CHAPTER 13

キャッシングの設定

この章では、CSS のキャッシング機能の概要と、運用のための設定方法を説明します。この章の内容は、特に指定のない限り、すべての CSS モデルに共通です。

この章の内容は次のとおりです。

キャッシングの概要

インターネット上での情報に対する需要が高まると、輻輳が生じたり、情報の取得に遅延が生じたりします。同じ情報が何度も取得される場合が多いので、それらの情報を保存しておくと、一連の要求により効率的で、しかも狭い帯域幅で応えることができます。

情報をローカルに保存して格納することをキャッシングと呼びます。Web キャッシングを利用すると、前に要求されたコンテンツのコピーがトポロジ的にクライアントに近い場所にあるキャッシュ サーバに一時保存されます。保存されたコンテンツは、同じコンテンツに対する一連のクライアント要求で再び利用されます。

ローカルにコンテンツを保存すると、次の効果を得られます。

- ネットワーク リソースの最適化
- ネットワーク帯域幅の節約
- インターネットでの輻輳の軽減
- ネットワークの応答時間および全体的なサービスの質の向上

コンテンツ キャッシング

ネットワークにコンテンツ キャッシングを配置すれば、Web キャッシングのコスト効率と信頼性を高めることができます。キャッシュ サーバを使用するようにコンテンツ ルールを作成することにより、CSS はキャッシュのフロントエンド装置として機能し、次の処理を行います。

- Web コンテンツ要求に応じてネットワーク トラフィックを検査する。
- キャッシュに保存できないコンテンツのキャッシングを自動的にバイパスする。
- コンテンツ要求を分散送信して、サービスのキャッシュ　ヒット率を最大化する。
- キャッシュ サービスに障害が起きた場合にキャッシュをバイパスさせたり、残りの各キャッシュ サービスにコンテンツ要求を分散送信する。

クライアントからコンテンツ要求があると、CSS は次の処理を行います。

- コンテンツ要求の代行受信

- コンテンツ インテリジェンスの適用（HTTP 要求ヘッダーを解析して、コンテンツ要求をキャッシュ サーバに配分する）

続いて、CSS は次のいずれかの処理を実行します。

- コンテンツ ルールで指定したロード バランシング方式（宛先 IP アドレスなど）に基づき、適切なキャッシュに要求を送信する。
- キャッシュできないコンテンツの場合、キャッシュ サーバをバイパスして、要求を本来の宛先サーバに転送する。

CSS からキャッシュ サーバに要求が送信されると、キャッシュ サーバは要求されたコンテンツを戻す（ローカルにコピーが存在する場合）か、そのコンテンツを提供している本来の宛先サーバに CSS 経由で新たなコンテンツへの要求を送信します。キャッシュ サーバは、新しいコンテンツの要求を送信して宛先サーバから応答を受信すると、クライアントに応答を戻します。そのコンテンツがキャッシュに保存可能な場合は、今後の要求に使用するためにキャッシュ サーバにコンテンツのコピーが保存されます。

要求されたコンテンツがローカルのキャッシュ サーバにある場合、その要求をキャッシュ ヒットと呼びます。要求されたコンテンツがローカルに存在しないときには、キャッシュ サーバからコンテンツに新しい要求が送信されます。このような要求はキャッシュ ミスと呼ばれます。

ここでは、次の CSS コンテンツ キャッシングの例を示します。

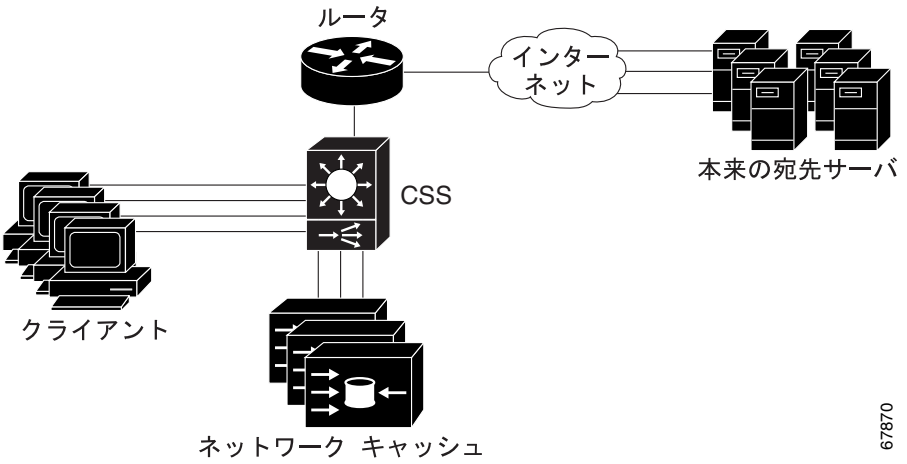
プロキシ キャッシングの使用

プロキシ キャッシングでは、各クライアントにプロキシ キャッシュの IP アドレスが設定され、このアドレスで指定されるプロキシ キャッシュにクライアントからコンテンツ要求が送信されます。また、プロキシ設定ファイルの場所を指定する URL をブラウザに設定すると、プロキシ設定を自動的に行うこともできます。各クライアントのコンテンツ要求は、プロキシ キャッシュの IP アドレスに直接送信されます。キャッシュ サーバは、要求されたコンテンツを戻す（ローカルにコピーがある場合）か、本来の宛先サーバに情報要求を送信します。

各クライアントにプロキシ キャッシュの VIP が設定されているため、プロキシ キャッシュ構成に含まれる全キャッシュ サーバが使用できない場合には、クライアント要求は本来の宛先サーバにも送信されません。

