HTTP 圧縮の設定

CSS では、クライアントに返す HTTP テキスト データを、SSL 圧縮モジュールを介して圧縮するように設定できます。HTTP 応答データを圧縮することで、ダイアルアップやその他の低速度リンクを使用しているクライアントのパフォーマンスが向上し、サーバ データが消費する帯域を低減できます。CSS に HTTP 応答データを圧縮させることで、サーバにこのタスクを実行する負荷をかけずに済みます。

この章では、CSS サービスで HTTP データ圧縮を設定する方法について説明します。この章の内容は、SSL 機能を備えたすべての CSS モデルに共通です。

この章の主な内容は次のとおりです。

CSS による圧縮の概要

CSS サービスで圧縮を設定すると、CSS は、サーバからの HTTP GET 応答データを圧縮します。CSS は、クライアントからの HTTP 要求やサーバ応答内の HTTP ヘッダーは圧縮しません。データが暗号化されている SSL 接続が終了した場合、CCS は暗号化データを圧縮できません。データを圧縮するには、SSL データを復号化してクリアテキスト データに戻す必要があります。その後で、CSS は、圧縮データを再度暗号化してクライアントに返します。

CSS が データを圧縮できるのは、HTTP Version 1.1 以降の接続です。CSS は、HTTP Version 1.0 の応答は圧縮せず、圧縮していないデータを渡します。

HTTP 1.1 では、さまざまなデータの符号化方式に対応しています。CSS がサポートしている符号化の設定値は、次のとおりです。

- gzip ：RFC1952 で定められたファイル圧縮形式
- deflate ：RFC1951 で定められたデータ圧縮形式
- identity ：圧縮は行わない

クライアントは、HTTP 要求のヘッダーの Accept-Encoding フィールドを通じて、自身の復号化能力をアドバタイズします。CSS は、Accept-Encoding フィールドの圧縮トークンを使用して、応答で使用する圧縮符号化タイプを判別します。Accept-Encoding フィールドには圧縮トークンのリストが格納されており、CSS は、次の優先順位で圧縮を適用します。

1. deflate
2. gzip
3. identity

CSS では、HTTP 要求ヘッダーに Accept-Encoding フィールドがない場合でも、圧縮符号化タイプを設定できます。サーバが応答データとともに GET 応答を返す場合は、その応答内容にコンテンツの圧縮符号化が示されます。

CSS は、特定のファイル拡張子とコンテンツ タイプの圧縮をサポートしています。サポートされているファイル拡張子は次のとおりです。

- .asp
- .aspx
- .css

- .htm
- .html
- .jhtml
- .js
- .jsp
- .php
- .shtml

サポートされているコンテンツ タイプは次のとおりです。

- application/x-javascript
- text/plain
- text/html
- text/css

GET の「/」で示される条件も圧縮対象です。

（注）　ブラウザごとの圧縮形式の互換性については、ブラウザの Web サイトを参照してください。

CSS での圧縮は、SSL Compression（SSL-C; SSL 圧縮）モジュール上で実行されます。SSL-C モジュールのある CSS で圧縮を有効にすると、このモジュールは、クライアントとサーバの接続で TCP エンドポイントとして動作し、サーバから返されたデータを圧縮します。

ここでは、SSL-C モジュールおよび CSS を介した圧縮トラフィックに関する、次のトピックについて説明します。

- SSL 圧縮のハードウェア
- CSS による圧縮トラフィック

SSL 圧縮のハードウェア

圧縮の実行がハードウェアによるかソフトウェアによるかは、SSL モジュールのタイプまたは CSS 11501 プラットフォームによって決まります。SSL-C モジュールでは、ハードウェアを介した圧縮を行います。このハードウェアには、圧縮機能を最適化するための、特別なチップセットが搭載されています。圧縮のほかに SSL の機能をフルに実行でき、圧縮と SSL の同時実行もできます。CSS では、CSS 11503 または CSS 11506 シャーシ内の SSL-C モジュールを使用するか、または CSS 11501 上に統合された SSL 圧縮機能を使用して、ハードウェア圧縮を実行します。これ以降は、SSL-C モジュールとして統合された SSL 圧縮について説明します。

圧縮機能が統合されていない SSL モジュールは、ソフトウェアによる圧縮を実行します。

（注） ハードウェアとソフトウェアによる圧縮機能には、パフォーマンスの点で著しい違いがあります。ハードウェアによる圧縮機能が統合されていない SSL モジュールでは、圧縮しないことを強くお勧めします。このようなモジュールではソフトウェアを介して圧縮を実行しますが、最適なパフォーマンスは得られません。

表 9-1 を参照して、モジュール タイプ別、または CSS 11501 プラットフォーム別に、CSS が実行可能な圧縮方法を確認してください。

表 9-1　CSS での圧縮方式

		CSS プラットフォームおよび SSL モジュール タイプ				
		11501	11501S	11501S-C	11503 または 11506 SSL モジュールを使用	11503 または 11506 SSL-C モジュールを使用
圧縮方式	ソフトウェア	使用不可	可	不可	可	不可
	ハードウェア	使用不可	不可	可	不可	可

CSS SSL 設定の一部として圧縮を行う場合は、SSL の終了サービスまたは開始サービスに圧縮を設定します。このように SSL と圧縮処理を統合して同じフローパスを使用することで、SSL と 圧縮を同じモジュール上で実行することができます。

