L'amministrazione dei servizi web

Simone Piccardi
piccardi@truelite.it
Revisione: 1350

L'amministrazione dei servizi web
Prima edizione

Simone Piccardi

Truelite Srl – http://www.truelite.it – info@truelite.it

L'amministrazione dei servizi web – Prima edizione

Questa documentazione libera è stata sviluppata all'interno delle attività formative effettuate da Truelite S.r.l. Il materiale è stato finanziato nel corso della realizzazione dei corsi erogati dall'azienda, e viene messo a disposizione di tutti sotto licenza GNU FDL.

Questo testo, insieme al resto della documentazione libera realizzata da Truelite S.r.l., viene distribuito su internet all'indirizzo:

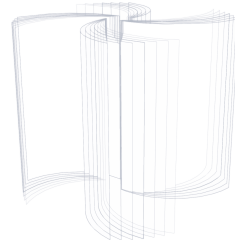
`http://svn.truelite.it/truedoc`

dove saranno pubblicate nuove versioni ed aggiornamenti.

Società italiana specializzata nella fornitura di servizi, consulenza e formazione esclusivamente su GNU/Linux e software libero.

Per informazioni:

Truelite S.r.l
Via Monferrato 6,
50142 Firenze.
Tel: 055-7879597
Fax: 055-7333336
e-mail: info@truelite.it
web: http://www.truelite.it



Indice

Capitolo 1

I servizi Web

1.1 Il *World Wide Web*

Il servizio più noto di quelli disponibili su internet, tanto che spesso viene usato come sinonimo della rete stessa, è il cosiddetto *World Wide Web*, abbreviato nella sigla WWW. In questa prima sezione faremo una breve introduzione a questo servizio, descrivendo le caratteristiche essenziali del protocollo HTTP su cui è basato.

1.1.1 Il protocollo HTTP

Il protocollo HTTP (sigla che sta per *HyperText Transfer Protocol*) nasce per permettere di trasferire dei dati sulla rete in maniera indipendentemente dal sistema che lo utilizza. Il protocollo è stato creato nei primi anni '90 al CERN[1] per veicolare contenuti ipertestuali scritti in un apposito linguaggio, l'HTML (*HyperText Markup Language*) in cui sono scritte, usando una sintassi basata appunto su dei *marcatori*, le pagine web. Questo consentiva che esse fossero disponibili direttamente sulla rete ed in grado di referenziarsi direttamente anche se poste su macchine diverse, in modo da semplificare la gestione della pubblicazione e diffusione delle informazioni, creando quella rete di collegamento fra documenti che venne chiamata *World Wide Web* e che per molti è diventata un sinonimo di Internet.

Il protocollo prevede la presenza di un server, il *server web*, su cui sono posti i file da distribuire. Questi possono essere trasmessi direttamente su una semplice connessione; è cioè possibile leggerne il contenuto con una sessione`telnet` usando gli opportuni comandi del protocollo, ma in genere vengono letti da un apposito client, il *browser*, che si incarica anche di interpretarne il contenuto[2] e di scaricare eventuali altri file da essi richiesti (come eventuali immagini) per poi disegnare in una finestra quella che viene chiamata una "*pagina web*".

[1] il *Centro Europeo Ricerche Nucleari* è uno dei più grandi laboratori di fisica sperimentale del mondo, famoso per gli acceleratori e le ricerche sulle particelle elementari.

[2] non tratteremo qui il *linguaggio* HTML, sia semplicemente noto che questo prevede varie direttive per la formattazione del testo e la composizione di una pagina, inoltre i browser più moderni sono in grado di interpretare, oltre all'HTML, anche altri formati (come SVG, MATHML, CSS, ecc.) e generare pagine con il relativo contenuto.

Protocollo	Significato
`HTTP/0.9`	Prima versione del protocollo, definiva soltanto una modalità di trasferimento di dati attraverso internet, non viene praticamente più usata da nessuno.
`HTTP/1.0`	La prima versione *stabile* del protocollo, standardizzata nell'RFC 1945, è ancora utilizzata da molti server e supportata da tutti i browser.
`HTTP/1.1`	La versione più recente, è standardizzata dall'RFC 2616; è la versione che attualmente è più utilizzata ed è supportato da tutti i server ed i browser più recenti.

Tabella 1.1: Le versioni del protocollo HTTP.

In tab. **??**, utilizzando la stringa che le identifica (e che viene riportata anche nelle comunicazioni interne del protocollo), si sono riportate le tre versioni finora definite per il protocollo HTTP, ordinate in successione temporale. Per ciascuna di esse si è fornito un breve sommario sulle RFC di definizione e le principali caratteristiche.

La prima versione del protocollo è stata la 0.9, identificata come `HTTP/0.9`, che permetteva semplicemente uno scambio di dati grezzi attraverso la rete. Questa versione supportava soltanto la capacità di richiedere un file, inviato dal server al client. Il protocollo in questa prima versione era estremamente semplice: il client apre una connessione verso il server ed invia una richiesta, il server trasmette indietro il file indicato nella richiesta e chiude la connessione. La semplice chiusura della connessione identifica il completamento del trasferimento.

Come il numero stesso testimonia, la `HTTP/0.9` era una versione preliminare (e non standardizzata). Benché tutti i server che implementano i protocolli successivi rispondano correttamente a queste richieste, `HTTP/0.9` è talmente primitivo da essere di scarsa utilità ed al giorno d'oggi è totalmente inutilizzato.

Lo sviluppo di browser e server portò a numerosi miglioramenti rispetto alla prima versione del protocollo, come l'introduzione del supporto per scambiare i file in formato MIME (vedi sez. 1.5.2), che permetteva sia di specificare in maniera standardizzata il tipo di contenuto scambiato, che di avere l'infrastruttura per la possibilità di scambiare informazioni sui dati, sulle richieste e le risposte trasmesse.

Inoltre oltre alla possibilità di richiedere un file vennero introdotte nuove operazioni (i metodi illustrati in sez. 1.1.2) come la richiesta della sola intestazione o l'invio di informazioni. La documentazione di queste innovazioni portò alla creazione della successiva versione del protocollo, identificata come `HTTP/1.0`, che venne standardizzato nell'RFC 1945.

Il fatto che il protocollo `HTTP/1.0` fosse nato come documentazione di uno stato di fatto comportava però il problema che non tutti i server implementassero completamente le specifiche, inoltre erano già stati rilevati dei problemi con gli effetti dell'esistenza di proxy e cache, per questo motivo venne sviluppata la standardizzazione di una nuova versione del protocollo, identificata come `HTTP/1.1`, definito inizialmente nell'RFC 2068, ed attualmente aggiornato dall'RFC 2616.

Il nuovo protocollo manteneva una completa compatibilità con le operazioni definite dal precedente, introducendo però una serie di nuove funzionalità le principali delle quali vengono usualmente riassunte nel seguente elenco:

- la possibilità di usare *connessioni persistenti* (detta *persistent connections*).

- la possibilità di spezzare il trasferimento di un file in tronconi successivi (detta *chunked transfers*).
- la possibilità di richiedere il trasferimento di parti del contenuto di un file (detta *byte ranges*).
- la capacità di identificare il nome della macchina a cui ci si è rivolti per eseguire una richiesta (detta *hostname identification*).
- il supporto per le operazioni dei *proxy*[3] (detta *proxy support*).
- la possibilità di negoziare i contenuti (detta *content negotiation*).

La prima funzionalità, quella delle *connessioni persistenti*, nasce dal fatto che, con le precedenti versioni del protocollo, in presenza di pagine web contenenti riferimenti ad altri dati (ad esempio le immagini), si doveva effettuare una nuova connessione per scaricare ciascuno di essi,[4] comportamento che causa un notevole degrado nelle prestazioni. Il nuovo protocollo prevede che la dimensione del file sia comunicata inizialmente (tramite le opportune intestazioni, vedi sez. 1.1.3) rendendo possibile sia trasmettere più file su una unica connessione, che avere più richieste.

Per questo motivo con la versione del protocollo `HTTP/1.1` il default prevede che tutte le connessioni si mantengono attive per un certo tempo anche dopo aver completato la trasmissione di un file, in modo che il server sia in grado di ricevere ulteriori richieste. È comunque possibile, usando un'apposita intestazione, tornare al comportamento originale, richiedendo la chiusura immediata della connessione da parte del server dopo che questo ha completato la sua risposta.

La seconda funzionalità nasce come complemento alla prima; il problema si pone infatti per i file creati dinamicamente, dei quali non è possibile sapere a priori le dimensioni. Non usando connessioni persistenti il problema non si pone, in quanto basta chiudere la connessione una volta che si sono trasmessi tutti i dati generati, ma con le connessioni persistenti occorre conoscere la dimensione di un file prima di inviarlo. Senza questa funzionalità l'unica possibilità in caso di creazione dinamica è quella di aspettare la conclusione della generazione del file prima di eseguirne la trasmissione, ma se il file è di grandi dimensioni questo comporta la necessità di dover mantenere in memoria tutto quanto, con un forte impatto negativo nell'uso delle risorse da parte del server.

Per questo motivo è stato introdotto il cosiddetto *chunked transfer* che consente ad un server di eseguire la trasmissione di un file un pezzo alla volta, cosicché, in caso di generazione dinamica, si può spedire la parte che è disponibile al momento, senza che sia necessario sapere fin dall'inizio quale sarà la lunghezza complessiva.

La terza funzionalità è sostanzialmente analoga alla seconda, ma dal lato client. È con questa funzionalità che è possibile *riprendere* lo scaricamento di un file precedentemente interrotto, senza doverlo ricominciare da capo (il cosiddetto *resume*). Con una apposita intestazione è infatti possibile richiedere l'invio soltanto di una determinata parte di un file che si vuole ricevere, specificando un intervallo di byte da scaricare (da cui il nome *byte range*).

La quarta funzionalità nasce per il supporto dei cosiddetti *domini virtuali* (che tratteremo in sez. 1.3.3), una modalità con cui diventa possibile mantenere sullo stesso server con un unico

[3] si indica con *proxy* la possibilità di far fare ad un altro server da intermediario (come vedremo in sez. **??**) per le richieste HTTP, consentendogli eventualmente anche di operare da *cache*, cioè di memorizzare localmente i file già richiesti in modo da non doverli riscaricare ad una richiesta successiva.

[4] si ricordi che la versione classica del protocollo notifica la conclusione della trasmissione del documento con la chiusura della connessione da parte del server; questo comporta un documento per connessione.

indirizzo IP le pagine web relative a più domini.[5] Per questo motivo il client deve, attraverso una opportuna intestazione, identificare a quale dominio si sta rivolgendo (da questo il nome di *hostname identification*). In questo modo il server potrà utilizzare l'informazione per restituire contenuti differenti a seconda del *dominio* a cui si è fatta la richiesta.

La quinta funzionalità attiene alla possibilità di utilizzare ulteriori intestazioni per una migliore gestione dei sistemi di cache o proxy, ad esempio diventa possibile determinare se un documento è già presente o meno e usare richieste condizionali per richiederne lo scaricamento soltanto quando sul server risulta essere presente una versione più recente dello stesso.

La sesta funzionalità attiene alla possibilità di avere diverse versioni di un stessa risorsa (ad esempio documenti in lingua diversa, o file in diversi formati). In questo caso si possono avere due forme automatiche per stabilire quale sia il contenuto da ricevere (da cui il nome *content negotiation*); la prima, detta *server-driven negotiation*, in cui il server effettua la selezione in base alle informazioni fornite dal client, la seconda, detta *agent-driven negotiation*, in cui il server si limita a fornire una lista delle possibili alternative ed è il client ad effettuare la scelta (o a proporre la lista per una scelta all'utente).

Per concludere, oltre a quelli relativi alle funzionalità appena illustrate, riportiamo un breve sommario degli ulteriori cambiamenti introdotti con `HTTP/1.1` rispetto ad `HTTP/1.0`:

- nuovi metodi di richiesta.
- nuove intestazioni.
- nuovi messaggi di errore.
- *digest authentication.*

1.1.2 Le *transazioni* ed i *metodi* HTTP

Finora abbiamo parlato in maniera generica delle caratteristiche delle varie versioni del protocollo, vediamo adesso con maggiori dettagli come questo funziona effettivamente. Tralasceremo la versione preliminare `HTTP/0.9` e ci concentreremo per lo più sulla più recente `HTTP/1.1`; ciò non di meno vale la pena introdurre alcuni concetti fondamentali usando la versione `HTTP/1.0`.