図 13-1 に、プロキシ キャッシュ構成での CSS の使用例を示します。

図 13-1　　プロキシ キャッシュ構成の例



逆プロキシ キャッシングの使用

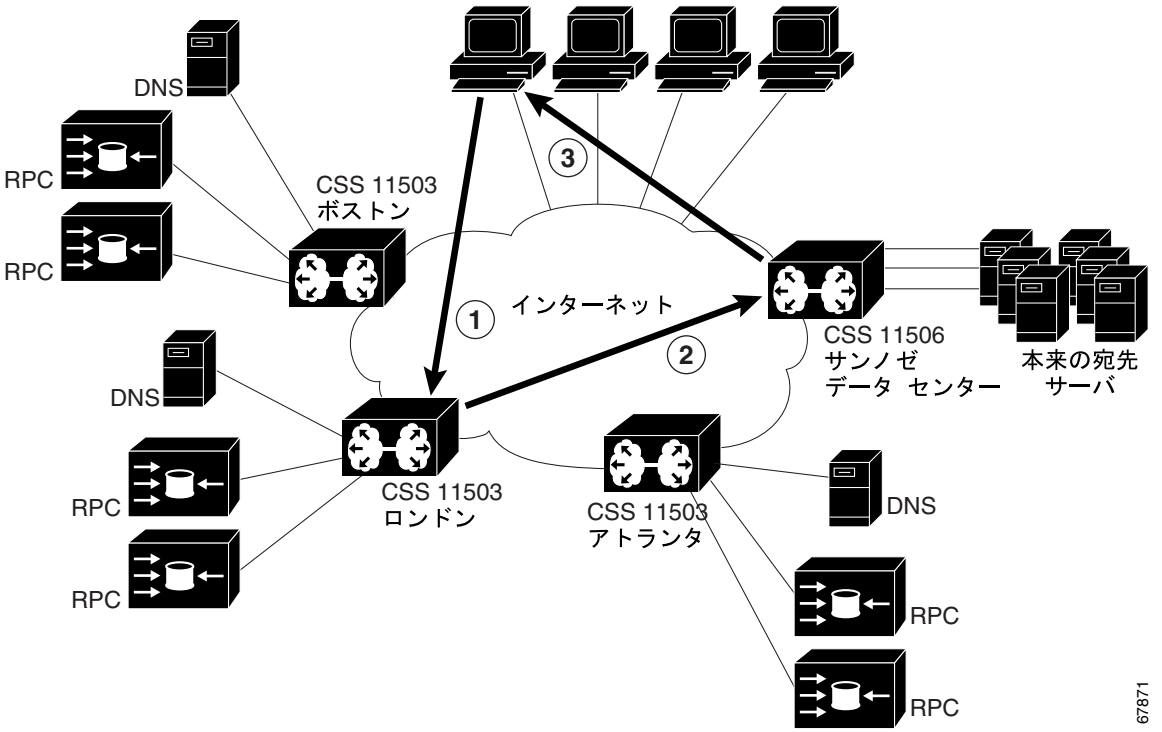
逆プロキシ キャッシュ構成では、プロキシ サーバにインターネットでルーティング可能な IP アドレスを設定します。クライアントからの要求は、ドメイン名の Domain Name System （DNS; ドメイン ネーム システム）解決に基づきプロキシ サーバに送られます。クライアント側では、逆プロキシ サーバがあたかも Web サーバのように認識されます。

通常のプロキシ キャッシュ構成では、プロキシ サーバがクライアントのプロキシ（代理処理者）として機能しますが、逆プロキシ キャッシュ構成では、逆プロキシ サーバがサーバのプロキシとして機能します。また、プロキシ キャッシュおよび透過キャッシュでは、頻繁に要求されるコンテンツがキャッシュに保存されますが、逆プロキシ キャッシュでは、特定のコンテンツがキャッシュに保存されます。逆プロキシ キャッシュには次の 2 つの基本的な機能があります。

- 地理的に分散した領域でのコンテンツ レプリケーション
- ロード バランシングのためのコンテンツ レプリケーション

図 13-2 は、CSS 11506 と CSS 11503 を含む逆プロキシ キャッシュ構成の例を示しています。

図 13-2　逆プロキシ キャッシュ構成の例



透過キャッシングの使用

透過キャッシングでは、ブラウザ側に存在を意識させない透過的なキャッシュ サーバを配置します。キャッシュ サーバを指すようにブラウザを設定する必要はありません。クライアントが以前に要求した着信インターネット データがキャッシュ サーバに複製、保存されます。

CSS に透過キャッシングを設定すると、その CSS はクライアントからインターネットに搬出されるデータ要求を代行受信し、ネットワークのキャッシュ サーバにリダイレクトします。キャッシュ サーバは、要求されたコンテンツを戻す（ローカルにコピーがある場合）か、そのコンテンツを提供している本来の宛先サーバに新たな要求を送信します。

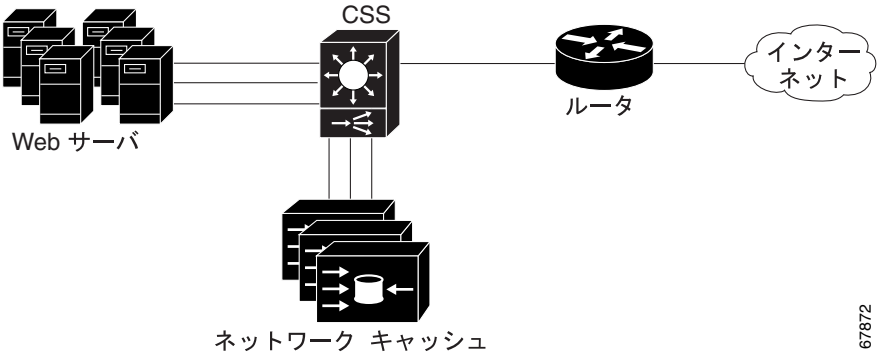
透過キャッシュ構成に含まれるすべてのキャッシュ サーバが使用できない場合、CSS はすべてのクライアント要求を本来の宛先サーバに送信します。

透過キャッシング構成では、次の効果が得られます。

- HTTP トラフィックによるネットワークの輻輳の軽減
- ネットワーク効率の向上
- インターネット経由で情報を取得するのではなく、ローカルに保存してある同じ情報にアクセスすることによる、クライアント要求への応答時間の削減