圧縮のみのサービスの場合、スロット番号を設定することによって、圧縮トラフィックを SSL-C モジュールへダイレクトできます。圧縮トラフィックにスロット番号を設定しない場合、CSS は常に、最も小さいスロット番号で SSL-C モジュールを実行します。CSS に SSL-C が備えられていないが標準の SSL モジュールがある場合も、CSS は、最も小さいスロット番号で標準の SSL モジュールを実行します。

圧縮のみのサービスの場合は、SSL スロットの設定をお勧めします。CSS に複数の SSL-C モジュールがあり、利用可能な最初の SSL-C モジュールへ圧縮トラフィックがすべて流れる場合、コンテンツのロード バランシングが機能するため、このモジュール上とすべての SSL-C モジュール上の SSL スループットが低下するおそれがあります。このような場合は、特定のスロットを設定することで、他の SSL-C モジュールとのリソース競合を緩和することができます。

（注）　SSL 圧縮機能と SSL トラフィックのある CSS 11501 では、圧縮を有効にすると SSL スループットが低下するおそれがあります。

CSS による圧縮トラフィック

CSS は、次の種類の圧縮トラフィックをサポートしています。

- 圧縮のみのサービス
- 圧縮と SSL 終了
- 圧縮と SSL 開始
- 圧縮とバックエンド SSL による SSL 終了

ここでは、次の内容について説明します。

- SSL 終了とバックエンド SSL による HTTP 圧縮

HTTP 圧縮のみのサービス

クライアントとサーバ間のトラフィックを暗号化する必要がない場合、CSS では、SSL-C モジュールを使用して、応答データに対して HTTP 圧縮を実行できます。CSS は、サーバからクリアテキスト データを受信して、圧縮したデータをクライアントへ転送します。この圧縮タイプを設定するには、サービスに対して圧縮を設定し、これにコンテンツ ルールを適用します。

次の手順は、CSS を介した HTTP 圧縮のフローを示しています。

1. クライアントが HTTP GET 要求を CSS に送信します。
2. レイヤ 4 接続の場合、CSS はすべてのクライアント トラフィックを SSL-C モジュールに直接転送します。

   レイヤ 5 接続の場合、CSS は TCP ハンドシェイクを実行します。CSS が GET 要求を受信し、レイヤ 5 のロード バランシングを実行すると、レイヤ 5 ハンドシェイクと HTTP GET 要求は SSL-C モジュールに転送されます。
3. SSL-C モジュールが、クライアント側の TCP 接続を終了します。このモジュールで HTTP 圧縮が実行されます。
4. モジュールが、選択されたサーバへのバックエンド TCP 接続を確立して、GET 要求を転送します。SSL-C モジュールは、これ以降のトラフィックに対し、TCP 終了プロキシとして動作します。
5. サーバが GET 応答を CSS に送信すると、CSS はその応答を SSL-C モジュールにダイレクトします。
6. モジュールがデータを圧縮して、クライアントに返します。

SSL 終了による HTTP 圧縮

SSL-C モジュールは、クライアント側の接続で SSL を終了すると、クライアントからのデータを復号化し、サーバからクライアントへのデータを暗号化します。圧縮を行う場合は、モジュールはサーバからのデータを圧縮してから、データを暗号化してクライアントへ送信します。

この設定の場合、SSL 終了と圧縮のサービスを設定し、このサービスにコンテンツ ルールを適用します。SSL 終了のサービスを設定する方法については、第 4 章「SSL 終了の設定」を参照してください。

次の手順は、CSS を介した SSL 終了と HTTP 圧縮のフローを示しています。

1. クライアントが、暗号化済みの SSL データを CSS に送信します。
2. CSS はそのデータを SSL-C モジュールに転送します。
3. SSL-C モジュールは、クライアント側の接続を終了して、データを復号化してクリアテキスト データに戻します。
4. モジュールが、圧縮のために HTTP ヘッダーを変更します。
5. モジュールがサーバへの TCP 接続を開始して、要求を CSS に送信します。
6. CSS がデータを受信すると、サーバへのフローをセットアップして、GET 要求を配信します。
7. サーバが応答を返すと、CSS は圧縮を実行するため、トラフィックを SSL-C モジュールへダイレクトします。
8. モジュールがデータを圧縮してから、これを暗号化します。
9. モジュールが、暗号化されたデータを CSS へ送信します。
10. CSS が、暗号化された圧縮データをクライアントへ返します。

SSL 開始の HTTP 圧縮

SSL-C モジュールは、SSL を開始すると、クライアントからのデータを暗号化し、サーバからクライアントへのデータを復号化します。圧縮を行う場合は、モジュールはサーバからのデータを復号化した後に圧縮し、そのデータをクライアントへ送信します。

この設定の場合、同じサービスに SSL 開始と圧縮のサービスを設定し、そのサービスにコンテンツ ルールを適用します。SSL 開始のサービスを設定する方法については、第 6 章「SSL 開始の設定」を参照してください。