In generale il protocollo HTTP è un protocollo senza stati che prevede un semplice meccanismo di richiesta e risposta, guidato dal client che esegue una serie di richieste, ottenendo le corrispondenti risposte dal server. L'insieme di una richiesta e della relativa risposta è quello che che viene chiamata una *transazione* HTTP.

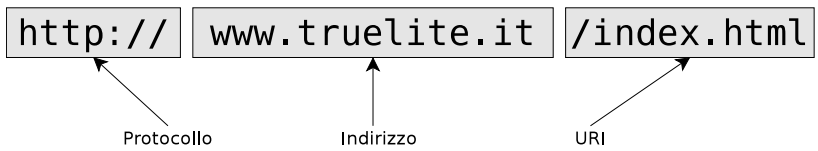


Figura 1.1: Struttura di una URL.

La richiesta è quella effettuata da un browser quando si immette l'indirizzo di un sito web da visitare, che viene espresso tramite quella particolare stringa chiamata URL (acronimo di

[5] con `HTTP/1.0` l'unica possibilità di avere più domini sulla stessa macchina era associare ciascuno di essi ad un numero IP diverso.

Uniform Resource Locator), la cui struttura si è illustrata in fig. 1.1. Una URL non è altro che una modalità per poter indicare una risorsa in maniera uniforme in qualunque luogo ci si trovi. La prima componente indica il protocollo da usare e nel nostro caso si tratta appunto di HTTP. La seconda componente indica l'indirizzo della risorsa, nel caso in fig. 1.1 si tratta di un indirizzo internet espresso tramite un nome a dominio.[6] L'ultima componente è la URI (acronimo che sta per *Uniform Resource Indicator*) che indica la risorsa come viene vista alla destinazione.

Nel caso di HTTP il client deve ricavare l'indirizzo del server web dalla URL e connettersi, dopo di che dovrà dare al server un opportuno comando, detto *metodo*, per eseguire la richiesta. La forma generica di una richiesta HTTP, riportata in fig. 1.2, prevede che al comando segua, separata da uno spazio, la URI della risorsa che si vuole ottenere, così come viene ricavata dalla URL passata al client.[7] Chiude la richiesta l'indicazione della versione del protocollo che si vuole utilizzare (nella forma di tab. 1.1); dato che dopo il comando possono essere specificate altre righe, la conclusione della richiesta viene indicata con una riga vuota (cioè con due caratteri di a capo).

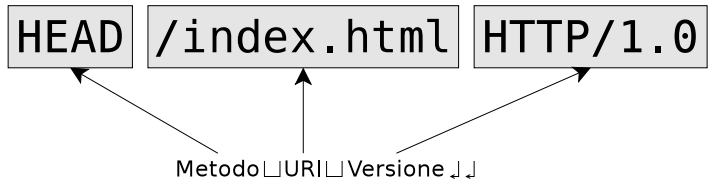


Figura 1.2: Schema generale di una richiesta con `HTTP/1.0`.

Nelle forme più elementari di richiesta il protocollo `HTTP/1.0` non richiede niente altro rispetto alla forma elementare di fig. 1.2. In alcuni casi però possono seguire sulle righe seguenti le intestazioni necessarie a dare ulteriori indicazioni; torneremo sulle varie intestazioni utilizzabili nelle richieste in sez. 1.1.3. È compito del server identificare la risorsa indicata dalla URI e restituire i relativi dati (o un errore qualora essa non esista).

I metodi allora non sono altro che i *comandi* di base del protocollo HTTP, è attraverso questi comandi che un client esegue le sue richieste nei confronti di un server; nel caso di `HTTP/1.0` i metodi previsti erano soltanto tre, riportati in tab. 1.2; il primo veniva utilizzato per richiedere una pagina, il secondo per verificarne la presenza, ed il terzo per inviare dei dati al server.

Ad una richiesta da parte del cliente il protocollo richiede debba seguire una risposta da parte del server che inizia sempre con un *codice di ritorno*, (detto *Status Code*) che identifica l'esito della richiesta (li tratteremo in sez. 1.1.4), seguito da una serie di intestazioni ed eventualmente (nel caso di `GET`) dal contenuto del file.

Come esempio si è illustrata la struttura completa di una transazione eseguita usando il metodo `HEAD` con il protocollo `HTTP/1.0` in fig. 1.3. Per realizzare la richiesta basterà utilizzare il

[6] la sintassi completa prevede una forma più complessa che permette di inserire pure un username ed una password e la porta da usare per il collegamento come alternativa a quelle standard associata al protocollo, con un qualcosa del tipo di `utente:password@indirizzo:porta`.

[7] in genere essa ha la forma di un pathname, dato che l'operazione più comune è quella di richiedere un file posto sul server contenente una pagina HTML, inizialmente si poteva fare solo questo, ma in seguito l'idea è stata estesa e una URI può indicare tanto un file quanto l'output di un opportuno programma o di un'altra operazione eseguita dal server in risposta alla richiesta di tale risorsa.

Metodo	Significato
GET	richiede il trasferimento della risorsa identificata dalla URI specificata, che può essere sia un file che uno script, in quest'ultimo caso dovrà essere trasferito al client il risultato dell'elaborazione dello stesso.
HEAD	è identico al precedente GET soltanto in questo caso verranno trasferite solo le intestazioni relative alla risorsa richiesta.
POST	con questo metodo si possono inviare (specificandole nel corpo della richiesta) delle informazioni al server, il caso classico è quello del contenuto delle *form* HTML.

Tabella 1.2: I metodi standard del protocollo HTTP/1.0.

comando telnet sulla porta 80 del nostro server web, e scrivere la linea di richiesta (terminata con due a capo), si riceverà immediatamente la risposta e la connessione verrà chiusa.

Si noti come in questo caso, avendo usato il protocollo HTTP/1.0, la richiesta sia stata estremamente semplice, e composta dalla sola linea già illustrata in fig. 1.2, cui è seguita una risposta in cui, oltre al codice di ritorno, si sono ricevute solo le intestazioni fornite dal server relative al file richiesto (fra cui quella che ne specifica il contenuto).

Metodo	Significato
PUT	richiede che i dati inseriti all'interno della richiesta vengano inviati al server e memorizzati con il nome indicato dalla risorsa indicata dalla URI, permettendo ad un client di eseguire il caricamento di un file sul server.
DELETE	richiede che la risorsa indicata dalla URI venga cancellata dal server.
OPTIONS	richiede le informazioni relative alle opzioni di comunicazione disponibili per la risorsa indicata, dalla URI, permettendo al client di determinare le capacità del server.
CONNECT	riservata all'utilizzo con i proxy che possono essere trasformati in maniera dinamica in tunnel.
TRACE	un metodo diagnostico per consentire al client di vedere cosa viene ricevuto all'altro capo della connessione (usato a scopo di controllo o di prova quando ci sono catene di richieste).

Tabella 1.3: I metodi aggiunti nel protocollo HTTP/1.1.

Nel caso di HTTP/1.1, per fornire le funzionalità aggiuntive illustrate in sez. 1.1.1, oltre ai tre metodi del protocollo HTTP/1.0 illustrati in tab. 1.2, ne sono stati definiti degli altri, riportati in tab. 1.3.

Nel caso di HTTP/1.1 una richiesta diventa necessariamente più complessa rispetto al semplice esempio di fig. 1.2 dato che in questo caso è necessario specificare delle intestazioni minime per ciascun metodo; in particolare il protocollo richiede la specificazione del dominio verso il quale si esegue la richiesta attraverso una opportuna intestazione.

Possiamo allora ripetere la richiesta di fig. 1.3 usando il protocollo HTTP/1.1, ed in questo caso dovremo modificare la richiesto aggiungendo una intestazione, ottenendo la transazione illustrata in fig. 1.4.

Anche in questo caso la richiesta potrà essere eseguita usando telnet con una connessione sulla porta 80 del server; si potrà notare che in questo caso, rispetto al precedente, la connessione

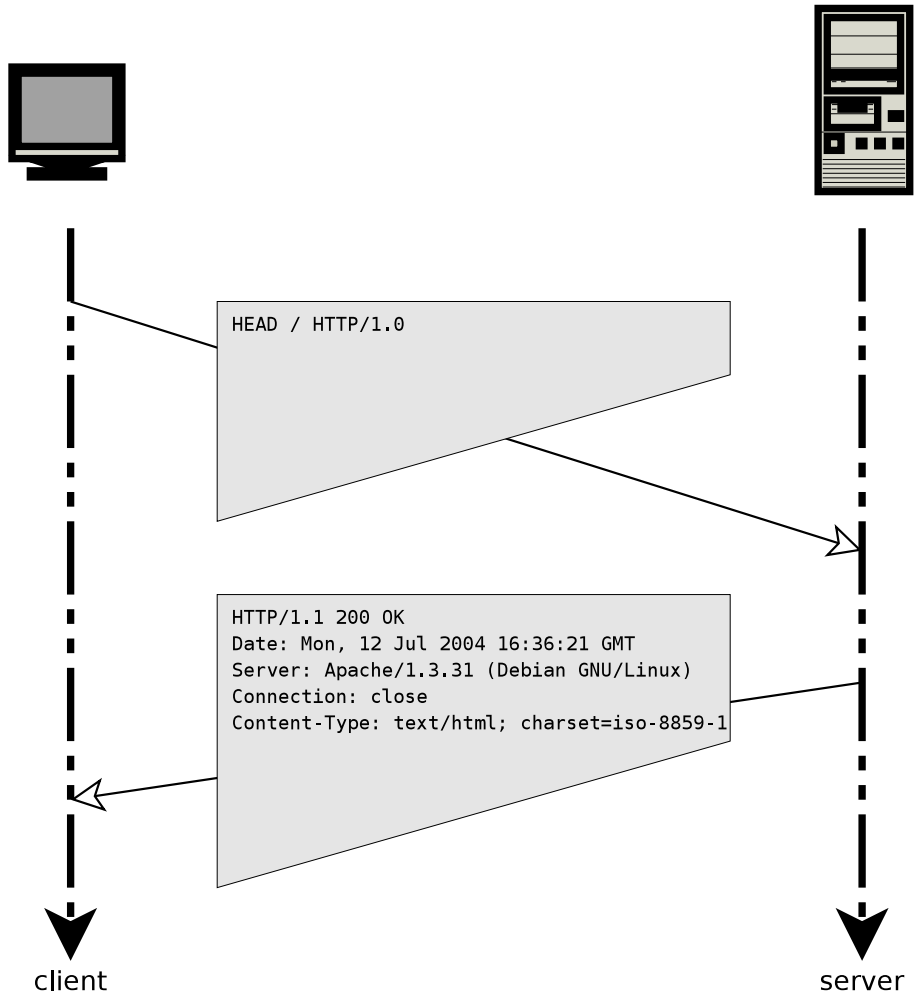


Figura 1.3: Schema di una transazione eseguita con il protocollo `HTTP/1.0`.

non viene terminata immediatamente con la ricezione della risposta, dato che, come accennato in sez. 1.1.1, con `HTTP/1.1` il default prevede che le connessioni restino attive.

1.1.3 Le intestazioni HTTP

Come già brevemente accennato in sez. 1.1.2, una parte essenziale del protocollo HTTP è costituito dalle *intestazioni* (i cosiddetti *header*) che vengono utilizzate sia nelle richieste che nelle risposte. In particolare con `HTTP/1.1` queste sono necessarie anche nelle richieste, e permettono di attivare tutte le funzionalità aggiuntive fornite dalla nuova versione del protocollo.