図 13-3 に、典型的な透過キャッシュ構成の例を示します。

図 13-3　透過キャッシュ構成の例



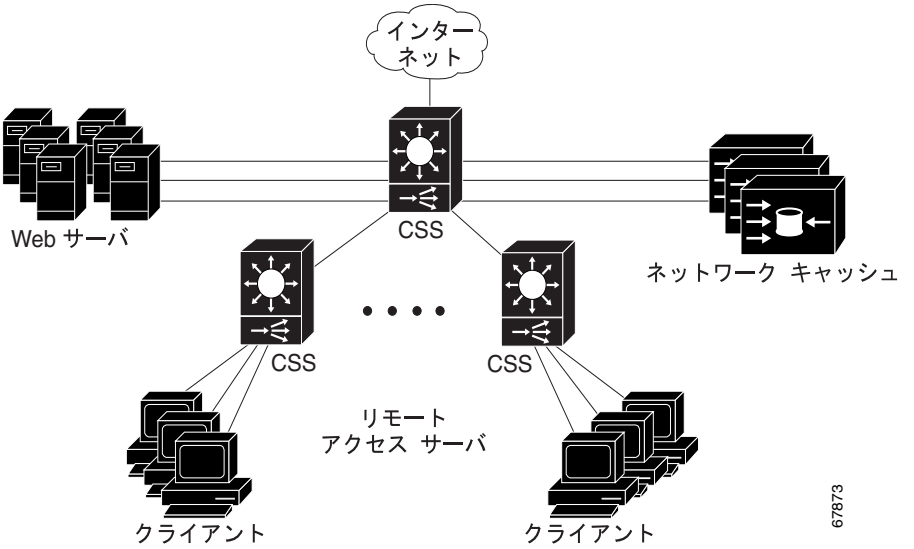
キャッシュ クラスタリングの使用

複数のキャッシュを一箇所に配置することをキャッシュ クラスタリングと呼びます。キャッシュ クラスタリングを行うと、次の効果を得られます。

- スケーラビリティ
- 冗長性
- 透過性
- 管理の簡素化

図 13-4 に、キャッシュ クラスタによるコンテンツ キャッシングの設定例を示します。

図 13-4　　キャッシュ クラスタ構成の例



キャッシング設定のクイック スタート

表 13-1 に、サービス serv1 をキャッシング サービスとして設定する手順を示します。それぞれの手順に、作業を実行するために必要な CLI コマンドも示します。CSS のキャッシングを設定する前に、サービス、所有者、およびコンテンツ ルールを設定してください。

（注） コンテンツ キャッシングを使用する場合、キープアライブのタイプを ICMP（デフォルト設定）に設定する必要があります。

キャッシングに使用する各コマンドの詳細は、表 13-1 以降の項を参照してください。

表 13-1　キャッシング設定のクイック スタート

作業とコマンドの例
1. サービスのタイプを指定します（**type local**、**type proxy-cache**、**type redirect**、**type transparent-cache**）。デフォルトは local です。 `(config-service[serv1])# type transparent-cache`

2. Extension Qualifier List（EQL; 拡張子修飾子リスト）を作成します（このリスト内で、CSS でキャッシュに保存するコンテンツのタイプを指定します）。
`(config)# eql graphics`
`(config-eql[graphics])#`

3. EQL の説明として、63 文字以内のテキスト文字列を引用符で囲んで入力します。 `(config-eql[graphics])# description "This EQL specifies cacheable graphic files"`

表 13-1　キャッシング設定のクイック スタート（続き）

作業とコマンドの例

4. CSS がキャッシュに保存するコンテンツの拡張子を指定します。1 ～ 8 文字のテキスト文字列を入力します。
`(config-eql[graphics])# extension jpeg`
必要に応じて、拡張子のタイプを説明することもできます。64 文字以内のテキスト文字列を引用符で囲んで入力します。
`(config-eql[graphics])# extension gif "This is a graphics file"`
`(config-eql[graphics])# exit`
`(config)#`

5. コンテンツ ルールに EQL を指定して、すべてのコンテンツ要求が、必要な拡張子と一致するようにします。 `(config-owner-content[arrowpoint.com-rule1])# url "/*" eql graphics`
6. キャッシュ コンテンツ ルールのロード バランシング方式を設定します。デフォルトはラウンドロビン方式です。 `(config-owner-content[arrowpoint.com-rule1])# balance domain`
7. フェールオーバー タイプを指定して、サービスに障害が起きた場合のコンテンツ要求の処理方法（**bypass**、**next**）を定義します。デフォルトは linear です。 `(config-owner-content[arrowpoint.com-rule1])# failover bypass`
8. EQL の設定を表示します。 `(config-owner-content[arrowpoint.com-rule1])# show eql`
9. コンテンツ ルールを表示してキャッシュ設定を確認します。 `(config-owner-content[arrowpoint.com-rule1])# show rule`

表 13-1 に示した各コマンドを実行した場合の実行設定の一例を次に示します。

```
!************************** SERVICE **************************
service serv1
  type transparent-cache
  ip address 192.168.100.100
  active

!*************************** EQL ***************************
eql graphics
  extension .jpg
  description “This EQL specifies cacheable graphic files”
  extension jpeg
  extension gif “This is a graphics file”

!************************** OWNER **************************
owner arrowpoint
  address “200 Beaver Brook Road, Boxborough, MA 01719"

  content rule1
    vip address 192.1.1.100
    add service serv1
    protocol tcp
    port 80
    url “/*” eql graphics
    balance domain
    failover bypass
```

キャッシング コンテンツ ルールの設定

キャッシングは、コンテンツ ルールを使用して設定します。キャッシング コンテンツ ルールを作成する場合は、次の設定要件を追加する必要があります。

- キャッシュをサポートするサービス タイプの指定
- キャッシュ サーバのフェールオーバー タイプの指定
- キャッシュをサポートするロード バランシング アルゴリズムの設定
- CSS がキャッシュ サービスに送信するファイル拡張子を特定する EQL の設定

（注） 無差別モードでは受信しないシステムで Inktomi Traffic Server を実行している場合、Inktomi Adaptive Redirect Module （ARM）をバイパスする（つまり、トラフィックをポート 80 ではなくポート 8080 に直接送信する）には、CSS のサービス タイプを **type proxy-cache** に指定します。CSS のサービス タイプを **type proxy-cache** に設定すると、トラフィックを Traffic Server に送信するときに、CSS が完全な Network Address Translation （NAT; ネットワーク アドレス変換）処理を行います。

サービス タイプの指定

CSS では、**type** コマンドを使用して次のキャッシュ固有のサービス タイプを指定することができます。デフォルトのサービス タイプは local です。

- **type nci-direct-return** ：サービスをダイレクト リターン用の NAT チャネル通知に指定する。逆プロキシ キャッシュと NAT ピアリングで使用します。
- **type nci-info-only** ：サービスを読み取り専用の NAT チャネル通知に指定する。逆プロキシ キャッシュと NAT ピアリングで使用します。
- **type proxy-cache** ：サービスをプロキシ キャッシュに指定する。このオプションを使用すると、キャッシュ サーバから着信する要求へのコンテンツ ルールの適用がバイパスされます。コンテンツ ルールをバイパスすることで、キャッシュと CSS 間のループを防ぐことができます。
- **type rep-cache** ：サービスを複製キャッシュに指定する。
- **type rep-cache-redir** ：サービスを、リダイレクトを行う複製キャッシュに指定する。