次の手順は、CSS を介した SSL 開始と HTTP 圧縮のフローを示しています。

1. クライアントが、クリアテキスト データを CSS に送信します。
2. CSS は そのデータを SSL-C モジュールに転送します。
3. SSL-C モジュールは、クライアント側の接続を終了し、圧縮のために HTTP ヘッダーを変更します。
4. モジュールが、サーバへのバックエンド SSL 接続を開始し、要求を暗号化します。
5. モジュールがサーバへの TCP 接続を開始して、要求を CSS に転送します。
6. CSS はデータを受信し、サーバへのフローをセットアップして、GET 要求を配信します。
7. 暗号化された応答をサーバが返すと、CSS は圧縮を実行するため、このトラフィックを SSL-C モジュールへダイレクトします。
8. モジュールは、データを復号化してクリアテキスト データに戻してから、これを圧縮します。
9. モジュールは、圧縮したデータを CSS へ送信します。
10. CSS が、圧縮済みのクリアテキスト データをクライアントへ返します。

SSL 終了とバックエンド SSL による HTTP 圧縮

SSL-C モジュールがバックエンド SSL で SSL を終了すると、クライアントからのデータを復号化した後で、そのデータを再び暗号化してからサーバへ送信します。サーバが暗号化されたデータを返すと、モジュールはこのデータを復号化し、再び暗号化してからクライアントへ送信します。圧縮を行う場合は、モジュールはサーバからの復号化済みデータを圧縮してから、再びデータを暗号化してクライアントへ送信します。

次の手順は、CSS を介した SSL 終了、バックエンド SSL および HTTP 圧縮のフローを示しています。

1. クライアントが、暗号化された SSL データを CSS に送信します。
2. CSS はそのデータを SSL-C モジュールに送信します。
3. SSL-C モジュールは、クライアント側の接続を終了して、データを復号化してクリアテキスト データに戻します。
4. モジュールが、圧縮のために HTTP ヘッダーを変更します。
5. モジュールが、サーバへのバックエンド SSL 接続を開始し、要求を暗号化します。
6. モジュールがサーバへの TCP 接続を開始して、要求を CSS に転送します。
7. CSS はデータを受信し、サーバへのフローをセットアップして、GET 要求を配信します。
8. 暗号化された応答をサーバが返すと、CSS は圧縮を実行するため、このトラフィックを SSL-C モジュールへダイレクトします。
9. モジュールがデータを復号化してクリアテキスト データに戻します。
10. モジュールがデータを圧縮してから、これを暗号化します。
11. モジュールが、暗号化した圧縮データを CSS へ送信します。
12. CSS が、暗号化された圧縮データをクライアントへ返します。

圧縮のみのサービスのクイック スタート

表 9-2 に、圧縮のみのサービスおよび関連のコンテンツ ルールを設定するために必要な手順を簡略に示します。それぞれの手順に、作業を行うために必要な CLI コマンドも示します。CLI コマンドに関する各機能とすべてのオプションの詳細については、表 9-2 の後に続く各項を参照してください。

SSL 終了または SSL 開始の圧縮サービス設定およびコンテンツ ルールに関する詳細は、第 2 章「SSL 設定のクイック スタート」を参照してください。

表 9-2　圧縮のみのサービスのクイック スタート

作業とコマンドの例

1. グローバル設定モードに入ります。
`# config`
`(config)#`

2. 圧縮を設定したいサービスの、サービス設定モードを入力します。 `(config)# service server3`
3. このサービスの IP アドレスを割り当てます。 `(config)# ip address 13.1.1.8`
4. このサービスの キープアライブ タイプを設定します。デフォルトでは、このサービスのキープアライブ タイプは ICMP です。キープアライブ タイプを設定しない場合は、次のように入力します。 `(config-service[server3])# keepalive type none`
5. このサービスでの圧縮を有効にします。 `(config-service[server3])# compress enable`
6. （推奨）デフォルトでは、圧縮は、最も小さいスロット番号で　SSL-C　モジュールを使用して実行されます。スロット 3 で圧縮を実行するように設定するには、次のように入力します。 `(config-service[server3])# slot 3` 詳細については、「圧縮のみのサービスへの SSL スロットの設定」を参照してください。

表 9-2　圧縮のみのサービスのクイック スタート（続き）

作業とコマンドの例
7. サービスでの圧縮を設定したら、サービスをアクティブにします。 `(config-service[server3])# active` **（注）** アクティブなサービスの設定を変更する場合は、先にそのサービスを一時停止する必要があります。

8. サービスのコンテンツ ルールを設定し、ルールをアクティブにします。
`(config)# owner customer`
`(config-owner[customer])# content gzip`
`(config-owner-content[gzip])# vip address 192.168.77.40`
`(config-owner-content[gzip])# protocol tcp`
`(config-owner-content[gzip])# port 80`
`(config-owner-content[gzip])# url "/*"`
`(config-owner-content[gzip])# add service server3`
`(config-owner-content[gzip])# active`

次の実行設定例は、表 9-2 のコマンドの入力結果を表しています。

```
!************************* SERVICE **************************
service server3
  ip address 13.1.1.8
  keepalive type none
  compress enable
  slot 3
active

!************************* OWNER ****************************
content gzip
    vip address 192.168.77.40
    protocol tcp
    port 80
    url "/*"
    add service server3
active
```

CSS での 圧縮設定

ここでは、HTTP 圧縮の設定に関する次の内容について説明します。

- サービスでの圧縮の有効化
- サービスでの圧縮の無効化
- 圧縮のみのサービスへの SSL スロットの設定
- 圧縮の優先アルゴリズムの設定
- Accept-Encode フィールドが省略された場合の圧縮符号化タイプの設定
- 圧縮データ タイプの設定