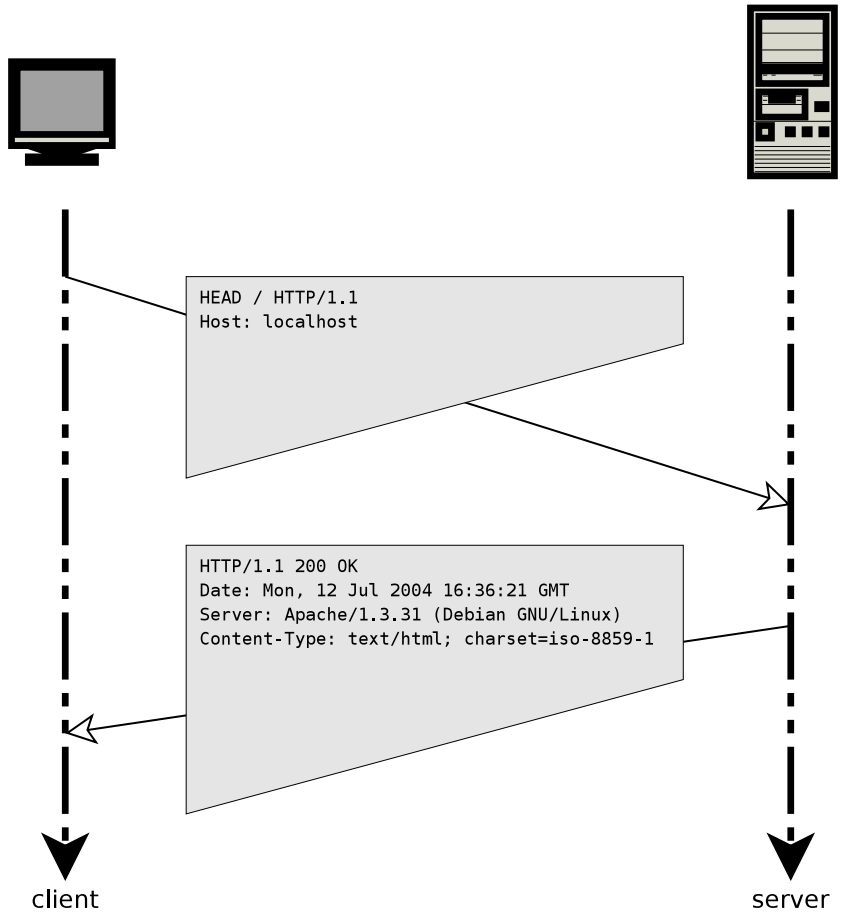


Figura 1.4: Schema di una transazione eseguita con il protocollo `HTTP/1.1`.

Il protocollo prevede che in ogni transazione alla prima riga di ciascun messaggio (che indica il metodo nella richiesta e il codice di ritorno nella risposta) possano seguire una o più righe di intestazione; ciascuna delle quali è sempre nella forma:

```
Intestazione: valore
```

se sono presenti più intestazioni queste sono terminate con una riga vuota, a cui poi potrà seguire, quando previsto, il corpo del messaggio. Le intestazioni disponibili sono standardizzate nella definizione del protocollo, nell'RFC 2616 esse vengono suddivise in quattro categorie diverse:

general intestazioni generiche applicabili sia a richieste che a risposte; contengono informazioni generali sulle modalità di trasferimento e sul protocollo. Fanno parte di questa categoria le intestazioni legate all'uso della cache, la data, e l'eventuale supporto di una versione specifica del protocollo HTTP. Le principali sono riportate in tab. 1.4.

Intestazione	Significato
`Connection`	indica la modalità della connessione, con il valore `close` si richiede l'immediata chiusura della connessione, con `keep-alive` la si mantiene attiva.
`Date`	indica data ed ora in cui è stato generato il messaggio.
`Cache-Control`	specifica i meccanismi di controllo da applicare alla gestione della cache delle pagine.

Tabella 1.4: Le principali intestazioni della categoria *general*.

request intestazioni utilizzate solo nelle richieste, che contengono informazioni aggiuntive sulle stesse o sul client. Fanno parte di questa categoria le intestazioni che permettono di indicare i tipi di file accettati dal client, il tipo di codifica, e le lingue preferite o accettate. Le principali sono riportate in tab. 1.5.

Intestazione	Significato
`Accept`	indica l'elenco dei tipi di documenti (come lista di *MIME-type*, secondo quanto illustrato in sez. 1.5.2) che sono accettabili come risposta assegnandovi delle preferenze.
`Accept-Encoding`	indica l'elenco delle codifiche (sostanzialmente il tipo di compressione) accettate per l'invio dei dati richiesti.
`Accept-Language`	indica l'elenco delle lingue preferite nell'invio dei dati richiesti.
`Host`	specifica l'indirizzo internet (numerico o simbolico) ed eventualmente la porta a cui viene indirizzata la richiesta (obbligatorio per `HTTP/1.1`).
`If-Modified-Since`	se la risorsa richiesta non è stata modificata dopo la data specificata il server non deve inviarla ma solo confermare l'informazione.
`Range`	permette di richiedere una sezione di un documento, specificando quale intervallo di byte si vuole ricevere.
`User-Agent`	descrive il client che ha originato la richiesta.

Tabella 1.5: Le principali intestazioni della categoria *request*.

response intestazioni utilizzate solo nelle risposte, che contengono informazioni aggiuntive sulle risorse richieste. Fanno parte di questa categoria le intestazioni che indicano proprietà del server o informazioni riguardo la risposta che non possono essere fornite attraverso il codice di ritorno.

Intestazione	Significato
`Age`	stima del server del tempo passato da quando la risposta è stata generata all'origine.
`Location`	usata per indicare al client una locazione alternativa per la risorsa richiesta.
`Retry-After`	fornisce una indicazione del tempo per il quale il servizio non sarà disponibile.
`Server`	informazioni riguardo al software del server.
`ETag`	una etichetta associata alla risposta a fini di gestione della cache.

Tabella 1.6: Le principali intestazioni della categoria *response*.

entity sono intestazioni che vengono usate sia nelle richieste che nelle risposte, ma che, rispetto alle *general* contengono informazioni specifiche sull'entità di quanto trasferito. Fanno parte di questa categoria le intestazioni che permettono di indicare la codifica del contenuto, la lingua e la lunghezza del file.

Intestazione	Significato
`Content-Encoding`	indica le ulteriori codifiche (come il tipo di compressione) cui è stato sottoposto il messaggio trasferito.
`Content-Language`	indica la localizzazione linguistica del messaggio trasferito.
`Content-Length`	indica la lunghezza del messaggio trasferito.
`Content-Type`	indica il formato (secondo il *MIME-type*) del messaggio trasferito.
`Expires`	indica il termine di validità della risposta (usato per la gestione della cache).
`Last-Modified`	indica il tempo cui il server ritiene sia stata eseguita l'ultima modifica.

Tabella 1.7: Le principali intestazioni della categoria *entity*.

Per una trattazione dettagliata del significato delle varie intestazioni si può fare riferimento direttamente al quattordicesimo capitolo dell'RFC 2616 dove esse sono descritte singolarmente una per una.

1.1.4 I codici HTTP

Come accennato in sez. 1.1.2 ogni risposta fornita dal server è introdotta da una riga contenente un codice di ritorno (il cosiddetto *status code*), che ci informa dell'esito della richiesta e della natura di eventuali errori.

Il significato generale del codice di ritorno è determinato dalla sua prima cifra; un elenco dei valori principali ed una classificazione dei significati dei vari codici (con alcuni esempi) è riportata in tab. 1.8.

La forma generica della riga del codice di ritorno è illustrata in fig. 1.5. Essa inizia con la versione del protocollo utilizzato e seguita dal valore numerico del codice seguita dalla stringa descrittiva secondo le prime due colonne di tab. 1.8. Ad essa poi seguiranno le intestazioni ed i dati relativi alla risposta.

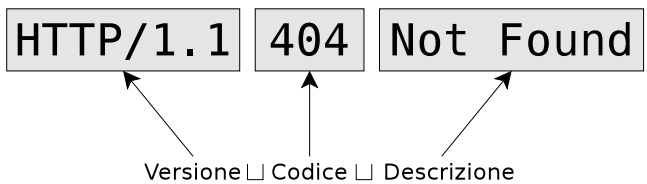


Figura 1.5: Prima riga di una risposta HTTP.

Allora come abbiamo visto nell'esempio di transazione HTTP di fig. 1.4 la risposta più comune ad una richiesta è quella del codice `200` che indica il successo della stessa e la ricezione dei dati richiesti.

Per ottenere una risposta diversa si può provare a richiedere una pagina che non esiste, ad esempio `http://www.truelite.it/index.html`; in questo caso si avrà un risultato completamente diverso, come illustrato in fig. 1.6 dove si è riportato il traffico generato dalla richiesta della suddetta pagina effettuata con Mozilla Firefox.

Dato che la pagina non esiste in questo caso si è ottenuto un codice `404`, si noti comunque nella risposta come questa non sia stata ottenuta direttamente, ma attraverso un proxy. Inoltre dato che la pagina non esiste non è più necessario mantenere attiva la connessione, ed il server

Codice/Tipo		Significato
1XX	Codici informativi	
100	Continue	segnala al client di proseguire con la richiesta, indicando che la prima parte è stata ricevuta.
101	Switching Protocols	usato in seguito ad una richiesta di cambiamento di protocollo.
2XX	Codici di successo	
200	OK	la richiesta ha avuto successo e il server ha eseguito trasferimento della risorsa richiesta.
202	Accepted	la richiesta è stata accettata ma il server non ha ancora eseguito il trasferimento della risorsa richiesta.
206	Partial Content	il server ha eseguito il trasferimento relativo ad i dati di una richiesta parziale (in risposta ad una intestazione `Range` da parte del client).
300	Codici di redirezione	
301	Moved Permanently	la risorsa richiesta è stata definitivamente spostata ad un altro indirizzo.
302	Found	la risorsa richiesta risiede temporaneamente sotto una URI diversa, siccome questa può cambiare il cliente deve continuare ad eseguire le richieste usando la URI originale.
304	Not Modified	la richiesta del client prevedeva un header `If-Modified-Since`, che permette di ricevere un file solo se modificato. In questo caso la risposta è negativa ed il client non riceverà il file perché ne è già in possesso.
4XX	Codici per errori del client	
400	Bad Request	la richiesta non è comprensibile (ad esempio la sintassi è errata).
401	Unauthorized	la richiesta richiede una autenticazione che non è stata eseguita.
403	Forbidden	il server ha capito la richiesta ma si rifiuta di soddisfarla, si usa al posto del codice `404` per notificare attraverso le opportune intestazioni il motivo del rifiuto.
404	Not Found	la risorsa richiesta non esiste.
405	Method Not Allowed	il metodo specificato non è fra quelli permessi per la risorsa richiesta.
410	Gone	la risorsa richiesta non è più presente sul server e non si è in grado di fornire un indirizzo alternativo.
5XX	Codici per errori del server	
500	Internal Server Error	errore non precisato, il server non è in grado di rispondere alla richiesta.
503	Service Unavailable	il server è momentaneamente non in grado di rispondere (ad esempio per un carico eccessivo).

Tabella 1.8: I codici di risposta del protocollo HTTP.

lo comunica con una intestazione `Connection: close`. Si noti anche come l'esempio ci mostri l'utilizzo di altre intestazioni del protocollo, come `User-Agent` e `Server`, ecc.