- **type transparent-cache** ：サービスを透過キャッシュに指定する。このサービス タイプからの要求には、コンテンツ ルールが適用されません。この場合、コンテンツ ルールをバイパスするとことで、キャッシュと CSS の間でループは発生しなくなります。

たとえば、サービス serv1 をプロキシ キャッシュに指定するには、次のように入力します。

```
(config-service[serv1])# type proxy-cache
```

（注） レイヤ 5 コンテンツ ルールは、RFC-2518 の CONNECT、GET、HEAD、POST、PUSH、PUT の各 HTTP メソッドをサポートします。また、透過キャッシュ環境では、CSS は RFC-2518 拡張メソッドの OPTIONS、TRACE と PROPFIND、PROPPATCH、MKCOL、MOVE、LOCK、UNLOCK、COPY、DELETE を認識して宛先サーバに直接転送しますが、これらのロード バランシングは行いません。RFC-2518 拡張方式をサポートするように CSS を設定する方法については、第 10 章「コンテンツ ルールの設定」を参照してください。

（注） コンテンツへの要求がルールに一致した場合に、CSS がその要求をリモート サービスにリダイレクトできるようにするには、そのコンテンツ ルールに URL を指定する必要があります。

フェールオーバー タイプの指定

デフォルトでは、CSS は linear フェールオーバー方式を使用します。この方式では、障害が発生したサービスへのコンテンツ要求が、残りの各サービス間で均等に配分されます。

キャッシュ サービスで障害が発生した場合または一時停止した場合のコンテンツ要求の処理方法を定義するには、**failover** コマンドを使用します。CSS でこの設定を使用するには、各サービスにキープアライブを設定する（キープアライブ タイプを none 以外に設定する）必要があります。デフォルトのキープアライブは ICMP です。CSS は、キープアライブの設定を使用してキャッシュ サービスをモニタし、サーバの状態とアベイラビリティを判断します。**keepalive** コマンドについては、第 3 章「サービスの設定」を参照してください。

（注）　**remove service** コマンドを使用してサービスを削除すると、再度残りのサービスのバランスが取られます。CSS はフェールオーバーの設定を使用しません。

このコマンドには、次のオプションがあります。

- **failover bypass** ：障害が発生したサービスをすべてバイパスして、コンテンツ要求をを本来の宛先サーバに直接送信する。このオプションは、プロキシ キャッシュ構成や透過キャッシュ構成で、障害が発生したキャッシュをバイパスし、コンテンツが存在するサーバにコンテンツ要求を直接送信する場合に使用します。
- **failover linear**（デフォルト）：コンテンツ要求を残りのサービス間で均等に配分する。
- **failover next** ：障害が発生したサービスの次のキャッシュ サービスへコンテンツ要求を送信する。CSS は、サービスの設定順序を参照して、コンテンツ要求のリダイレクト先のサービスを選択します。

たとえば、次のように入力します。

```
(config-owner-content[arrowpoint.com-rule1])# failover bypass
```

フェールオーバー方式をデフォルトの linear に戻すには、次のように入力します。

```
(config-owner-content[arrowpoint.com-rule1])# no failover
```

図 13-5 に、**failover next** に設定された 3 つのキャッシュ サービスを示します。ServerB に障害が発生すると、ServerB のコンテンツ要求は、コンテンツ ルールで ServerB の次に設定されている ServerC に送信されます。

図 13-5　キャッシュ サービス設定（Failover Next 例 1）

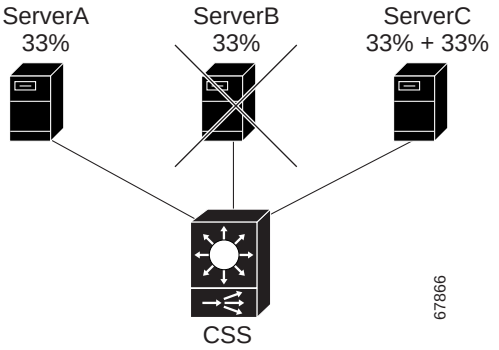


図 13-6 に示すように、ServerC に障害が発生すると、ServerC の次にはサービスが何も設定されていないので、ServerC 宛てのコンテンツ要求は ServerA に送信されます。

図 13-6　キャッシュ サービス設定（Failover Next 例 2）

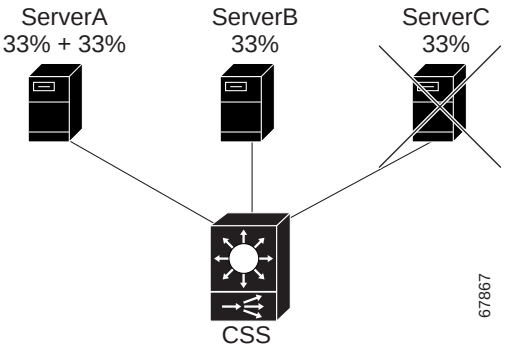


図 13-7 に、**failover linear**（デフォルト）に設定された 3 つのキャッシュ サービスを示します。ServerB を一時停止した場合または、ServerB に障害が発生した場合、CSS は残りのサービスの間で再度負荷のバランスはとりません。ServerB のキャッシュ負荷は、ServerA と ServerC の間で均等に分散されます。

図 13-7 と図 13-8 では、負荷バランスをアルファベットで示しています。

図 13-7　一時停止または障害時のキャッシュ サービス設定（Failover Linear）

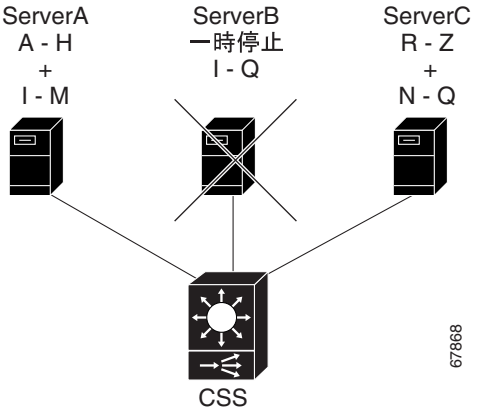
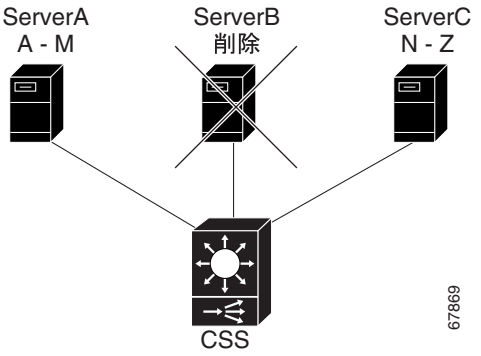