サービスでの圧縮設定が終了したら、サービスをアクティブにします。アクティブなサービスを変更する必要がある場合、先にそのサービスを一時停止してから変更を行います。

（注） サービスに Adaptive Session Redundancy（ASR; 適応型セッションの冗長性）が設定されている場合は、圧縮を設定できません。CSS では、圧縮が有効なサービスを持つコンテンツ ルールに冗長インデックスを設定したり、冗長インデックスが設定されたルール内のサービスで圧縮を有効にしたりすることはできません。アクティブなサービスの設定を変更する場合は、先にそのサービスを一時停止する必要があります。

サービスでの圧縮の有効化

デフォルトでは、サービスでの圧縮は無効です。クライアントへの HTTP 応答データを圧縮できるようにするには、サービスでの圧縮を有効にする必要があります。サービスでの圧縮を設定するには、サービス設定モードで **compress enable** コマンドを使用します。このコマンドのシンタックスは次のとおりです。

compress enable

たとえば、次のように入力します。

```
(config-service[server3])# compress enable
```

サービスでの圧縮の無効化

サービスをリセットしてデフォルトの状態に戻すことで、HTTP 応答データを圧縮しないようにできます。サービスでの圧縮を無効にしても、そのサービスの圧縮設定が削除されるわけではありません。

圧縮を無効にするには、**compress disable** コマンドを使用します。サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

たとえば、次のように入力します。

```
(config-service[server3])# compress disable
```

圧縮のみのサービスへの SSL スロットの設定

CSS 11503 および 11506 では、SSL-C モジュールがクライアントへの HTTP 応答データを圧縮します。HTTP 圧縮が SSL 終了または SSL 開始のサービス設定の一部である場合、圧縮トラフィックは、SSL サービスに設定されたスロット番号を使用します。

圧縮だけのサービスの場合、スロット番号を設定することによって、SSL-C モジュールへ圧縮トラフィックをダイレクトすることもできます（オプション）。圧縮トラフィックの場合、SSL-C スロット番号を設定することをお勧めします。圧縮にスロット番号を設定しない場合、CSS は常に、最も小さいスロット番号で SSL-C モジュールを使用します。CSS に複数の SSL モジュールがあり、利用可能な最初の SSL モジュールへ圧縮トラフィックがすべて流れる場合、コンテンツのロード バランシングが機能するため、このモジュール上とすべての SSL モジュール上の SSL スループットが低下するおそれがあります。このような場合は、特定のスロットを設定することで、他の SSL モジュールとのリソース競合を緩和することができます。

圧縮を実行するスロットを設定するには、サービス設定モードで **slot** コマンドを使用します。このコマンドのシンタックスは次のとおりです。

slot *number*

number 変数は、SSL-C モジュールのスロット番号です。

たとえば、スロット 3 で SSL-C モジュールのサービスを設定するには、次のように入力します。

```
(config-service[server3])# slot 3
```

（注）　サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

圧縮の優先アルゴリズムの設定

CSS が圧縮サービスの HTTP 要求を受信すると、ヘッダーを処理することで使用する圧縮タイプを判別します。HTTP ヘッダーの Accept-Encode フィールドには、圧縮符号化タイプの情報が含まれています。デフォルトでは、CSS　は、Accept-Encoding フィールドでアドバタイズされたとおりの圧縮アルゴリズムを使用します。ただし、HTTP 要求に Accept-Encoding フィールドが含まれていない場合は、CSS は **compress accept-omit** コマンドで設定された圧縮符号化タイプを使用します。このコマンドの詳細については、「Accept-Encode フィールドが省略された場合の圧縮符号化タイプの設定」を参照してください。

Accept-Encode フィールドまたは **compress accept-omit** コマンドで指定される圧縮符号化タイプを優先するように CSS を設定するには、**compress encode** コマンドを使用します。CSS は、自身の優先設定と Accept-Encode フィールドまたは **compress accept-omit** コマンドのエントリと比較して、そのフローの圧縮方法、またはそのフローの圧縮処理を無視するかどうかを判別します。

表 9-3 は、Accept-Encoding フィールド、および **compress encode** コマンドによる圧縮符号化の優先設定に基づいて、CSS が使用する圧縮符号化タイプが決まる様子を示しています。「無視」は、CSS がフローを圧縮しないことを意味します。たとえば、Accept-Encoding フィールドに圧縮を行わないことを示す identity エントリがある場合、CSS は　**compress encode** コマンドの設定に関係なく、そのフローを圧縮しません。また、**force-gzip** および **force-deflate** の設定は、Accept-Encode フィールドのエンティティよりも優先されます（identity を除く）。

表 9-3　Accept-Encode フィールドおよび compress encode コマンドに基づく圧縮タイプ

		compress encode コマンドの設定				
		gzip	**deflate**	**auto**（デフォルト）	**force-gzip**	**force-deflate**
Accept-Encode フィールドの圧縮タイプ	**identity**	無視	無視	無視	無視	無視
	deflate	無視	deflate	deflate	gzip	deflate
	gzip	gzip	無視	gzip	gzip	deflate

表 9-4 は、**compress accept-omit** コマンド、および **compress encode** コマンドで指定した圧縮符号化の優先設定に基づいて、CSS が使用する圧縮符号化タイプが決まる様子を示しています。たとえば、**compress accept-omit** コマンドのデフォルト設定は identity であり、これは圧縮を無視します。**compress encode** コマンドが **auto**、**gzip**、または **deflate** に設定されていても、CSS は圧縮を実行しません。ただし、**force-gzip** または **force-deflate** に設定されている場合、圧縮は実行されます。