1.2 Una introduzione ad Apache

Apache è il web server più diffuso su internet. Deve la sua popolarità alle ottime prestazioni, alla stabilità ed alla flessibilità che gli consente di utilizzare, grazie ad una architettura modulare,

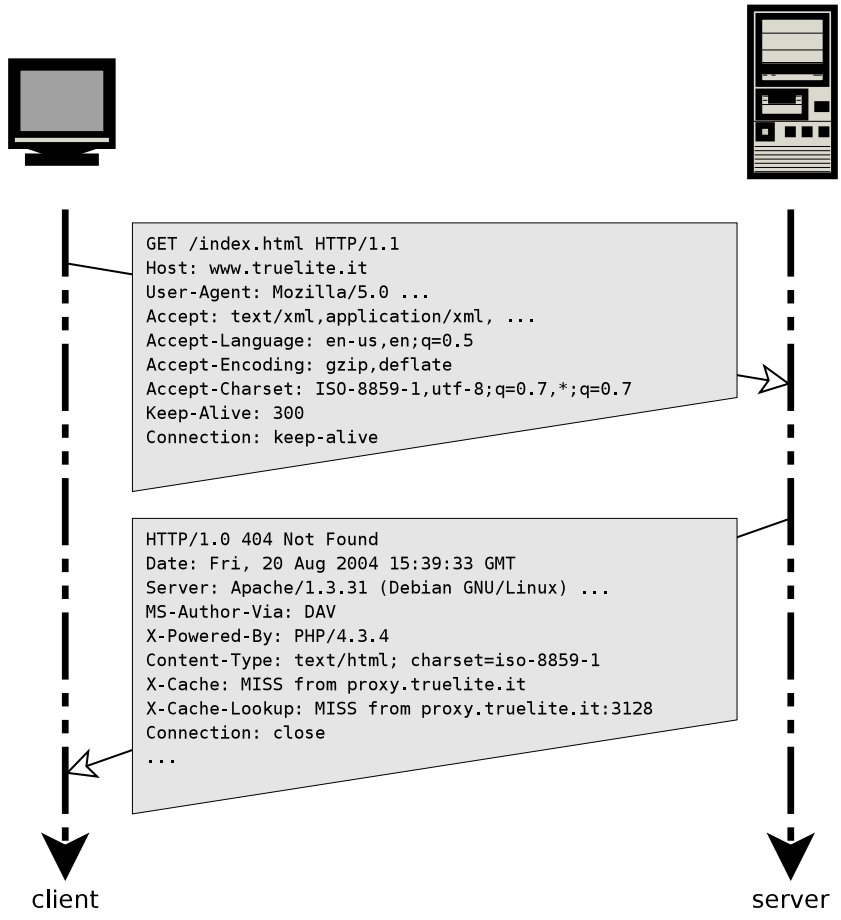


Figura 1.6: Schema di una transazione fallita per la richiesta di una pagina inesistente.

una grande quantità di estensioni e supporto per l'utilizzo di una grande varietà di linguaggi di scripting.

1.2.1 Una panoramica generale

Apache è il server HTTP più utilizzato dal 1996, ed a tutt'oggi, secondo le statistiche di Netcraft, (disponibili su `http://news.netcraft.com/archives/web_server_survey.html`) mantiene circa il 60% dei siti web presenti su Internet, costituendo uno dei più significativi esempi di successo nella storia del software libero.

Il progetto nasce tra il 1994 ed il 1995, partendo dal codice dello storico server web `httpd` sviluppato della NCSA (*National Center for Supercomputing Applications*), e deve il suo nome al fatto di essere nato come un insieme di *patch* a questo primo server. Dato che in breve le modifiche portarono ad una sostanziale riscrittura del codice originario, il progetto, nato come

"*a patch*" di `httpd`, prese il nome di Apache, scherzando sull'assonanza delle parole. La prima release ufficiale di Apache fu la 0.6.2, rilasciata nell'aprile del 1995.

Oggi il progetto è supportato dalla *Apache Software Foundation* (`http://www.apache.org`) che oltre ad esso contribuisce allo sviluppo di una grande quantità di altro software libero. Lo sviluppo è portato avanti in maniera completamente aperta da un gruppo di sviluppatori di talento che formano l'*Apache Group*, in cui si viene invitati su una stretta base di eccellenza tecnica dei contributi portati.

Apache è un server web potente e flessibile, pienamente conforme ad HTTP/1.1; la sua struttura modulare lo rende altamente configurabile e estendibile. Inoltre è estremamente portabile ed è disponibile per Windows, Netware, OS/2 e su quasi tutte le piattaforme UNIX, compreso ovviamente GNU/Linux. Tra le tante caratteristiche un breve elenco delle più rilevanti è il seguente:

- permette di avere pagine ad accesso autenticato con diversi meccanismi.
- consente i *server side include*.
- consente l'utilizzo di CGI e l'interpretazione nativa di vari linguaggi.
- permette di gestire infiniti alias e infinite regole di rewrite.
- permette la negoziazione dei contenuti in base alle capacità del client.
- ha un supporto completo per i *domini virtuali*.
- è altamente configurabile per quanto riguarda la gestione dei file di log.

Nella situazione attuale il progetto Apache mantiene stabilmente tre diverse versioni del server, la 2.0, la 2.2 e la 2.4. La vecchia versione 1.3, ormai classificata come "*legacy*", ha ormai solo interesse storico e non viene più mantenuta da nessuna distribuzione. La versione utilizzata in prevalenza nelle attuali installazioni è invece la 2.2.

Rispetto alla storica versione 1.x l'architettura delle nuove versioni è stata completamente rivista ed in particolare è stato introdotto l'uso dei *thread* che ne migliora notevolmente le prestazioni sotto Windows (ed in certe condizioni anche con Linux). Questo ha comportato l'introduzione di molti cambiamenti nelle configurazioni e di nuove funzionalità.

Infine la versione stabile più recente è la 2.4, che però è stata rilasciata da poco,[8] ed ancora non viene utilizzata da nessuna delle principali distribuzioni dato che pur essendo dichiarate stabile (ed essendolo per le funzionalità di base), non altrettanto vale per tutte le estensioni ed i moduli che in realtà vanno a costituire la gran parte delle funzionalità evolute di Apache.

Gran parte della sintassi delle direttive di configurazione è sostanzialmente identica per tutte le versioni citate, ma ci sono alcune differenze dovute ad i cambiamenti introdotti. In queste dispense tratteremo principalmente la sintassi usata da Apache 2.2, citando le differenze con le altre versioni tutte le volte che sarà necessario. Per la sua diffusione ancora piuttosto ridotta non prenderemo in considerazione la versione 2.4.

1.2.2 Installazione e file di configurazione

Essendo uno dei programmi di uso più comune Apache è presente praticamente in tutte le distribuzioni, per cui in genere tutto quello che c'è da fare è utilizzare il sistema di gestione dei pacchetti della propria distribuzione (`apache2` su Debian e SuSE, `httpd` su RedHat) e richiederne l'installazione, sempre che questa non sia già stata eseguita di default.

[8] alla stesura di queste dispense (Novembre 2012) siamo alla 2.4.3.

Si può comunque installare il programma anche dai sorgenti; a partire dalla versione 2.0 il progetto è passato all'uso estensivo degli *Autotools* del progetto GNU; per cui si può utilizzare la procedura standard nella sequenza dei comandi `./configure`, `make`, `make install`, e la configurazione dettagliata deve essere fatta attraverso le opzioni passate allo script di configurazione `configure`.

In questo caso oltre ad impostare come di norma la directory di installazione attraverso l'opzione `--prefix`, si potranno abilitare i vari moduli delle estensioni con una serie di direttive `--enable-`*`modulename`*, dove *`modulename`* è il nome del rispettivo modulo. Si tenga presente che `configure` non dà nessun avviso se il modulo non esiste, ignorando semplicemente l'opzione. In questo modo il codice del modulo sarà compilato staticamente ed inglobato direttamente nel binario del server; come vedremo in sez. 1.4.1 sarà sempre possibile attivarne le funzionalità a richiesta, ma il codice sarà comunque caricato insieme al programma.

Se si vuole che un modulo sia compilato come libreria dinamica per consentirne il caricamento a richiesta, si dovrà specificarlo passando all'opzione il parametro `shared`. Un esempio di configurazione che usa questa opzione potrebbe essere qualcosa del tipo:

```
./configure --prefix=/opt/apache --enable-rewrite=shared
```

Qualora invece si vogliano disabilitare alcuni dei moduli che sono preimpostati di default,[9] si può utilizzare la corrispondente direttiva `--disable-`*`modulename`*.

La più grossa innovazione di Apache 2 rispetto alla precedente versione 1.3 è stata l'introduzione dei cosiddetti MPM (*Multi-Processing Modules*), che portano la modularizzazione del sistema fino alle funzionalità più elementari del server, consentendo di ottimizzarne il funzionamento rispetto a diverse possibili condizioni di lavoro o all'installazione su diversi tipi di sistemi operativi.

Modulo MPM	**Descrizione**
`prefork`	realizza il modello classico di Apache 1.3, basato sull'uso dei processi.
`worker`	nuovo modulo ottimizzato per le prestazioni basato sull'uso dei *thread*.
`event`	una variante sperimentale del modulo `worker`.
`mpm_winnt`	modulo ottimizzato per Windows NT.[10]

Tabella 1.9: I vari moduli MPM disponibili con Apache 2.2.

In questo caso si tratta in sostanza di scegliere che "*tipo*" di server utilizzare fra le diverse possibilità fornite dai vari MPM, per cui al contrario dei moduli delle estensioni, fino alla versione 2.2 è possibile compilare Apache scegliendo solo un MPM alla volta. Questo viene fatto passando allo script `configure` l'opzione `--with-mpm=`*`name`*, dove *`name`* indica il nome dell'MPM scelto. Un elenco di quelli disponibili è riportato in tab. 1.9; se non se ne specifica nessuno viene utilizzato come default `prefork`, che realizza un server analogo al modello classico usato da Apache 1.3.

[9] la lista dei moduli abilitati di default è riportata nella documentazione dello script `configure`, che può essere consultata su `http://httpd.apache.org/docs/2.2/programs/configure.html`.

[10] si è citato questo particolare modulo, non disponibile per un sistema Unix, solo come esempio di un MPM che fornisce una versione del server ottimizzata per l'uso su un particolare sistema operativo; ne esistono di analoghi per Beos, Netware e OS2.

Si tenga presente che benché in teoria questi moduli siano intercambiabili, ciò è vero soltanto per le funzionalità di base, e non è detto che i moduli di estensione funzionino allo stesso modo (o funzionino affatto) indipendentemente dall'MPM scelto. In generale si sceglie il modulo `prefork` quando si vuole privilegiare la massima stabilità e compatibilità con le funzionalità presenti in Apache 1.3, si usa `worker` quando si vogliono il massimo delle prestazioni e della scalabilità. Ad esempio un modulo di estensione molto usato, come quello che fornisce il supporto per la creazione dinamica di pagine PHP, non funziona con il più prestante `worker`, e questo può costituire un esempio delle motivazioni per cui si può scegliere comunque di usare `prefork`.

Questa nuova caratteristica si riflette anche nella presenza, all'interno di una stessa distribuzione, di diverse versioni dei pacchetti di Apache 2, ad esempio Debian fornisce diversi pacchetti binari del server corrispondenti alle diverse scelte possibili per i *Multi-Processing Modules*, per cui occorrerà installare il pacchetto `apache2-mpm-prefork` se si vuole usare `prefork` o `apache2-mpm-worker` se si vuole utilizzare `worker`.

Una volta installato il server il suo comportamento è controllato dal un file di configurazione principale. In origine i file di configurazione di Apache erano tre: `httpd.conf` per la configurazione del server, `access.conf` per la gestione degli accessi e `srm.conf` per la gestione dei tipi di file e documenti. A partire dalla versione 1.3 è stato fatto riferimento ad un unico file di configurazione `httpd.conf`, dal quale è comunque possibile includere altri file con la direttiva `Include`.

Nel caso di Debian il file invece è `apache2.conf` è installato sotto `/etc/apache2`, SuSE usa sotto la stessa directory `httpd.conf` mentre per RedHat mette quest'ultimo sotto `/etc/httpd/conf`.[11] Nel prosieguo faremo comunque riferimento alla disposizione dei file di configurazione adottata da Debian.

In realtà con il passaggio ad Apache 2 anche Debian ha anche modificato l'organizzazione dei file di configurazione, mettendo a disposizione una serie di strumenti che permettono di gestire in maniera semi-automatica sia i diversi moduli di estensione che l'uso di *domini virtuali* (vedi sez. 1.3.3). Per questo motivo `httpd.conf` viene installato vuoto per compatibilità, mentre il file di configurazione principale è `apache2.conf`, all'interno del quale vengono inclusi, grazie alla citata direttiva `Include`, tutti gli altri file su cui vengono distribuite le varie informazioni di configurazione; torneremo su questo più avanti.