図 13-8 も、**failover linear** に設定された 3 つのキャッシュ サービスを示していますが、この例では、所有者コンテンツ モードで **remove service** コマンドを使用して ServerB を削除しています。サービスを削除した場合はフェールオーバーの設定が適用されないため、CSS は残りのサービス間で再度負荷のバランスをとります。

図 13-8　キャッシュ サービスの削除（Failover Linear）



ロード バランシングの設定

コンテンツ ルールのロード バランシング アルゴリズムを指定するには、**balance** コマンドを使用します。このコマンドは、コンテンツ設定モードで使用可能です。このコマンドには、次のオプションがあります。

- **balance aca** ：ArrowPoint Content Awareness ロード バランシング アルゴリズム（第 6 章「サービスの負荷の設定」の「サーバの負荷と重みに基づいた ArrowPoint Content Awareness (ACA) の使用」を参照）。ACA では、負荷、またはサーバの重みと負荷に基づいてサービス間でトラフィックのバランスを取ります。
- **balance destip** ：宛先 IP アドレスによる分配アルゴリズム。CSS は、同じ宛先 IP アドレスを持つすべてのクライアント要求を同じサービスに送ります。このオプションは、通常、キャッシング環境で使用します。
- **balance domain** ：ドメイン名による分配アルゴリズム。CSS は、キャッシュごとに均等にアルファベットを割り当てます。まず、最初のドットに続く最初の 4 文字のホスト タグを解析し、ドメイン名のこれらの文字を使用して、要求をどのサーバに送信すべきかを判断します。このオプションは、通常、キャッシング環境で使用します。
- **balance domainhash** ：ドメイン文字列に基づく CSS の内部ハッシュ アルゴリズム。CSS はホスト タグを解析して、ホスト名全体に XOR ハッシュを適用します。次に、XOR ハッシュの値を使用して、要求の転送先のサーバを決定します。この方式では、同じホスト タグを持つ要求がすべて確実に同じサーバに送信され、キャッシュ ヒットの確率が高まります。このオプションは、通常、キャッシング環境で使用します。

（注） プロキシ キャッシュ サービスで **domainhash** ロード バランシング方式を使用すると、キャッシュ間でサイトが重複する場合があります。これは、固定接続の最初の GET 要求でバランスが調整されるためです（続く GET 要求が、同じ指定のプロキシ サービスを持つルールに一致しない場合を除きます）。キャッシュ間で同一内容がヒットするのを避けるには、固定接続をリセットして、再マッピングを行い、コンテンツ ルールの持続性を無効にします。グローバルに **(config) persistence reset remap** コマンドを発行し、コンテンツ ルールに対して **(config-owner-content) no persistent** コマンドを発行します。

- **balance leastconn** ：最小接続アルゴリズム。このバランス方式は、実行中のサービスのうち、接続数が最も少ないサービスを選択します。
- **balance roundrobin** ：ラウンドロビン アルゴリズム（デフォルト）。ローカルとリモートのコンテンツ ドメイン サイト間で負荷を均等に分散してドメイン名を解決し、要求を解決します。
- **balance srcip** ：送信元 IP アドレスによる分配アルゴリズム。CSS は、同じ送信元 IP アドレスから送られたクライアント要求をすべて同じサービスに送信します。このオプションは、通常、キャッシング設定で使用します。
- **balance url** ：URL による分配アルゴリズム。CSS は、キャッシュごとに均等にアルファベットを割り当てます。次に、ルールに一致する URL 部分に続く最初の 4 文字を解析します。たとえば、コンテンツ ルールの URL が /news/* に設定されている場合、/news/ に続く 4 文字に基づいて負荷分散を計ります。このオプションは、通常、キャッシング環境で使用します。
- **balance weightedrr** ：加重ラウンドロビン アルゴリズム。CSS は、ラウンドロビンを使用しますが、サーバに設定された重みに基づいて、サービスに重み付けを行います。サーバはすべて、デフォルトで重み 1 に設定されています。サーバの重みを設定するには、所有者コンテンツ モードで **add service weight** コマンドを使用します。
- **balance urlhash** ：URL 文字列に基づく CSS の内部ハッシュ アルゴリズム。CSS は URL を解析し、その URL 全体に XOR ハッシュを適用します。次に、XOR ハッシュの値を使用して、要求の転送先のサーバを決定します。この方式では、同じ URL に対する要求がすべて確実に同じサーバに送信され、キャッシュ ヒットの確率が高まります。このオプションは、通常、キャッシング環境で使用します。

（注） レイヤ 5 コンテンツ ルールは、RFC-2518 の CONNECT、GET、HEAD、POST、PUSH、PUT の各 HTTP メソッドをサポートします。また、透過キャッシュ環境では、CSS は RFC-2518 拡張メソッドの OPTIONS、TRACE および PROPFIND、PROPPATCH、MKCOL、MOVE、LOCK、UNLOCK、COPY、DELETE を認識して宛先サーバに直接転送しますが、これらのロード バランシングは行いません。RFC-2518 拡張方式をサポートするように CSS を設定する方法については、第 10 章「コンテンツ ルールの設定」を参照してください。

透過キャッシュ環境（レイヤ 5 コンテンツ ルールに VIP アドレスがない場合など）では、CSS は HTTP メソッドをバイパスして宛先サーバに転送します。

たとえば、加重ラウンドロビンロード バランシングを指定するには、次のように入力します。

```
(config-owner-content[arrowpoint-rule1])# balance weightedrr
```

バランス タイプをデフォルトのラウンドロビン方式に戻すには、次のように入力します。

```
(config-owner-content[arrowpoint-rule1])# no balance
```

ダブル ワイルドカード キャッシング コンテンツ ルールの設定

レイヤ 3 および レイヤ 4 の TCP/IP トラフィックを最適化するには、透過キャッシュで使用するコンテンツ ルールを、VIP アドレスまたはポート番号を指定しないで設定します。このような設定は、バックエンド サーバにインテリジェンス機構が組み込まれている無線環境で、特に有用です。