表 9-4　compress accept-omit コマンドおよび compress encode コマンドに基づく圧縮タイプ

		compress encode コマンドの設定				
		gzip	**deflate**	**auto**（デフォルト）	**force-gzip**	**force-deflate**
compress accept-omit コマンドの設定	**identity**（デフォルト）	無視	無視	無視	gzip	deflate
	deflate	無視	deflate	deflate	gzip	deflate
	gzip	gzip	無視	gzip	gzip	deflate

（注）　CSS は、HTTP GET 応答のみを圧縮します。それ以外のフローに対しては、CSS は圧縮を実行しないものとしてマークを付けます。

compress encode コマンドのシンタックスは次のとおりです。

compress encode auto|deflate|gzip|force-deflate|force-gzip

このコマンドのキーワードの意味は次のとおりです。

- **auto** ：（デフォルト）CSS は、クライアントからの HTTP 要求に含まれる Accept-Encoding フィールドの圧縮アルゴリズム トークンを使用します。Accept-Encoding フィールドのトークン リストをもとに、CSS は、次の優先順位で圧縮アルゴリズムを適用します。
  1. deflate
  2. gzip
  3. identity

  このフィールドに圧縮タイプが含まれていないかまたは存在しない場合は、CSS は **compress accept-omit** コマンドの設定を使用します。
- **deflate** ：CSS は、圧縮処理に deflate アルゴリズムを優先し、identity または gzip の圧縮符号化タイプではフローの圧縮を行いません。
- **gzip** ：CSS は、圧縮処理に gzip アルゴリズムを優先し、identity または deflate の圧縮符号化タイプではフローの圧縮を行いません。

- **force-deflate** ：CSS は常に deflate アルゴリズムを使用して符号化します。ただし、Accept-Encoding フィールドに identity エントリが含まれている場合を除きます。
- **force-gzip** ：CSS は常に gzip アルゴリズムを使用して符号化します。ただし、Accept-Encoding フィールドに identity エントリが含まれている場合を除きます。

（注）　**force-deflate** または **force-gzip** のキーワードがクライアントの機能に対応していない場合、クライアントが応答を処理できないため、トランザクションは失敗するおそれがあります。

たとえば、CSS が deflate アルゴリズムを優先するように設定するには、次のように入力します。

```
(config-service[server3])# compress encode deflate
```

優先する符号化方式をデフォルト値の auto に戻すには次のように入力します。

```
(config-service[server3])# compress encode auto
```

（注）　サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

Accept-Encode フィールドが省略された場合の圧縮符号化タイプの設定

HTTP ヘッダーに Accept-Encode フィールドが含まれていない場合、デフォルトでは、CSS はフローの圧縮を実行しません。Accept-Encoding フィールドが省略された HTTP ヘッダーを CSS が受信した場合は、圧縮符号化タイプを設定できます。CSS は、このタイプと **compress encode** コマンドの圧縮符号化設定を比較します。表 9-4 を参照して、**compress accept-omit** コマンド、および **compress encode** コマンドで指定した圧縮符号化の優先設定に基づいて CSS が使用する圧縮符号化タイプを確認してください。

Accept-Encoding フィールドを含まない HTTP 要求の圧縮復号化タイプを指定するには、**compress accept-omit** コマンドを使用します。このコマンドのシンタックスは次のとおりです。

compress accept-omit deflate|gzip|identity

キーワードの意味は次のとおりです。

- **deflate** ：deflate 圧縮符号化タイプ
- **gzip** ：gzip 圧縮符号化タイプ
- **identity** ：（デフォルト）フローを圧縮しない、クリアテキストの identity 符号化タイプ

（注） 設定した圧縮符号化タイプをクライアントが受け入れられるかどうかを確認してください。受け入れられない場合、トランザクションの失敗など、予期せぬ結果が発生するおそれがあります。

たとえば、deflate 符号化タイプを設定するには次のように入力します。

```
(config-service[server3])# compress accept-omit deflate
```

符号化タイプをデフォルト設定の identity に戻すには、次のように入力します。

```
(config-service[server3])# compress accept-omit identity
```

（注） サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

圧縮データ タイプの設定

デフォルトでは、CSS は、ほとんどがテキストで構成されるトラフィックに ASCII 英数字の最適化済みハフマン符号を使用します。ほとんどが数字であるようなその他のトラフィック タイプ、または混在するトラフィックの圧縮を最適化するには、ハフマン符号タイプを指定します。**compress type** コマンドを使用してください。このコマンドのシンタックスは次のとおりです。

compress type ascii|default|numeric

キーワードの意味は次のとおりです。

- **ascii**：（デフォルト）ほとんどがテキストのトラフィック（例：HTML、XML、TXT）で使用する、ASCII 英数字の最適化済みハフマン符号
- **default** ：混在するトラフィック用の deflate アルゴリズムによって定義されるデフォルトのハフマン符号
- **numeric** ：ほとんどが ASCII 数字のトラフィック（例：0 ～ 9）で使用する、ASCII 数字の最適化済みハフマン符号

たとえば、トラフィックとして、ASCII 数字の最適化済みハフマン符号を設定するには、次のように入力します。

```
(config-service[server3])# compress type numeric
```