1.2.3 La gestione del servizio

Di norma, per non penalizzarne le prestazioni, il server Apache viene lanciato direttamente; un tempo era possibile anche avviarlo attraverso un superdemone (vedi [AGL]) come `inetd` come intermediario, ma con la versione 2 del programma questa opzione non è più disponibile. La situazione normale pertanto prevede l'utilizzo di un opportuno script di avvio del servizio che prende gli usuali parametri utilizzati dal sistema di inizializzazione di System V. Lo script utilizzato varia a seconda della distribuzione, ad esempio per Debian è `/etc/init.d/apache2` mentre per RedHat è `/etc/init.d/httpd`. Il programma può comunque anche essere eseguito direttamente, e sempre a seconda delle distribuzioni lo si può lanciare eseguendo direttamente

[11] la scelta di Debian di usare il nome del server al posto di quello del servizio è dovuta al fatto che in questa distribuzione, al contrario di altre che ne scelgono uno e forniscono soltanto quello, sono disponibili anche altri programmi che possono fare da web server e fornire il relativo servizio.

il binario con cui è stato installato con `httpd`, `apache` o `apache2`. Nel caso di Debian, cui faremo riferimento da qui in avanti, il programma usato è `apache2`.

Al suo avvio Apache legge il file di configurazione, apre i file di log, si pone in ascolto sulla porta 80 e poi fa partire un certo numero di processi in attesa di richieste (torneremo su questo in sez. 1.3.2).[12] In questo modo solo il processo iniziale ha i diritti di amministratore, mentre i processi che rispondono alle richieste girano con i privilegi dell'utente specificato in sede di configurazione (vedi sez. 1.3).

Opzione	Significato
`-d`	imposta la directory radice del server (viene sovrascritto dal valore della direttiva `ServerRoot`).
`-D`	definisce un parametro di configurazione (vedi sez. 1.3.1).
`-f`	imposta un file di configurazione alternativo a quelle od default.
`-h`	stampa una schermata di aiuto con breve sommario delle opzioni.
`-l`	stampa una lista dei moduli compilati all'interno del server.
`-L`	stampa una lista delle direttive di configurazioni disponibili con i moduli compilati nel server.
`-S`	esegue la scansione dei *Virtual Host* (vedi sez. 1.3.3).
`-t`	esegue un controllo della sintassi del file di configurazione.
`-T`	identico a `-t` ma non controlla le directory dei documenti.
`-X`	esegue il server in modalità interattiva, il server non si stacca dal terminale e non esegue figli.
`-V`	stampa i parametri con cui si è compilato il server.
`-v`	stampa la versione del server.

Tabella 1.10: Le principali opzioni a riga di comando di Apache.

In genere l'invocazione diretta del comando viene fatta solo a scopo di test; in questo caso con l'opzione `-X` lo si può fare eseguire in modalità interattiva e senza l'esecuzione di processi figli, in modo da avere tutti i risultati stampati sul terminale corrente; con `-f` inoltre si può indicare un file di configurazione alternativo rispetto a quello di default. Un elenco delle principali opzioni a riga di comando è in tab. 1.11, per la lista completa si può fare riferimento alla pagina di manuale.

Molte delle opzioni di tab. 1.10 non mettono effettivamente in esecuzione il server, e servono solo per eseguire compiti diagnostici. Con `-t`, si effettua un controllo sulla sintassi della configurazione mentre `-T` esegue lo stesso controllo senza però verificare la *Document Root* (vedi sez. 1.3.2). Il controllo è puramente formale, e non assicura nulla sulla validità della configurazione; tutto quello che si potrà ottenere è qualcosa del tipo:

```
anarres:/home/piccardi# apache2 -t
Syntax OK
```

in cui non sono stati rilevati errori di sintassi, in caso di errore al posto di `Syntax OK` si sarebbe ottenuto un `Syntax Error`.

[12] in realtà questa è la modalità di funzionamento tradizionale, realizzata dall'MPM `prefork`, nel caso di `worker` le diverse richieste vengono gestite con l'uso dei *thread*.

L'opzione `-S` permette di eseguire una scansione della configurazione per elencare i *Virtual Host* (vedi sez. 1.3.3), in essa presenti. Infine con `-V` si potranno stampare a video i parametri con cui si è compilato il server, con qualcosa del tipo:

```
anarres:/home/piccardi# apache2 -V
Server version: Apache/2.2.16 (Debian)
Server built:   Sep 13 2012 02:54:14
Server's Module Magic Number: 20051115:24
Server loaded:  APR 1.4.2, APR-Util 1.3.9
Compiled using: APR 1.4.2, APR-Util 1.3.9
Architecture:   64-bit
Server MPM:     Prefork
  threaded:     no
    forked:     yes (variable process count)
Server compiled with....
 -D APACHE_MPM_DIR="server/mpm/prefork"
 -D APR_HAS_SENDFILE
 -D APR_HAS_MMAP
 -D APR_HAVE_IPV6 (IPv4-mapped addresses enabled)
 -D APR_USE_SYSVSEM_SERIALIZE
 -D APR_USE_PTHREAD_SERIALIZE
 -D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
 -D APR_HAS_OTHER_CHILD
 -D AP_HAVE_RELIABLE_PIPED_LOGS
 -D DYNAMIC_MODULE_LIMIT=128
 -D HTTPD_ROOT="/etc/apache2"
 -D SUEXEC_BIN="/usr/lib/apache2/suexec"
 -D DEFAULT_PIDLOG="/var/run/apache2.pid"
 -D DEFAULT_SCOREBOARD="logs/apache_runtime_status"
 -D DEFAULT_LOCKFILE="/var/run/apache2/accept.lock"
 -D DEFAULT_ERRORLOG="logs/error_log"
 -D AP_TYPES_CONFIG_FILE="mime.types"
 -D SERVER_CONFIG_FILE="apache2.conf"
```

Oltre all'invocazione diretta del comando o all'uso dello script di avvio, il pacchetto di Apache fornisce un programma di controllo apposito, `apachectl`, che permette di gestire, a server avviato, le varie operazioni.

Comando	Effetto
`start`	Avvia il servizio.
`stop`	Ferma il servizio.
`restart`	Avvia il servizio se non attivo, altrimenti causa la rilettura della configurazione con l'invio di un `SIGHUP`.
`status`	Stampa un breve rapporto sullo stato del server. Richiede la presenza del browser testuale `lynx` e l'attivazione di `mod_status`.
`fullstatus`	Stampa un rapporto più esteso sullo stato del server. Richiede la presenza del browser testuale `lynx` e l'attivazione di `mod_status`.
`graceful`	Riavvia il servizio attendendo la conclusione di tutte le connessioni pendenti.
`configtest`	Esegue un controllo della configurazione. Si tenga conto che viene solo controllata la sintassi formale.

Tabella 1.11: Valori dei comandi di controllo disponibili con `apachectl`.

In genere il comando si usa per avviare e fermare il server ma rispetto all'invocazione diretta fornisce alcune funzionalità ulteriori, permettendo ad esempio di riavviare il server completando prima di fermarlo le connessioni correnti o controllando preventivamente la configurazione. L'elenco dei comandi di controllo da passare al programma si ottiene con direttamente con `apachectl help`, ed il rispettivo significato è illustrato in tab. 1.11.

Il comando `apachectl` inoltre può essere utilizzato per eseguire direttamente il server, senza preoccuparsi del nome del binario con è stato installato, se gli si passa una qualunque delle opzioni dello stesso, pertanto le opzioni di tab. 1.10 valgono anche per `apachectl`.

1.3 La configurazione delle funzionalità di base

Esamineremo in maggiore profondità in questa sezione la configurazione delle funzionalità di base di Apache, trattando in dettaglio il formato del file di configurazione ed esaminando le direttive usate per controllare le principali caratteristiche del server ed alcune delle funzionalità utilizzate più comunemente.

1.3.1 La sintassi e le direttive fondamentali

La sintassi del file di configurazione di Apache è leggermente più complessa rispetto a quella di un normale file di configurazione, il programma infatti supporta diversi tipi di direttive. La prima classificazione è fra direttive "*semplici*" ed i cosiddetti "*container*". Una direttiva semplice ha una forma del tipo:

```
DirettivaSemplice1 valore
DirettivaSemplice2 valore2 altrovalore2
```

e consiste in una riga in cui si specifica il nome della direttiva, a cui seguono, separati da uno o più spazi, uno o più valori. Se una riga non è sufficiente si può utilizzare come ultimo carattere della stessa una "\" per proseguire la lista dei valori sulla successiva; si faccia attenzione che la "\" non sia seguita da caratteri vuoti come spazi o tabulatori, perché in tal caso la prosecuzione alla riga successiva fallirebbe.

I nomi delle direttive sono *case insensitive*, ma vengono usualmente scritti, per aumentarne la leggibilità, nel cosiddetto *camel case*, seguendo la convenzione in cui le singole parole che compongono il nome sono scritte in maiuscolo, come nell'esempio precedente; i parametri delle direttive invece sono spesso *case sensitive*.

Come accennato oltre alle direttive semplici esistono i cosiddetti *container* che sono delle direttive speciali che permettono di definire dei "*contenitori*" per delle altre direttive, in modo che sia possibile applicare queste ultime ad una classe di oggetti nel suo insieme; in tal caso la sintassi cambia ed assume una forma simile all'HTML con qualcosa del tipo:

```
<DirettivaContenitore argomento>
  SottoDirettiva1 valore1
  SottoDirettiva2 valore2
  SottoDirettiva3 valore3 altrovalore3
</DirettivaContenitore>
```

si ha un marcatore per segnare l'inizio del *contenitore* ed un altro per chiuderlo, ed all'interno una serie di altre direttive che si applicano agli oggetti identificati dallo stesso. Infine le righe vuote, e quelle inizianti per "#", che sono considerate commenti, vengono ignorate.

Le direttive semplici poi vengono a loro volta suddivise in tre categorie, quelle generali (o *server-level*), che controllano le caratteristiche del server in quanto tale, e che pertanto possono essere usate soltanto nella configurazione generica dello stesso; quelle *locali* che specificano funzionalità che hanno senso solo all'interno degli oggetti raccolti in un *container* (non avendo significato per il server in quanto tale) e quelle *locali e globali* che se specificate in un *container* si applicano solo agli oggetti cui questo fa riferimento, mentre al di fuori assumono il significato di valore di default usato dal server.

In tab. 1.12 si sono elencate le direttive standard che definiscono dei *container*. Si ricordi che ciascuna di essa prende un argomento che definisce gli oggetti a cui le regole inserite all'interno del *container* verranno applicate. Si tenga presente inoltre che alcune di queste direttive possono essere annidate una all'interno dell'altra.

Container	**Descrizione**
`Directory`	definisce una directory ed il sottostante ramo dell'albero dei file cui si applicano le direttive specificate; prende come argomento il pathname della directory in questione (ad esempio `/var/www/GaPiL`) ed accetta anche i caratteri jolly del *filename globbing*.
`DirectoryMatch`	analoga alla precedente, ma permette di specificare un insieme di directory attraverso una espressione regolare (ad esempio con qualcosa come `/var/www/www.*.(com|it)`).
`Files`	simile a directory, ma serve a selezionare dei file singoli; anche in questo caso accetta come argomento una stringa con i caratteri jolly del *filename globbing*.
`FilesMatch`	analoga alla precedente ma permette di specificare un insieme di file usando una espressione regolare.
`Location`	definisce un insieme di risorse cui applicare le direttive; è simile a `Directory`, ma opera nello spazio dei nomi delle URI richieste dal client, che non è assolutamente detto debbano corrispondere ad un qualche file presente su disco; in questo caso l'argomento specifica una sezione in forma di *pathname*, e supporta l'uso dei caratteri jolly.
`LocationMatch`	analoga a `Location` ma permette l'uso di espressioni regolari per la selezione.
`Limit`	permette di restringere l'effetto delle direttive di controllo di accesso ai soli metodi HTTP citati nella lista passata come argomento; la lista deve essere specificata con i nomi dei metodi riportati in tab. 1.2 e tab. 1.3 separati da spazi, con qualcosa del tipo di `<Limit POST PUT DELETE>`.
`LimitExcept`	permette di restringere l'effetto delle direttive di controllo di accesso ai metodi HTTP che non rientrano nella lista specificata come argomento.
`VirtualHost`	definisce un *dominio virtuale* (argomento trattato in sez. 1.3.3) a cui si applicano le direttive specificate.

Tabella 1.12: Le direttive che definiscono i *container* nella configurazione di Apache.