コンテンツ ルールの他のすべての基準にクライアント要求が適合する場合、要求される VIP またはポートに関わらず、その要求はそのコンテンツ ルールに一致します。これをダブル ワイルドカード キャッシング ルールと呼びます。この場合でも、ルールにはプロトコルを指定する必要があります。通常、このタイプのルールを使用するのは、**transparent-cache** タイプのサービスをロード バランシングしている場合ですが、その他のサービス タイプでも使用できます。

（注） ダブル ワイルドカード ルールを必要とする設定を行った場合は、クライアントがサーバの IP アドレスに直接接続しようとするときにもそのクライアント要求がこのルールに一致するので、注意が必要です。

コンテンツ要求でのキャッシュ バイパスの有効化

ここでは、コンテンツ要求でのキャッシュ バイパスの有効化について説明します。

param-bypass コマンドの使用

区切り文字の「#」と「?」は、その直後の引数にコンテンツが依存していることを示します。サーバから戻されたコンテンツはコンテンツ要求自身に依存するため、戻されたコンテンツをキャッシュに保存することはできません。

param-bypass コマンドを使用すれば、特殊な区切り文字が検出されたコンテンツ要求に、透過キャッシュをバイパスさせることが可能です。このコマンドには、次のオプションがあります。

- **param-bypass disable**（デフォルト）：特殊な区切り文字を含むコンテンツ要求は、透過キャッシュをバイパスしない。
- **param-bypass enable** :特殊な区切り文字を含むコンテンツ要求は透過キャッシュをバイパスし、宛先サーバに転送される。

たとえば、**param-bypass** コマンドを有効にするには、次のように入力します。

```
(config-owner-content[arrowpoint-rule1])# param-bypass enable
```

cache-bypass コマンドの使用

デフォルトでは、要求されたコンテンツがキャッシュに存在しない場合、proxy-cache タイプまたは transparent-cache タイプのサービスから本来の宛先サーバに送信される要求にはコンテンツ ルールは適用されません。**no cache-bypass** コマンドを使用すると、proxy-cache タイプまたは transparent-cache タイプのサービスからの要求にコンテンツ ルールを適用することができます。**no cache-bypass** コマンドを実行した後に、CSS をデフォルトの動作に戻すには、**cache-bypass** コマンドを使用します。

たとえば、proxy-cache タイプまたは transparent-cache タイプのサービスからの要求にコンテンツ ルールを適用するには、次のように入力します。

```
(config-service[serv1])# no cache-bypass
```

no cache-bypass コマンドを実行した後に、CSS をデフォルトの動作に戻すには、次のコマンドを入力します。

```
(config-service[serv1])# cache-bypass
```

bypass-hosttag コマンドの使用

デフォルトでは、キャッシュに格納できないコンテンツもキャッシュ ファームを通過し、バイパスされることはありません。**bypass-hosttag** コマンドを使用すると、Client Side Accelerator（CSA; クライアント側アクセラレータ）として設定した CSS に、キャッシュ ファームをバイパスして本来の宛先サーバとの接続を確立させ、キャッシュできないコンテンツを取得することが可能です。CSA の元の IP アドレスを検索するには、host tag フィールドに示されているドメイン名を使用します。

（注）　**bypass-hosttag** コマンドは、CSS が CSA 環境で動作している場合だけに使用します。CSA の詳細については、『*Cisco Content Services Switch Advanced Configuration Guide*』を参照してください。

次にコマンドの入力例を示します。

```
(config-service[serv1])# bypass-hosttag
```

キャッシュできないコンテンツに対してキャッシュのバイパスを無効にするには、次のように入力します。

```
(config-service[serv1])# no bypass-hosttag
```

透過キャッシュのためのネットワーク アドレス変換（NAT）の設定

CSS のデフォルトでは、透過キャッシュ（transparent-cache）タイプのサービスについては、宛先ネットワーク アドレス変換（NAT）は実行されません。透過キャッシュ タイプのサービス で宛先のネットワーク アドレスを変換する（NAT）には、**transparent-hosttag** コマンドを使用します。このコマンドにより、（CSS がキャッシュに転送した）クライアント パケットの宛先アドレスが、要求されたドメインに到達できるよう、本来の宛先サーバの IP アドレスに NAT 処理されます。このコマンドを使用すると、CSS がすべての高速化ドメインに対して定期的に行う DNS ルックアップに基づいて、キャッシュに元のサーバの最新 IP アドレスを常に保持できます。

代替手段として、本来の宛先サーバの IP アドレスをすべて手動でキャッシュに設定する方法もあります。ただし、キャッシュは必ずしも静的 IP アドレスの設定をサポートしているとは限りません。また、静的に設定された IP アドレスは、宛先サーバの IP アドレスが変わると使用できなくなります。DNS 解決をサポートし、コンテンツの取得に DNS 応答を使用するキャッシュまたは、本来の宛先サーバの IP アドレスを設定できるキャッシュでは、**transparent-hosstag** の使用は必須ではありません（ただし、使用することが推奨されます）。

（注）　**bypass-hosttag** コマンドは、クライアント側アクセラレータ（CSA）環境で CSS が 動作している場合だけに使用します。CSA の詳細については、『*Cisco Content Service Switch Advanced Configuration Guide*』を参照してください。

次にコマンドの入力例を示します。

```
(config-service[serv1])# transparent-hosttag
```

透過キャッシュ タイプのサービスでの宛先 NAT を無効にするには、次のように入力します。

```
(config-service[serv1])# no transparent-hosttag
```

ネットワーク アドレス変換ピアリングの設定

NAT ピアリングを使用すると、クライアントは CSS を経由してリモート Web サイトに接続し、最短のネットワーク パスで応答トラフィックを得ることができます。クライアントからサーバへの転送パスでは 2 つの CSS 間の TCP 接続を利用しますが、サーバからクライアントに戻るパスでは同じ CSS を通過して戻るのではなく、最短のネットワーク ルートが使用されます。

（注） NAT ピアリングを使用するには、CSS 拡張機能セットのライセンスが必要です。

NAT ピアリングを使用すると、CSS で次の処理を実行できます。

- リモート CSS へのクライアント接続の転送
- リモート CSS での最終変換処理。これにより、戻りのトラフィック パケットはクライアントに通じる任意のネットワーク パスを使用できます。
- 元のサーバへトラフィックが転送された場合のクライアント IP アドレスの保存