ハフマン符号をデフォルト設定の ascii に戻すには、次のように入力します。

```
(config-service[server3])# compress type ascii
```

（注） サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

圧縮のみのサービスに対する TCP オプションの設定

圧縮のみのサービスに対し、クライアントまたはサーバ間の TCP 接続、および SSL-C モジュールを設定できます。

（注）　**compress tcp** コマンドは、ssl-accel、backend または init の SSL サービス タイプには設定しないでください。この場合、CSS はエラーを生成します。

TCP 接続を設定するには、サービス設定モードで **compress tcp** コマンドを使用します。ここで説明する内容は、次のとおりです。

- クライアントの TCP 接続タイムアウト値の設定
- サーバの TCP 接続タイムアウト値の設定
- TCP 接続における確認応答遅延の設定
- TCP 接続における Nagle アルゴリズムの設定
- TCP 接続における TCP バッファリングの設定
- TCP 再送信タイマーの設定

クライアントの TCP 接続タイムアウト値の設定

CSS とクライアント間の TCP 接続は、指定した時間が経過すると終了します。この TCP タイムアウト機能を使用すると、CSS SSL モジュールとクライアント間の TCP 接続を、より柔軟に管理できます。クライアントとの TCP 接続の終了を設定する方法については、次の項を参照してください。

- クライアントの TCP SYN タイムアウト値の設定
- クライアントの TCP 無活動タイムアウト値の設定

クライアントの TCP SYN タイムアウト値の設定

CSS の SYN タイマーは、TCP 3 ウェイ ハンドシェイクを終了する手段として、CSS による SYN/ACK の送信とクライアントによる ACK の応答の間の時間差をカウントします。このタイムアウト値は、クライアントと CSS モジュール間の TCP 接続で TCP 3 ウェイ ハンドシェイクが正常に完了しなかったときに、その接続をデータ転送前に終了させるために使用します。この値は **compress tcp virtual syn-timeout** *seconds* コマンドをサービス設定モードで使用して指定します。

TCP SYN の無活動タイムアウト値（秒単位）には、1 ～ 3600 （1 時間）を入力します。デフォルトでは 30 秒に設定されています。

（注） 接続タイマーは、新しい TCP 接続についても、常に再送信終了時間より短く設定する必要があります。

たとえば、TCP SYN のタイムアウトを 30 分（1800 秒）に設定するには、次のコマンドを入力します。

```
(config-service[server3])# compress tcp virtual syn-timeout 1800
```

TCP SYN タイムアウトの値をデフォルトの 30 秒に戻すには、次のコマンドを入力します。

```
(config-service[server3])# no compress tcp virtual syn-timeout
```

（注） サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

クライアントの TCP 無活動タイムアウト値の設定

TCP 3 ウェイ ハンドシェイクを終了するための TCP 無活動タイムアウトのカウントは、ACK を CSS がクライアントから受信したときに開始されます。この無活動タイマーは、そのトラフィック フローの SYN タイマーが停止した時点で再開されます。**compress tcp virtual inactivity-timeout** *seconds* コマンドをサービス設定モードで使用することで、クライアントとの間でほとんどあるいはまったく活動していない TCP 接続を終了するために使用するタイムアウト値を指定します。

TCP 無活動タイムアウト値（秒単位）には、0 （TCP 無活動タイムアウトは無効）～ 3600 （1 時間）の値を入力します。デフォルトでは 240 秒に設定されています。

たとえば、TCP 無活動タイマーを 30 分（1800 秒）に設定するには、次のコマンドを入力します。

```
(config-service[server3])# compress tcp virtual inactivity-timeout
1800
```

TCP 無活動タイマーの値をデフォルトの 240 秒に戻すには、次のコマンドを入力します。

```
(config-service[server3])# no compress tcp virtual inactivity-timeout
```

（注） サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

サーバの TCP 接続タイムアウト値の設定

CSS とサーバ間の TCP 接続は、指定した時間が経過すると終了します。この TCP タイムアウト機能を使用すると、CSS SSL モジュールとサーバ間の TCP 接続を、より柔軟に制御できます。

サーバとの TCP 接続を終了する方法については、次の項を参照してください。

- サーバの TCP SYN タイムアウト値の設定
- サーバの TCP 無活動タイムアウト値の設定

サーバの TCP SYN タイムアウト値の設定

TCP SYN タイマーは、CSS が SYN を送信してバックエンドの TCP 接続を開始したタイミングと、サーバが SYN/ACK で応答したタイミングの時間差をカウントします。このタイムアウト値は、サーバとの TCP 接続で TCP 3 ウェイ ハンドシェイクが正常に完了しなかったときに、その接続をデータ転送前に終了させるために使用し、**compress tcp server syn-timeout** *seconds* コマンドをサービス設定モードで使用して指定します。

TCP SYN のタイムアウト値（秒単位）には、1 ～ 3600（1 時間）の値を入力します。デフォルトでは 30 秒に設定されています。

（注） 接続タイマーは、新しい TCP 接続についても、常に再送信終了時間より短く設定する必要があります。

たとえば、TCP SYN のタイムアウトを 30 分（1800 秒）に設定するには、次のコマンドを入力します。

```
(config-service[server3])# compress tcp server syn-timeout 1800
```