Le due direttive `DirectoryMatch` e `FilesMatch` sono in genere relativamente poco usate in quanto è possibile usare le espressioni regolari anche direttamente all'interno di una direttiva `Directory` o di una direttiva `Files` se il loro argomento viene introdotto dal carattere "~", ad esempio specificando una direttiva:

```
<Directory ~ /var/www/www.*.(com|it)>
```

è sostanzialmente equivalente alla direttiva:

```
<DirectoryMatch  /var/www/www.*.(com|it)>
```

Oltre alle vere e proprio direttive *container* di tab. 1.12 esistono altre due direttive che seguono una sintassi dei marcatori analoga, ma da considerare a parte in quanto esse non servono affatto a definire dei *contenitori* per altre direttive quanto piuttosto a permettere di gestire in maniera più flessibile la configurazione attraverso l'uso di blocchi condizionali che attivano o disattivano delle funzionalità. Queste direttive sono valutate solo all'avvio del server, nel senso che l'esito delle configurazioni applicate al loro interno è deciso all'avvio del server, mentre l'esito delle configurazioni specificate con uno dei container di tab. 1.12 viene deciso per ogni richiesta corrispondente.

La prima di queste direttive è `IfModule`, che consente di eseguire le regole di configurazione inserite al suo interno qualora il modulo (torneremo su questa funzionalità in sez. 1.4.1) da essa specificato sia attivo. Dato che molti moduli implementano delle funzionalità controllate da direttive specifiche che sono riconosciute dal server solo quando il modulo è attivo, l'uso di `IfModule` consente di utilizzare lo stesso file di configurazione anche se il modulo non è attivo, mantenendo all'interno di uno blocco condizionale tutte le direttive di configurazione relative al modulo stesso; in questo modo se esso non è attivo le direttive non saranno considerate e non avranno errori.

La direttiva prende come argomento il nome del modulo indicato dal nome del file che ne contiene il codice, come indicato nell'output del comando `apachectl` l'opzione `-l` (vedi sez. 1.2.3). Un esempio di questa direttiva condizionale è il seguente:

```
<IfModule mod_status.c>
    ExtendedStatus On
</IfModule>
```

La seconda direttiva è `<IfDefine>` (presente solo a partire dalla versione 1.3.1 del server) che è più propriamente utilizzata per realizzare delle configurazioni con sezioni opzionali; essa utilizza i valori dei parametri di configurazione che vengono passati al server sulla riga di comando con l'opzione `-D` (vedi tab. 1.10), pertanto si potranno inserire nella configurazione delle sezioni come:

```
<IfDefine DEBUG>
    LogLevel debug
</IfDefine>
```

che saranno utilizzate solo qualora si sia avviato il server invocandolo passando il parametro `DEBUG` all'opzione `-D`.

Infine c'è una direttiva specifica, `Include`, che attiene direttamente alla gestione dei file di configurazione, e consente, come indica il nome di includere il contenuto di altri file all'interno di un file di configurazione. In questo modo diventa possibile suddividere la configurazione delle varie funzionalità in parti diverse, distribuendole su file diversi, in maniera da facilitarne la gestione.

La direttiva prende come argomento il nome di un file, il cui contenuto sarà letto come se fosse stato scritto al posto della direttiva. Si possono anche specificare gruppi di file utilizzando la sintassi dei caratteri *jolly* del *filename globbing* della shell (disponibile a partire da Apache 1.3.27) nel qual caso saranno inclusi nell'ordine di corrispondenza. Come esempio di uso della direttiva consideriamo il seguente estratto, preso dalla configurazione di Apache installata di default su una Debian Squeeze:

apache2.conf

```
...
# Include module configuration:
Include mods-enabled/*.load
Include mods-enabled/*.conf

# Include all the user configurations:
Include httpd.conf

# Include ports listing
Include ports.conf
...
# Include generic snippets of statements
Include conf.d/

# Include the virtual host configurations:
Include sites-enabled/
```

A partire dalla versione 1.3.13 del server la direttiva `Include` consente anche di eseguire l'inclusione di una directory, nel qual caso Apache eseguirà la scansione della stessa e di tutte le sue sottodirectory includendo automaticamente ogni file che viene trovato. In questo modo diventa possibile gestire in maniera estremamente comoda configurazioni in situazioni in cui ci sono tante sezioni indipendenti inserendo un piccolo file per ciascuna di esse; un esempio è riportato nell'ultima riga dell'estratto precedente.

La prima parte dell'esempio riporta le direttive utilizzate nella parte iniziale del file di configurazione: prima viene usato il contenuto dei file `.conf` e `.load` presenti nella directory `/etc/apache2/mods-enabled/` per indicare quali moduli (vedi sez. 1.4.1) caricare poi si usa un eventuale contenuto di configurazioni "*legacy*" presente in `httpd.conf` ed il file `ports.conf` per definire su quali porte mettere in ascolto il server, infine si leggono eventuali configurazioni particolari presenti in `conf.d`, dove in genere si mettono le direttive di configurazione specifiche delle varie applicazioni web. L'ultima parte dell'estratto riporta le ultime righe di `apache2.conf`, dove si includono le configurazioni dei vari *Virtual Host* (vedi sez. 1.3.3).

1.3.2 La configurazione base del server

Come accennato in sez. 1.3.1 alcune direttive di configurazione sono applicabili solo a livello globale dell'intero server e consentono di impostarne le funzionalità generali. È a questo livello che ci sono state le maggiori differenze fra la versione 1 e la versione 2 di Apache, infatti con l'introduzione degli MPM, alcune di quelle che nella versione 1 erano direttive generali del server, sono diventate direttive specifiche del modulo `prefork`.

In generale la prima cosa che si deve fare per una configurazione di base di un server web gestito con Apache è impostare alcuni parametri fondamentali relativi al funzionamento del servizio e cioè:

- la porta, ed eventualmente l'indirizzo IP (se se ne usa più di uno) su cui mettere in ascolto il server.
- l'utente ed il gruppo con cui eseguire i processi del server.
- la directory di lavoro ed i file ausiliari

- le caratteristiche con cui il server gestisce le transazioni HTTP
- le modalità generali di funzionamento del server

Alcuni di questi parametri sono completamente generici, e sono perciò controllati da direttive che si applicano in maniera generale, mentre altri, in particolare quelli che riguardano le modalità con cui Apache fornisce le sue risposte ai client, dipendono dall'MPM scelto, ed in tal caso le direttive disponibili variano a seconda di quest'ultimo.

Nel caso di Debian inoltre (ma la politica è adottata anche da altre distribuzioni) una serie di impostazioni, tradizionalmente controllate dal file di configurazione con specifiche direttive, vengono demandate all'uso di alcune variabili di ambiente, che possono essere a loro volta riutilizzate all'interno del file di configurazione, utilizzando la forma `${VARIABILE}` analoga alla espansione fatta dalla shell. In particolare queste variabili sono definite nel file `/etc/apache2/envvars` e consentono di impostare alcune delle proprietà globali citate in precedenza sostituendosi alle rispettive direttive. Se ne è riportato un elenco in tab. 1.13.

Variabile	Significato
`APACHE_CONFDIR`	directory delle configurazioni (equivalente alla direttiva `ServerRoot`).
`APACHE_RUN_USER`	utente per conto del quale si esegue il programma (equivalente alla direttiva `User`).
`APACHE_RUN_GROUP`	gruppo per conto del quale si esegue il programma (equivalente alla direttiva `Group`).
`APACHE_PID_FILE`	file dove salvare il PID del processo principale.
`APACHE_LOCK_DIR`	directory dove salvare il file di lock.
`APACHE_LOG_DIR`	directory dove salvare i file di registrazione degli accessi e degli errori.

Tabella 1.13: Le variabili di ambiente in `envvars`.

Fra le direttive di base che controllano il funzionamento generale di Apache la più rilevante è probabilmente `Listen`, che controlla le modalità con cui Apache 2 si mette in ascolto sulla rete. La direttiva permette di indicare in una unica soluzione sia l'indirizzo che la porta su cui il server accetterà le connessioni. Specificando solo un numero si indica la porta, mettendosi in ascolto sull'indirizzo generico. La forma completa è però quella in cui si può specificare anche un indirizzo IP specifico insieme alla porta, con un argomento nella forma "`indirizzo:porta`". Nel caso di Debian tutte le direttive `Listen` vengono raccolte in un file separato, `/etc/apache2/ports.conf`, che poi viene incluso dal file di configurazione generale.

Insieme a `Listen` un primo gruppo di direttive generiche che controllano il funzionamento del server a livello globale sono quelle che permettono di specificare la collocazione dei vari file di cui il server ha bisogno o l'utente ed il gruppo per conto del quale vengono eseguiti i processi o le caratteristiche generali delle connessioni di rete; un elenco delle più significative è il seguente:

`User` indica l'utente per conto del quale devono essere eseguiti i processi del che rispondono alle richieste. Apache infatti viene usualmente eseguito dall'amministratore,[13] ma una volta compiute le operazioni di inizializzazione tutti i successivi processi verranno eseguiti per conto dell'utente specificato da questa direttiva. Per Debian è `www-data`, per RedHat è `httpd` e per SuSE `wwwrun`. Con l'uso delle variabili di configurazione di tab. 1.13 viene ignorata.

[13] se così non fosse questa direttiva fallirà, non avendo un utente normale la capacità di cambiare il titolare di un processo, ed il server continuerà ad essere eseguito per conto dell'utente che l'ha lanciato.

Group analoga alla precedente, indica il gruppo per conto del quale devono essere eseguiti i processi che rispondono alle richieste. Anche in questo caso per Debian è `www-data`, per RedHat è `httpd`, per SuSE `www`. Con l'uso delle variabili di configurazione di tab. 1.13 viene ignorata.

ServerRoot indica la directory di lavoro del server, e costituisce in pratica la directory in cui sono mantenuti i file di configurazione. Nel caso di Debian per Apache 1.3 era `/etc/apache` (come nell'esempio precedente) mentre per Apache 2 è `/etc/apache2`. Tutti i pathname relativi usati nelle configurazioni vengono risolti a partire da questa directory. Con l'uso delle variabili di configurazione di tab. 1.13 non è più necessaria.

CoreDumpDirectory indica la directory dove scrivere i *core dump*. Di default questa sarebbe la directory indicata da `ServerRoot` ma siccome quest'ultima usualmente è sotto `/etc` non sarà scrivibile da parte del demone una volta che questo ha ceduto i privilegi di amministratore; per cui se si vuole utilizzare questa funzionalità (in genere lo si fa a scopo di debug) occorre specificare una directory diversa. Su Linux se Apache è avviato da `root` i *core dump* vengono sempre disabilitati, solo la presenza di questa direttiva consente di riabilitarli.

Lockfile indica il nome del file di lock usato dal server, il valore di default sarebbe `logs/accept.lock`, ma in genere le configurazioni installate dalle varie distribuzioni preferiscono spostare i file su `/var/lock` come richiesto dal *Filesystem Hyerarchy Standard*; Debian ad esempio usa `/var/lock/apache2/accept.lock`.

PidFile indica il file in cui viene memorizzato il PID del processo principale di Apache; come per il file di lock il default sarebbe `logs/httpd.pid` che sta sotto `/etc`, per cui al solito si tende a spostarlo (come richiesto anche dal *Filesystem Hyerarchy Standard*) sotto `/var/run`; nel caso di Debian la distribuzione imposta questo valore nel file di configurazione standard a `/var/run/apache2.pid`.

ListenBackLog indica la profondità della coda delle connessioni che sono mantenute in attesa di un processo legga i loro dati.[14] In genere la si mantiene al valore di default, se non in caso di un *SYN flood*[15] in cui può essere il caso di aumentarne il valore.

SendBufferSize permette di impostare un'altra delle caratteristiche delle connessioni TCP, cioè le dimensioni del buffer usato per i dati delle connessioni; ha senso solo in casi particolari come le linee ad alta velocità ed alta latenza, in cui conviene appunto inviare il maggior numero di dati alla volta.

Un secondo gruppo di direttive è quello che consente di configurare le caratteristiche del comportamento del server nella gestione delle transazioni HTTP che a differenza delle precedenti

[14] corrisponde ad una caratteristica del funzionamento dell'interfaccia dei socket TCP, per una trattazione dell'argomento si può consultare la sezione 15.2.3 di [GaPiL].