（注） Adaptive Session Redundancy （ASR; 適応型セッションの冗長化）では NAT ピアリングをサポートしません。ASR の詳細については、『*Cisco Content Services Switch Global Server Load-Balancing Configuration Guide*』を参照してください。

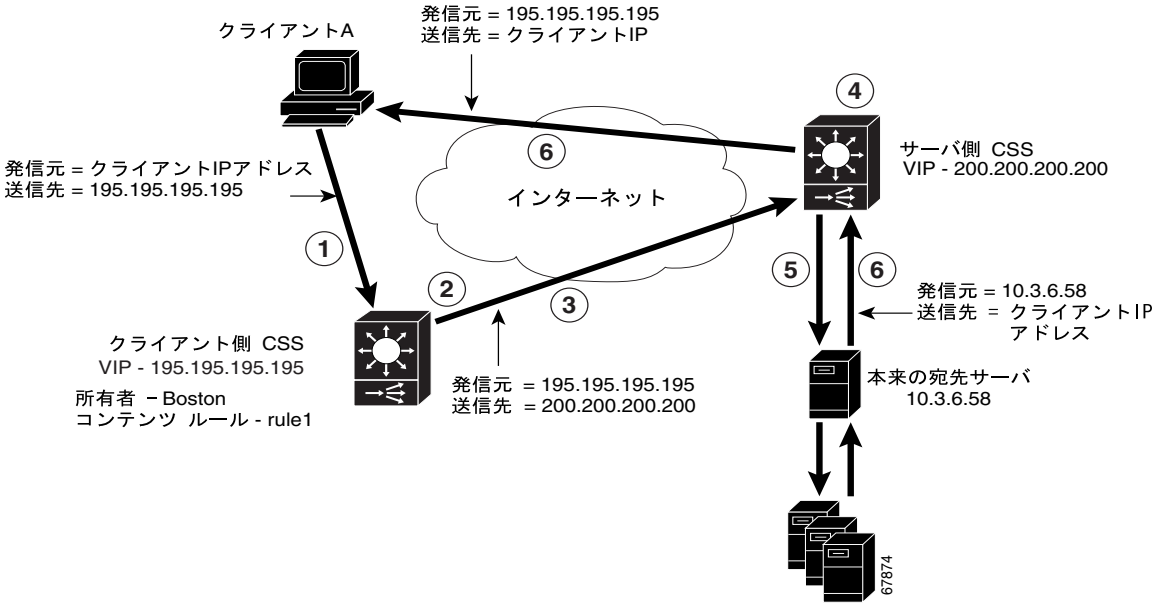
TCP フロー上で NAT 変換を行うために、クライアント側の CSS は、NAT チャネル経由でサーバ側の CSS にトラフィックを転送します。このチャネルでは、NAT Channel Indication （NCI; NAT チャネル通知）オプションという特別な TCP オプションが使用されます。このオプションは、NAT パラメータが使用されていることをサーバ側の CSS に示し、元の送信元 IP アドレスと宛先 IP アドレス、および TCP ポート番号を保持しています。また、このオプションはスプーフ ビットも持ち、フローの一部がスプーフされたこと、および送信先 CSS がパケットの情報を使用して戻りパスについて NAT 変換を行う前に、残りの転送パスを確立する必要があることも示します。

（注） スプーフィングは、CSS がロード バランシングを決定するために HTTP 要求から情報（たとえばホスト タグ、ファイル名、ファイル拡張子など）を要求すると発生します。

サーバ側の CSS では、クライアントのアドレスおよびポートが保存されます。これにより、本来の宛先サーバ側で元のトラフィックの送信元アドレス データに基づく統計情報を保持することが可能になり、戻りパスを転送パスから独立させることができます。

図 13-9 に NAT ピアリングの例を示します。図の次に、この例の手順を説明します。

図 13-9　NAT ピアリングの設定例



1. クライアント A が、クライアント側の CSS（CSS1、VIP 195.195.195.195）に /bostonInfo.html のコンテンツ要求を送信します。

2. クライアント側の CSS は、要求をコンテンツ ルールと照合します。このルールは、サーバ側の CSS （CSS2、VIP2 200.200.200.200）にあるサービスを指定しています。サーバ側の CSS サービスは、サービス タイプ **nci-direct-return** に設定されています。このサービス タイプによって、クライアント側の CSS は、サーバ側の CSS に送信する TCP パケットに NCI オプションを含めるように指示されます。レイヤ 5 のルールが一致する場合は、NCI オプションのスプーフ ビットが設定されます。
3. クライアント側の CSS は、サーバ側の CSS に TCP パケットを送信します。送信元アドレスのグループ マッピングによって、クライアント A の送信元アドレスおよびポートがクライアント側の CSS からの送信元アドレスとポートにマップされます。TCP パケットには、クライアント側の CSS の送信元情報、サーバ側の CSS の送信先情報、およびクライアント A からの元の送信元情報と送信先情報が含まれます。
4. サーバ側の CSS は、パケットにスプーフ ビットが設定されているかどうか判別します。スプーフ ビットが設定されている場合、CSS は接続がスプーフされるまで NAT 情報を保持します。サーバ側の CSS は、転送パスと戻りパスを設定します。次に、クライアント側の CSS からの要求をコンテンツ ルールと照合します。

（注） サーバ側の CSS （図 13-9）は、VIP ルールがローカル サービス、プロキシ キャッシュ サービス、または透過キャッシュ サービスに送信される場合、パケット内の NCI オプションを使用します。

5. サーバ側の CSS は、元のサーバに要求を送信します。このとき、宛先 IP アドレスは元のサーバの IP アドレスに変換し、送信元 IP アドレスはクライアントの IP アドレスに変換します。
6. 元のサーバは、クライアント A に直接応答します。パケットがサーバ側の CSS を通過するときに、CSS は送信元 IP アドレスを CSS1 の VIP に変換します。宛先 IP アドレスはクライアント IP アドレスです。

NAT ピアリングの設定

NAT ピアリングの設定は、すべてクライアント側の CSS で行います。設定の注意点は次のとおりです。

- NCI サービスを **nci-direct-return** に設定する場合、サービスをサーバ側の CSS の VIP に向けて、接続のエンドポイントを示す必要があります。サーバ側の CSS では常に **nci-direct-return** オプションを使用し、サーバが認識する送信元アドレスおよびポートを変更します。nci-direct-return サービスをクライアント側で使用すると、戻りパスが直接クライアントに戻るように変更されます。
- NCI サービス タイプを指定する場合は、次を指定する必要があります。
  - 別の CSS にある VIP を表すために **type nci-direct-return** を指定
  - すべての Web サーバに対して **type nci-info-only** を指定