TCP SYN タイムアウトの値をデフォルトの 30 秒に戻すには、次のコマンドを入力します。

```
(config-service[server3])# no compress tcp server syn-timeout
```

（注） サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

サーバの TCP 無活動タイムアウト値の設定

TCP 無活動タイムアウトのカウントは、CSS がサーバから SYN/ACK を受信した時点から開始されます。この無活動タイマーは、そのトラフィック フローの SYN タイマーが停止した時点で再開されます。サーバとの TCP 接続がほとんど、またはまったく活動していないときに、その接続を終了させるために使用するこのタイムアウト値は、**compress tcp server inactivity-timeout** *seconds* コマンドをサービス設定モードで使用して指定します。

TCP 無活動タイムアウト値（秒単位）には、0 （TCP 無活動タイムアウトは無効）～ 3600 （1 時間）の値を入力します。デフォルトでは 240 秒に設定されています。

たとえば、TCP 無活動タイマーを 30 分（1800 秒）に設定するには、次のコマンドを入力します。

```
(config-service[server3])# compress tcp server inactivity-timeout 1800
```

TCP 無活動タイマーの値をデフォルトの 240 秒に戻すには、次のコマンドを入力します。

```
(config-service[server3])# no compress tcp server inactivity-timeout
```

（注） サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

TCP 接続における確認応答遅延の設定

クライアントまたはサーバ接続におけるデフォルトの確認応答遅延時間は 200 ミリ秒（Ms）です。確認応答遅延の TCP タイマーの長さを無効にしたり調整するには、**compress tcp virtual|server ack-delay** コマンドをサービス設定モードで使用します。

compress tcp virtual|server ack-delay *value*

value 変数は、確認応答遅延のタイマーの長さをミリ秒（Ms）で指定します。デフォルト値は 200 です。0 ～ 10000 の値を入力します。0 を指定すると、クライアントまたはサーバからトラフィックを受信する際の確認応答遅延は無効になります。

たとえば、クライアントの TCP 接続の確認応答遅延を 400 ミリ秒に設定するには、次のように入力します。

```
(config-service[server3])# compress tcp virtual ack-delay 400
```

たとえば、サーバの TCP 接続の確認応答遅延を 400 ミリ秒に設定するには、次のように入力します。

```
(config-service[server3])# compress tcp server ack-delay 400
```

クライアントの TCP 接続の確認応答遅延をデフォルトの 200 ミリ秒に戻すには、次のように入力します。

```
(config-service[server3])# no compress tcp virtual ack-delay
```

サーバの TCP 接続の確認応答遅延をデフォルトの 200 ミリ秒に戻すには、次のように入力します。

```
(config-service[server3])# no compress tcp server ack-delay
```

（注） サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

TCP 接続における Nagle アルゴリズムの設定

TCP Nagle アルゴリズムは、クライアント / サーバと SSL-C モジュールの間の TCP 接続で転送されるサイズの小さい多数のバッファ メッセージを自動的に連結します。この処理で個々の TCP 接続で送信されるパケット数が減少し、CSS のスループットが向上します。ただし、Nagle アルゴリズムと TCP 遅延応答確認の相互作用によって、TCP 接続の遅延時間が長くなることもあります。TCP 接続で許容範囲を超える遅延が発生するようであれば、Nagle アルゴリズムを無効にしてください。

クライアントまたはサーバの TCP 接続に対し、Nagle アルゴリズムを無効にしたり、再び有効にするには、**compress tcp virtual|server nagle** コマンドをサービス設定モードで使用します。このコマンドのシンタックスは次のとおりです。

compress tcp virtual|server nagle enable|disable

キーワードの意味は次のとおりです。

- **enable** ：Nagle アルゴリズムを再び有効にします。デフォルトでは、Nagle アルゴリズムは TCP の各接続ごとに有効になっています。
- **disable** ：TCP 接続の遅延を監視する場合に、Nagle アルゴリズムを無効にします。

たとえば、クライアントの TCP 接続の Nagle アルゴリズムを無効にするには、次のように入力します。

```
(config-service[server3])# compress tcp virtual nagle disable
```

サーバの TCP 接続の Nagle アルゴリズムを無効にするには、次のように入力します。

```
(config-service[server3])# compress tcp server nagle disable
```

クライアントの TCP 接続の Nagle アルゴリズムを再び有効にするには、次のように入力します。

```
(config-service[server3])# compress tcp virtual nagle enable
```

サーバの TCP 接続の Nagle アルゴリズムを再び有効にするには、次のように入力します。

```
(config-service[server3])# compress tcp server nagle enable
```

（注）　サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

TCP 接続における TCP バッファリングの設定

ネットワークの速度が遅くて輻輳が発生している場合、特定の TCP 接続に対して、TCP ウィンドウがシャットダウンされて 0 に設定されるまでにバッファリングするデータ量（バッファ サイズ）を増やすことができます。バッファ サイズを増やすと、クライアントへの遅い接続の待機時間は減りますが、CSS でのバッファリング時間は増加します。速度の遅いクライアント接続が多数存在すると CSS のメモリが不足する可能性があるため、この機能を使用する場合は注意が必要です。

指定した TCP 接続で、クライアントまたはサーバから TCP バッファリングを設定するには、**compress tcp buffer-share** コマンドをサービス設定モードで使用します。このコマンドのシンタックスは次のとおりです。

compress tcp buffer-share [**rx** *bytes*|**tx** *bytes*]