[15] è un tipico attacco di *Denial of Service*, in cui vengono create connessioni fittizie cercando appunto di riempire questa coda.

possono essere usate anche a livello di singolo *dominio virtuale*, anche se questo non viene quasi mai fatto. Un elenco delle più rilevanti è il seguente:

KeepAlive abilita la funzionalità di mantenere attive le connessioni TCP in modo da avere la possibilità di ricevere su di esse richieste multiple (questo può comportare notevoli miglioramenti nelle prestazioni con pagine web con molte immagini), l'argomento prende i valori `On` e `Off` ed è attivo di default.

KeepAliveTimeout
qualora si sia attivata la precedente `KeepAlive` si può indicare con questa direttiva il numero di secondi per cui il server aspetta per l'arrivo di un'altra richiesta prima di chiudere la connessione. Tenerlo troppo alto può essere nocivo per le prestazioni in caso di traffico elevato dato che potrebbero esserci parecchi processi inutilmente occupati.

MaxKeepAliveRequests
indica il numero massimo di richieste che si consentono all'interno di una stessa connessione. Un numero nullo significa nessun limite, il default è 100 e conviene comunque usare un valore elevato (il default di Debian ad esempio è 150).

TimeOut indica una serie di tempi di scadenza per le varie operazioni di rete, come quello trascorso prima di ricevere una richiesta con `GET`, quello fra di ricezione dei dati nelle richieste `PUT` e `POST`, o quello della ricezione dei pacchetti TCP di ricevuto sulle risposte. Il valore di default è 300 e viene in genere lasciato immutato.

Un terzo gruppo di direttive è quello che permette di impostare delle limitazioni sui servizi offerti, in particolare le direttive `LimitRequest*` permettono di limitare le risorse usate dalle richieste dei client e possono essere utili per mitigare eventuali attacchi di *Denial of Service* o rilevare comportamenti anomali. Le direttive `RLimit*` invece permettono di imporre dei limiti[16] ai processi creati da Apache consentendo di controllare la quantità di risorse utilizzate da programmi invocati dagli stessi (come i CGI, vedi sez. 1.5.2).

LimitRequestBody
indica il numero di byte massimo consentiti all'interno di una richiesta. Il default è 0, che significa nessun limite ed il massimo è di 2GiB.

LimitRequestFields
indica il numero massimo di intestazioni che è possibile avere in una richiesta. Al solito 0 significa illimitato, il default è di 100.

LimitRequestFieldsize
indica la dimensione massima che possono avere le intestazioni. Al solito 0 significa illimitato, il default è di 8190.

[16] per farlo vengono usate le funzionalità di limitazione delle risorse che il sistema stesso mette a disposizione; l'argomento è trattato in dettaglio nella sezione 8.3 di [GaPiL].

LimitRequestLine indica la dimensione massima di una riga di richiesta (quella di fig. 1.2) cioè in sostanza della URI in essa specificata.

RLimitCPU impone un limite sul tempo di CPU, espresso in secondi per processo, che può essere utilizzato da tutti i processi creati da uno dei processi figli di Apache nel soddisfare una richiesta.

RLimitMEM impone un limite sulla quantità di memoria, espressa in byte per processo, che può essere utilizzato da tutti i processi creati da uno dei processi figli di Apache nel soddisfare una richiesta.

RLimitNPROC impone un limite sul numero di processi che possono essere creati da uno processi lanciato dai processi figli di Apache nel soddisfare una richiesta.

Infine abbiamo lasciato per ultime fra quelle globali le direttive con cui si gestiscono la modalità con cui il server crea i vari processi e come questi rispondono alle richieste, perché, come accennato all'inizio, queste dipendono dall'MPM che si utilizza. Abbiamo accennato in sez. 1.2.3 come all'avvio il server legga il file di configurazione, apra i file di log e dopo essersi posto in ascolto sulla rete mandi in esecuzione un certo numero di processi figli.

Questo è il comportamento dell'ormai storico Apache 1.3 e dell'MPM `prefork` di Apache 2 che usano una tecnica di gestione delle connessioni dalla rete che si chiama *pre-forking*: invece di creare processi figli in risposta ad ogni singola connessione da parte dei client, un certo numero di questi vengono creati preventivamente in modo che siano già pronti a rispondere senza introdurre il ritardo relativo alla loro creazione.

Il server inoltre è in grado di aumentare e diminuire a seconda del carico il numero di questi processi, e anche queste funzionalità sono controllate da direttive di questo gruppo. Dato che con Apache 2 queste direttive sono disponibili solo nel caso si stia usando l'MPM `prefork` esse sono usualmente protette da una appropriata direttiva condizionale di tipo `IfModule`. L'elenco delle più significative è il seguente:

MaxClients indica il numero massimo di richieste contemporanee che il server può soddisfare, significa in pratica che il server non metterà mai in esecuzione più processi del numero specificato da questa direttiva (il default è 256, ma tutte le distribuzioni usano un valore inferiore, ad esempio Debian usa 150). Tutte le richieste che eccedono questo limite vengono accodate (si veda la successiva `ListenBacklog`) e saranno esaudite non appena uno dei processi del server diventa disponibile.

Il valore da assegnare dipende in sostanza dalla quantità di RAM disponibile, un valore troppo alto infatti può esaurire la RAM causando l'uso della swap, peggiorando così terribilmente le tutte prestazioni del server (che sarà impegnato a spostare dati da e verso il disco, invece che a inviarli sulla rete). Questa situazione deve essere assolutamente evitata; se ci sono troppe richieste rispetto al numero dei processi è preferibile farle accodare, in modo da avere comunque il server impegnato nell'invio dei dati sulla rete per quelle che è in grado di gestire. Si può calcolare un valore ragionevole per questo

parametro andando a dividere la RAM totale per la dimensione media dei processi di Apache (da esaminare con un qualunque programma come `top`).

StartServers indica il numero di processi figli che vengono lanciati all'avvio, il valore di default è 5, e dato che il numero dei processi viene aggiustato dinamicamente (aumentando se ci sono molte richieste) non ci sono motivi significativi per modificarlo, se non quella, qualora si abbia già un'idea del carico e questo sia molto più alto, di evitare di perdere tempo nel periodo iniziale di aggiustamento dopo l'avvio del server.

MinSpareServers indica il numero *minimo* di processi in più rispetto a quelli già attivi che devono restare fermi in attesa di connessione; quando il numero corrente è inferiore a questo limite il server inizia a creare nuovi processi (così da poterli utilizzare per una risposta immediata). Il valore di default è 5, in genere non è necessario modificarlo, se non per siti molto trafficati con rapide variazioni nei picchi di traffico. Un valore molto alto è sempre sconsigliabile (si tiene occupata inutilmente della memoria).

Originariamente questo avveniva ad una frequenza massima di uno al secondo, con siti con molto traffico (e molte connessioni simultanee) questo poteva comportare tempi di reazione all'avvio del server piuttosto lenti, per risolvere il problema con Apache 1.3 il numero di nuovi processi creati viene raddoppiato tutte le volte che la creazione non supera il limite (fino ad un massimo di 32 al secondo).

MaxSpareServers indica il numero *massimo* di processi fermi in attesa di connessione, quando si supera questo valore i processi in eccesso vengono terminati. Il valore di default è 15 e di nuovo, se non nel caso di siti con molto traffico, non è il caso di modificarlo. Anche in questo caso non è mai il caso di tenerlo su un valore molto alto.

MaxRequestsPerChild indica il numero massimo di richieste che un processo deve servire, dopo le quali viene terminato. Il valore di default è 0, che significa un numero illimitato. L'utilizzo può essere utile in caso di perdite di memoria (caso piuttosto raro, a meno di non usare moduli sperimentali), e per ridurre più velocemente il numero di processi quando il carico diminuisce.

In generale queste direttive, che servono ad impostare le funzionalità generali del server, vengono scritte all'inizio del file di configurazione, un esempio del loro utilizzo, stavolta estratto dalla configurazione di default di Apache 2.2 installata da Debian Squeeze, è il seguente:

apache2.conf

```
ServerRoot "/etc/apache2"
LockFile ${APACHE_LOCK_DIR}/accept.lock
PidFile ${APACHE_PID_FILE}
Timeout 300
KeepAlive On
MaxKeepAliveRequests 100
KeepAliveTimeout 15
```

```
<IfModule mpm_prefork_module>
  StartServers         5
  MinSpareServers      5
  MaxSpareServers     10
  MaxClients          20
  MaxRequestsPerChild  0
</IfModule>
...
```

dove compaiono (con l'eccezione della sezione riguardante l'uso dei moduli, che tratteremo in sez. 1.4.1) soltanto direttive di tipo globale applicabili al server nel suo insieme.

Altri parametri di base necessari al funzionamento del servizio sono invece quelli attinenti alla modalità di gestione dei singoli siti. Questi si possono anche indicare a livello globale ma in tal caso le rispettive direttive consentiranno solo di impostare un default da applicarsi in maniera generica. A questa seconda categoria appartengono parametri come:

- il nome a dominio con cui si contatta il server (se ci sono più domini si dovranno usare i *Virtual Host* che tratteremo in sez. 1.3.3).
- la directory contenente le pagine HTML e le altre risorse messe a disposizione del server.
- l'indirizzo di posta dell'amministratore del server (questo non è obbligatorio ma è sempre bene definirlo).

Ancorché come accennato sia possibile impostare questi parametri a livello globale, normalmente essi vengono impostati singolarmente sito per sito all'interno dei cosiddetti *domini virtuali* (i *Virtual Host*), che tratteremo estesamente in sez. 1.3.3. Tratteremo comunque in questa sezione le direttive che li riguardano, dato che si tratta comunque di parametri attinenti la configurazione di base del servizio.

Fra queste la più rilevante è senza dubbio `DocumentRoot` che indica la directory radice delle pagine web, cioè la directory in cui dovranno trovarsi i file che vengono restituiti in risposta alle URI delle richieste (vedi sez. 1.1.2). In sostanza una URI viene tradotta in un *pathname* che viene risolto sul filesystem usando come radice la directory indicata da `DocumentRoot`. In genere questa direttiva viene usata all'interno di un *container* di tipo `VirtualHost` per indicare la radice dei dati per quel *dominio virtuale*.

Un gruppo di direttive di base relative alla configurazione di un sito, e che come accennato pur essendo impiegabili anche a livello generale vengono quasi sempre usate all'interno dei singoli *domini virtuali*, è quello delle direttive di identificazione. Un elenco delle principali è il seguente:

`ServerName` permette di impostare il nome a dominio (ed eventualmente la porta) che il server utilizza per identificare se stesso, cioè quello che poi compare nelle URL che vengono generate in caso di redirezione. Questo consente ad esempio di far identificare il server web con il nome a dominio specificato tramite questa direttiva, anche quando il suo indirizzo viene risolto altrimenti; occorrerà ovviamente che il nome specificato abbia avere una risoluzione valida sul DNS.[17]

[17] qualora la direttiva non sia specificata Apache cercherà di dedurre il nome con una risoluzione inversa dell'indirizzo IP.

Quando questa direttiva viene usata all'interno di un *container* di tipo `VirtualHost` essa serve a selezionare qual'è il nome a dominio, che come accennato in sez. 1.1.3 compare nell'intestazione `Host` della richiesta, con cui viene identificato quel *dominio virtuale*; torneremo su questo argomento in sez. 1.3.3.

`ServerAdmin` indica l'indirizzo di posta elettronica del gestore del servizio web, che verrà incluso nelle pagine di errore mostrate ai client. Anche questa direttiva può essere inserita in un *container* di tipo `VirtualHost`.

`ServerAlias` consente di impostare una serie di nomi a dominio alternativi ulteriori rispetto a quello indicato da `ServerName`, viene usata principalmente nella gestione dei *domini virtuali* (vedi sez. 1.3.3) all'intero di direttive *container* `VirtualHost`. La direttiva prevede l'indicazione di un elenco di nomi a dominio separati da spazi, e supporta l'uso dei carattere jolly "`?`" e "`*`".

`UseCanonicalName`
permette al server di ricostruire automaticamente dei valori delle URL quando deve fare riferimento a se stesso, utilizzando base dei valori specificati per `ServerName` e per `Listen`.

