属しています。たとえば、24 時間毎にログを交替させるには、 以下のものを使うことができます:

```
CustomLog "|/usr/local/apache/bin/rotatelogs /var/log/access_log 86400" common
```

パイプの先で呼ばれるコマンド全体が引用符で囲まれていることに注目して ください。この例はアクセスログを使っていますが、エラーログにも同じ技術を 使うことができます。

似ているけれど、よりずっと柔軟な cronolog [10] というログ交替用の プログラムが外部のサイトにあります。

条件付きロギングと同様、パイプ経由のログは非常に強力な 道具ですが、オフラインの後処理のような、より簡単な解決方法があるときは 使わない方が良いでしょう。

バーチャルホスト

多くの バーチャルホスト [11] のあるサーバを実行している ときは、ログファイルの扱い方にいくつかの方法があります。 まず、単独のホストのみのサーバとまったく同じようにログを使うことができます。 ロギングディレクティブを主サーバのコンテキストの <VirtualHost> セクションの外に置くことで、 すべてのログを同じアクセスログとエラーログにログ収集することができます。 この手法では個々のバーチャルホストの統計を簡単にとることはできません。

>CustomLog や ErrorLog ディレクティブが <VirtualHost> の中に 置かれた場合は、そのバーチャル ホストへのすべてのリクエストやエラーがそこで指定されたファイルにのみ ログ収集されます。ロギングディレクティブのないバーチャルホストは 依然としてリクエストが主サーバのログに送られます。この手法は少ない バーチャルホストに対しては非常に有用ですが、ホストの数が非常に多くなると 管理が大変になります。さらに、ファイル記述子の限界 [12]の問題を起こすことが あります。

アクセスログには、非常に良い妥協案があります。バーチャルホストの 情報をログのフォーマット文字列に加えることで、すべてのホストへの リクエストを同じログにログ収集して、後でログを個々のファイルに分割することが できます。たとえば、以下のディレクティブを見てください。

```
LogFormat "%v %l %u %t ¥"%r¥" %>s %b" comonvhost
CustomLog logs/access_log comonvhost
```

%v がリクエストを扱っているバーチャルホストの名前を ログ収集するために使われています。そして、split-logfile [13] のようなプログラムを 使ってアクセスログを後処理することで、 バーチャルホスト毎のファイルにログを分割することができます。

残念ながら、エラーログには同様の手法はありません。ですから、 すべてのバーチャルホストを同じエラーログの中に混ぜるか、 バーチャルホスト毎にエラーログを使うかを選ば

なければなりません。

他のログファイル

関連モジュール	関連ディレクティブ
mod_cgi	PidFile
mod_rewrite	RewriteLog
	RewriteLogLevel
	ScriptLog
	ScriptLogBuffer
	ScriptLogLength

PID ファイル

起動時に、Apache は親 httpd プロセスのプロセス ID を logs/httpd.pid に保存します。この ファイル名は PidFile ディレクティブを使って 変更することができます。プロセス ID は管理者が親プロセスに シグナルを送ることでデーモンを再起動したり終了させたりするときに 使用します。Windows では、代わりに -k コマンドオプションを 使ってください。詳しい情報は 終了と 再起動 [8] のページを見てください。

スクリプトログ

デバッグの補助のために、ScriptLog ディレクティブは CGI スクリプトの入力と出力を記録するようにできます。 これはテスト用にのみ使用して、通常のサーバでは使用しないでください。 詳しい情報は mod_cgi の文書 [14] にあります。

リライトログ

mod_rewrite の強力で 複雑な機能を 使っているときは、ほぼいつもデバッグを簡単にするために RewriteLog の使用が 必要でしょう。このログファイルにはリライトエンジンがリクエストを 書き換える方法の詳細な解析が出力されます。詳しさの度合は RewriteLogLevel で制御できます。

URI References

[1] http://httpd.apache.org/docs-2.1/misc/security_tips.html
[2] http://dmoz.org/Computers/Software/Internet/Site_Management/Log_analysis/
[3] http://dir.yahoo.com/Computers_and_Internet/Software/Internet/World_Wide_Web/Servers/Log_An
[4] http://httpd.apache.org/docs-2.1/mod/mod_log_config.html
[5] http://httpd.apache.org/docs-2.1/programs/logresolve.html
[6] http://www.w3.org/Protocols/rfc2616/rfc2616.txt
[7] http://httpd.apache.org/docs-2.1/env.html
[8] http://httpd.apache.org/docs-2.1/stopping.html
[9] http://httpd.apache.org/docs-2.1/programs/rotatelogs.html
[10] http://www.cronolog.org/
[11] http://httpd.apache.org/docs-2.1/vhosts/
[12] http://httpd.apache.org/docs-2.1/vhosts/fd-limits.html
[13] http://httpd.apache.org/docs-2.1/programs/other.html

[14] http://httpd.apache.org/docs-2.1/mod/mod_cgi.html