このコマンドのキーワードと変数は次のとおりです。

- **rx** *bytes* ：指定した接続でクライアント トラフィックをバッファリングするためのバッファ データ量をバイトで指定します。16400 ～ 262144 の範囲で数字を入力します。デフォルトのバッファ サイズは 32768 です。
- **tx** *bytes* ：指定した接続でサーバ トラフィックをバッファリングするためのバッファ データ量をバイトで指定します。16400 ～ 262144 の範囲で数字を入力します。デフォルトのバッファ サイズは 65536 です。

たとえば、クライアント トラフィックのバッファ サイズを 65536 バイトに設定するには、次のように入力します。

```
(config-service[server3])# compress tcp buffer-share rx 65536
```

クライアント トラフィックのバッファ サイズをデフォルトの 32768 にリセットするには、次のコマンドを入力します。

```
(config-service[server3])# no compress tcp buffer-share rx
```

サーバ トラフィックのバッファ サイズを 131072 バイトに設定するには、次のように入力します。

```
(config-service[server3])# compress tcp buffer-share tx 131072
```

サーバ トラフィックのバッファ サイズをデフォルトの 65536 にリセットするには、次のように入力します。

```
(config-service[server3])# no compress tcp buffer-share tx
```

（注） サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

TCP 再送信タイマーの設定

ネットワーク上で多数のパケット損失が起こる場合は、TCP トランザクションに時間がかかるおそれがあります。TCP トランザクションの再送信タイマーを調整するには、**compress tcp virtual|server retrans** コマンドを使用します。このコマンドのシンタックスは次のとおりです。

compress tcp virtual|server retrans *milliseconds*

milliseconds 変数は、TCP トランザクションの再送信時間の最小値を指定します（ミリ秒）。50 ～ 500 の数値を入力します。デフォルト値は 500 です。

たとえば、クライアント トランザクションの再送信タイマーを 100 ミリ秒に設定するには、次のように入力します。

```
(config-service[server3])# compress tcp virtual retrans 100
```

サーバ トランザクションの再送信タイマーを 100 ミリ秒に設定するには、次のように入力します。

```
(config-service[server3])# compress tcp server retrans 100
```

クライアント トランザクションの再送信タイマーをデフォルトの 500 ミリ秒に戻すには、次のように入力します。

```
(config-service[server3])# no compress tcp virtual retrans
```

サーバ トランザクションの再送信タイマーをデフォルトの 500 ミリ秒に戻すには、次のように入力します。

```
(config-service[server3])# no compress tcp server retrans
```

（注） サービスがアクティブである場合、先にサービスを一時停止してからこの設定を変更する必要があります。

圧縮情報の表示

CSS では、圧縮に関する統計情報、および圧縮コプロセッサの情報を表示できます。圧縮に関する統計情報を表示にするには、**show service** コマンドを使用します。

表 9-5 に、**show service** コマンドの圧縮関連のフィールドと、その説明を示します。これらのフィールドにおける CSS の圧縮関連情報をゼロに設定するには、**zero service compression** コマンドを使用します。

表 9-5　show service コマンドの出力における圧縮関連情報のフィールド

フィールド	説明
HTTP Responses In	SSL-C モジュールへ転送された、未圧縮の HTTP 応答の総数
Compressed HTTP Responses Out	SSL-C モジュールを介して渡された、圧縮済み HTTP 応答の総数
Bypassed HTTP Responses Out	HTTP ヘッダーによって圧縮が許可されなかった応答の総数
HTTP Response Bytes In	SSL-C モジュールへ転送された、未圧縮の HTTP 応答の総バイト数
Compressed HTTP Response Bytes Out	SSL-C モジュールを介して渡された､圧縮済みの HTTP 応答の総バイト数
Bypassed HTTP Response Bytes Out	HTTP ヘッダーによって圧縮が許可されなかった応答の総バイト数
Compression Ratio Percentage	圧縮によってサイズが縮小された HTTP 応答の圧縮率をパーセントで示します（Compressed HTTP Response Bytes Out 値を HTTP Response Bytes In 値で割った値）。この値が小さいと、その応答の圧縮率があまり高くないことを示します。この値が大きいと、その応答の圧縮率が非常に高いことを示します。

CSS のハードウェア圧縮においてコプロセッサが利用可能かどうかを表示するには、**show chassis coprocessors** コマンドを使用します。このコマンドにより、コプロセッサ インベントリの各フィールドに圧縮ハードウェアの情報が表示されます。また、コプロセッサ インベントリのフィールドは、**show chassis verbose** コマンドでも表示できます。

表 9-6 に、**show chassis coprocessor** コマンドで表示されるコプロセッサ インベントリの各フィールドと、その説明を示します。

表 9-6　show chassis coprocessor コマンドで表示されるコプロセッサ インベントリの各フィールド

フィールド	説明
Slot/Subslot	圧縮コプロセッサが存在するスロットとサブスロット
Coprocessor	プロセッサのタイプで、「COMPRESSION」のように示される。
Version	CSS がフラッシュ デバイスからコプロセッサにロードしたソフトウェア バージョン
Active	CSS がコプロセッサにロードしたイメージが存在する、フラッシュ デバイスの状態。状態には、LOCKED、OPERATIONAL があり、CSS がイメージをロードしたときにエラーが発生すると UNKNOWN になります。