表 13-2 で、図 13-9 の設定に基づいて、コマンド例を使用しながら NAT ピアリングの設定に必要な手順を説明します。NAT ピアリングはレイヤ 5 と同様にレイヤ 3 ルールにも適用されるため、表 13-2 に示されているポート、プロトコル、および URL のルール例はオプションです。

表 13-2　NAT 設定のクイック スタート

作業とコマンドの例

1. クライアント側の CSS （CSS1）で、コンテンツ ルールを作成してサーバ側の CSS （CSS2）をサービスに設定します。
a. サービス CSS2 を作成します。
`CSS1 (config)# service CSS2`
b. CSS2 の VIP をサービスの IP アドレスに設定します。
`CSS1 (config-service[CSS2])# ip address 200.200.200.200`
c. CSS2 の サービス タイプを nci-direct-return に設定します。
`CSS1 (config-service[CSS2])# type nci-direct-return`
d. コンテンツ ルールを有効にします。
`CSS1 (config-service[CSS2])# active`

表 13-2　NAT 設定のクイック スタート（続き）

作業とコマンドの例

2. クライアント側の CSS （CSS1）でコンテンツ ルールを作成します。このコンテンツ ルールには、クライアント側の CSS（CSS1）がサーバ側の CSS（CSS2）にトラフィックを転送するのに必要な基準を含めます。
a. 所有者を作成します。
`CSS1 (config)# owner boston.com`
b. コンテンツ ルールを命名して、所有者に割り当てます。
`CSS1 (config-owner[boston.com])# content rule1`
c. CSS1 の VIP を設定します。
`CSS1 (config-owner-content[boston.com-rule1])# vip address 195.195.195.195`
d. ポートとプロトコルを設定します。
`CSS1 (config-owner-content[boston.com-rule1])# port 80`
`CSS1 (config-owner-content[boston.com-rule1])# protocol tcp`
e. URL を定義します。
`CSS1 (config-owner-content[boston.com-rule1])# url "//bostoninfo.html/"`
f. CSS2 をサービスに追加します。
`CSS1 (config-owner-content[boston.com-rule1])# add service CSS2`
g. ルールをアクティブにします。
`CSS1 (config-owner-content[boston.com-rule1])# active`

表 13-2　NAT 設定のクイック スタート（続き）

作業とコマンドの例

3. クライアント側の CSS （CSS1）で、クライアント トラフィックに使用するソース グループを作成します。CSS1 によって、クライアント A の IP アドレスがソース グループで定義された IP アドレスに変換されます。ソース グループを設定するには、次の手順に従います。
a. ソース グループを作成します。
`CSS1 (config)# group boston`
`CSS1 (config-group[boston])#`
b. CSS1 の VIP を IP アドレスとして定義します。あとで、クライアント A の IP アドレスが変換されてこの IP アドレスになります。
`CSS1 (config-group[boston])# vip 195.195.195.195`
c. ソース グループをアクティブにします。
`CSS1 (config-group[boston])# active`

4. クライアント側の CSS （CSS1）で、Access Control List（ACL; アクセス コントロール リスト）句を作成して、ソース グループを使用する送信元 IP アドレスを指定します。句 20 は、他のトラフィックをすべて許可する必須の句です。句 20 がないと、句 10 で定義されていないトラフィックがすべて拒否されます。
`CSS1 (config)# acl 1`
`CSS1 (config-acl[1])# clause 10 permit tcp any destination content boston.com/rule1 sourcegroup boston`
`CSS1 (config-acl[1])# clause 20 permit any any destination any apply circuit-(VLAN1)`

5. サーバ側 の CSS （CSS2）で、CSS2 に接続する元のサーバを設定します。
a. 元のサーバ serv1 を作成します。
`CSS2 (config)# service serv1`
b. serv1 の IP アドレスを設定します。
`CSS2 (config-service[serv1])# ip address 10.3.6.58`
c. サーバをアクティブにします。
`CSS2 (config-service[serv1])# active`

表 13-2　NAT 設定のクイック スタート（続き）

作業とコマンドの例

6. サーバ側の CSS （CSS2）で、serv1 にコンテンツ要求を転送するために必要な基準を持つコンテンツ ルールを設定します。
a. 所有者を作成します。
`CSS2 (config)# owner boston.com`
b. コンテンツ ルールに名前を付け、所有者に割り当てます。
`CSS2 (config-owner[boston.com])# content rule1`
c. CSS2 の VIP を設定します。
`CSS2 (config-owner-content[boston.com-rule1])# vip address 200.200.200.200`
d. ポートとプロトコルを設定します。
`CSS2 (config-owner-content[boston.com-rule1])# port 80`
`CSS2 (config-owner-content[boston.com-rule1])# protocol tcp`
e. serv1 をサービスに追加します。
`CSS2 (config-owner-content[boston.com-rule1])# add service serv1`
f. URL を定義します。
`CSS2 (config-owner-content[boston.com-rule1])# url "/*"`
g. ルールをアクティブにします。
`CSS2 (config-owner-content[boston.com-rule1])# active`

表 13-2 に示したクライアント側 CSS コマンドを実行すると、次のような実行設定が得られます。

```
!************************** SERVICE **************************
service CSS2
  ip address 200.200.200.200
  type nci-direct-return
  active

!************************** OWNER ****************************
owner boston.com

  content rule1
    protocol tcp
    port 80
    url "//bostoninfo.html/"
    vip address 195.195.195.195
    add service CSS2
    active

!************************** GROUP ****************************
group boston
  vip address 195.195.195.195
  active

!************************** ACL ******************************
acl 1
clause 10 permit tcp any destination content boston.com/rule1
sourcegroup boston
clause 20 permit any any destination any apply circuit-(VLAN1)
```

表 13-2 に示したサーバ側 CSS コマンドを実行すると、次のような実行設定が得られます。

```
!************************** SERVICE **************************
service serv1
  ip address 10.3.6.58
  active

!************************** OWNER ****************************
owner boston.com

  content rule1
    vip address 200.200.200.200
    add service serv1
    protocol tcp
    port 80
    url "/*"
    active
```