1.3.3 I *domini virtuali* o *Virtual Host*

Il termine *Virtual Host* (o *dominio virtuale*) indica la possibilità di gestire più siti su uno stesso server, distinguendoli solo in base all'indirizzo a cui è rivolta la richiesta eseguita dal client, e non per un diverso percorso nell'albero dei contenuti. Si tenga conto che in questo caso per indirizzo si intende la parte centrale della URL (illustrata in fig. 1.1), che può essere sia un indirizzo IP, che un nome a dominio. Questa funzionalità consente di mantenere più siti web, completamente diversi e scorrelati fra loro, attraverso una unica istanza di Apache.

Tradizionalmente i domini virtuali sono classificati in *IP-based* e *name-based*. Il primo tipo funziona semplicemente assegnando contenuti diversi tramite diversi numeri IP assegnati alla stessa macchina; Apache è stato uno dei primi server a supportare questa modalità di funzionamento, che però comporta la scomodità (ed il costo) di dover utilizzare più IP.

Nel secondo caso invece si possono avere diversi domini virtuali sullo stesso indirizzo IP, in quanto questi vengono identificati solo sulla base del nome a dominio con cui il client esegue la richiesta. Per questo però è necessario il supporto da parte del client del protocollo `HTTP/1.1` in quanto per realizzare questa funzionalità Apache utilizza l'intestazione `Host` del protocollo (vedi sez. 1.1.3) che deve essere fornita dal client. Oggi questo non è più un problema in quanto ormai tutti i browser supportano il protocollo `HTTP/1.1`, ed in genere quando si parla di *Virtual Host* si intende sempre che sia di tipo *name-based*.

Come accennato in sez. 1.3.1 per realizzare un dominio virtuale Apache mette a disposizione una apposita direttiva *container*: `VirtualHost`. Questa prende come argomento l'indirizzo a cui si fa riferimento per le richieste indirizzate verso detto dominio, che può essere espresso sia in forma numerica che con un nome a dominio, in questo caso però, se si usano i domini virtuali *IP-based*, occorre essere sicuri della corretta risoluzione.

Si possono anche usare più indirizzi IP qualora il dominio faccia riferimento a più indirizzi (ad esempio un IP pubblico per l'esterno ed uno privato per l'accesso dalla LAN). In tutti i

casi con l'indirizzo si può specificare nella forma `indirizzo:porta`, e se non si indica la porta viene assunto come valore predefinito la porta 80. Esiste poi anche il valore riservato `_default_` che corrisponde ad ogni indirizzo che non è stato specificato esplicitamente in altre direttive `VirtualHost`; se questo manca per questi indirizzi verrà utilizzata la configurazione generale del server.

Per tutte le richieste relative a ciascun dominio virtuale saranno applicate le direttive di configurazione definite all'interno del rispettivo *container*; in genere un `VirtualHost` deve contenere almeno una direttiva `ServerName` che definisce il nome del dominio e una direttiva `DocumentRoot` che definisce la radice dell'albero dei contenuti dello stesso. In genere poi si utilizzano anche le varie direttive di sez. 1.4.3 per mantenere separati anche i file di log di ciascun dominio virtuale.

Definire un dominio virtuale *IP-based* è molto semplice, occorre semplicemente che il DNS risolva i diversi domini a diversi indirizzi IP e che si sia specificato l'indirizzo su cui Apache si mette in ascolto con la direttiva `Listen` in modo da essere sicuri che il server sia in ascolto su quello specifico indirizzo. Una configurazione tipica, assunto che `www.truelite.it` e `www.fountainpen.it` risolvano a due indirizzi IP diversi assegnati alla stessa macchina che ospita Apache, sarà qualcosa del tipo:

ip-based-vhost.conf

```
Listen www.truelite.it:80
Listen www.fountainpen.it:80
...
<VirtualHost www.truelite.it:80>
    ServerAdmin webmaster@truelite.it
    DocumentRoot /var/www/truelite.it
    ServerName www.truelite.it
    ErrorLog logs/truelite-error.log
    CustomLog logs/truelite-access.log combined
</VirtualHost>
<VirtualHost www.fountainpen.it:80>
    ServerAdmin webmaster@fountainpen.it
    DocumentRoot /var/www/fountainpen.it
    ServerName www.fountainpen.it
    ErrorLog logs/fountainpen-error.log
    CustomLog logs/fountainpen-access.log combined
</VirtualHost>
```

Come accennato oggi i *Virtual Host IP-based* non vengono più utilizzati in quanto richiedono, senza che ormai ve ne sia più la necessità, di avere un indirizzo IP dedicato ad ogni singolo sito distinto. Questo non vale per *Virtual Host name-based* in cui l'identificazione viene effettuata solo in base al nome a dominio, cosa che rende anche più semplice la configurazione.

Se si vuole attivare un dominio virtuale *name-based* occorre però usare un'altra direttiva, `NameVirtualHost`, per indicare ad Apache che se ne vuole fare uso. La direttiva richiede come argomento l'indirizzo IP e la porta su cui saranno ricevute le connessioni per i domini virtuali *name-based*.

Se si hanno più indirizzi IP basterà ripetere la direttiva per ciascuno di essi; ma è molto più comune usare come indirizzo il valore "∗" che indica l'indirizzo IP generico e corrisponde ad qualunque indirizzo presente sulla macchina che non sia già stato usato da una più specifica configurazione precedente (sia con `NameVirtualHost` che in un altro `VirtualHost` *IP-based*).

In questo caso in genere si ha una sola indicazione di `NameVirtualHost`, ad esempio nel caso di Debian questa direttiva viene predefinita insieme alla dichiarazione delle porte cui il server si pone in ascolto nel file `ports.conf`, installato dalla distribuzione insieme al pacchetto, di cui riportiamo di seguito un estratto della versione fornita con Debian Squeeze:

ports.conf

```
NameVirtualHost *:80
Listen 80

<IfModule mod_ssl.c>
    # If you add NameVirtualHost *:443 here, you will also have to change
    # the VirtualHost statement in /etc/apache2/sites-available/default-ssl
    # to <VirtualHost *:443>
    # Server Name Indication for SSL named virtual hosts is currently not
    # supported by MSIE on Windows XP.
    NameVirtualHost *:443
    Listen 443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 443
</IfModule>
```

Una volta che si è usata `NameVirtualHost` tutto quello che resta da fare è definire per ciascuno dei domini virtuali che si vogliono utilizzare uno specifico *container* con la direttiva `VirtualHost` usando lo stesso argomento utilizzato da `NameVirtualHost`, che questo sia uno specifico indirizzo IP o il valore "*" dell'indirizzo generico.

Tutti i *container* `Virtual Host` dovranno contenere al loro interno una opportuna direttiva `ServerName` che indichi qual'è il nome a dominio associato al dominio virtuale. Un esempio di configurazione di domini virtuali *name-based* per `www.truelite.it` e `www.fountainpen.it` corrispondente alla configurazione di `NameVirtualHost` adottata nell'esempio precedente è il seguente:

name-based-vhost.conf

```
<VirtualHost *:80>
    ServerAdmin webmaster@truelite.it
    DocumentRoot /var/www/truelite.it
    ServerName www.truelite.it
    ServerAlias truelite.it
    ErrorLog logs/truelite.it-error.log
    CustomLog logs/truelite.it-access.log combined
</VirtualHost>
<VirtualHost *:80>
    ServerAdmin webmaster@fountainpen.it
    DocumentRoot /var/www/fountainpen.it
    ServerName www.fountainpen.it
    ServerAlias fountainpen.it
    ErrorLog logs/fountainpen.it-error.log
    CustomLog logs/fountainpen.it-access.log combined
</VirtualHost>
```

In questo caso il server esamina tutte le richieste che arrivano alla macchina sulla porta 80 (si è infatti specificato `NameVirtualHost` sull'indirizzo generico) e confronta l'intestazione `Host`

inviata dal client con il valore riportato da `ServerName` in modo da restituire le pagine ad esso relative con la rispettiva `DocumentRoot`.

Nell'esempio si può notare anche l'uso della direttiva `ServerAlias`, che consente, come accennato in sez. 1.3.2, di definire un insieme di nomi alternativi da utilizzare per lo stesso dominio; con la configurazione illustrata allora si accetteranno non solo le richieste per `www.truelite.it` ma anche quelle per `truelite.it`.

Si tenga presente che una volta che si sia usato `NameVirtualHost` tutte le richieste indirizzata verso l'indirizzo in essa indicato verranno redirette verso i vari `VirtualHost` ad essa corrispondenti, se pertanto si vuole aggiungere un dominio virtuale per chi contatta direttamente il server con l'indirizzo IP o con un nome a dominio non riportato in nessun altro `VirtualHost`, occorrerà ripetere la precedente configurazione generica all'interno di un altro `VirtualHost`, e metterla come prima nella lista di tutti quelli che si sono definiti.

Infatti tutte le volte che viene fatta una richiesta prima il server controlla se questa fa riferimento ad un indirizzo corrispondente ad una direttiva `NameVirtualHost` dopo di che controlla tutte le varie istanze di `VirtualHost` per cercare una corrispondenza del nome richiesto con le direttive `ServerName` o `ServerAlias`, se la trova usa la rispettiva configurazione, ma se non trova nulla userà le impostazioni date con il primo `VirtualHost` che della configurazione che corrisponde a quell'indirizzo, per cui questo assumerà il significato di `VirtualHost` di default.

Si tenga presente che in caso di corrispondenza la ricerca si ferma subito, per cui si possono anche specificare anche dei *sottodomini virtuali* all'interno di altri domini virtuali, ma questi saranno riconosciuti in maniera indipendente solo se il loro `VirtualHost` è posto, nel file di configurazione, prima di quello del dominio più generico.

La gestione dei domini virtuali può inoltre essere resa più comoda facendo uso della direttiva `Include` (vedi sez. 1.3.1) che ci permette di includere uno o più file di configurazione specifici contenenti le direttive `VirtualHost`. In questo modo è anche possibile delegarne la gestione anche a un utente senza fornirgli anche la possibilità di alterare la configurazione di tutto il server.

1.3.4 Configurazione base della gestione delle risorse

Come abbiamo visto in sez. 1.1 la funzionalità essenziale di un server web è quella di rispondere alle richieste HTTP eseguite da un client fornendo le risorse richieste da quest'ultimo. Come già accennato in sez. 1.1.2 benché le risorse identificate dalle URI siano espresse come dei *pathname*, non è assolutamente detto che queste corrispondano al contenuto di un file; anzi una delle caratteristiche più interessanti di Apache è quello del supporto di una lunga serie di estensioni che permettono di restituire contenuti dinamici (argomento trattato ad esempio in sez. 1.5.2).

Il caso più semplice resta comunque quello in cui ad una richiesta il server risponde inviando il file (in genere una pagina HTML) ad essa direttamente corrispondente. Come accennato in questo caso la URI di fig. 1.2 viene interpretata dal server proprio come il *pathname* di un file. Vedremo in seguito (in sez. 1.6.3) come sia possibile manipolare le modalità con cui il server interpreta la URI. Nel caso più semplice comunque questa non indica altro che il nome del file a partire dalla directory radice dei documenti, che si è specificata con la direttiva `DocumentRoot` illustrata in sez. 1.3.2.

I primi server web erano in grado soltanto di restituire come risorsa il contenuto di un file, ed anche Apache, se se ne utilizza soltanto il nucleo essenziale, non è in grado di fare molto di più. Vedremo però in sez. 1.4.1 come sia possibile, con l'uso di opportuni moduli, aggiungere

al server una enorme quantità di funzionalità diverse, comprese buona parte di quelle previste dallo standard HTTP/1.1 come la negoziazione dei contenuti.

Le funzionalità disponibili nell'accesso alle risorse vengono controllate, in fase di configurazione, dalle direttive definite nel modulo che implementa le relative funzionalità, ed in molti casi queste direttive possono essere indicate direttamente a livello globale, nel qual caso l'impostazione sarà applicata a tutti i contenuti forniti dal server. Una delle caratteristiche più interessanti di Apache è però quella delle direttive *container* illustrate in sez. 1.3.1 con le quali è possibile associare funzionalità diverse a diversi insiemi di risorse e di file, ottenendo un controllo molto raffinato sul comportamento del server.

Per questo motivo in genere l'uso delle risorse e delle funzionalità ad esse associate viene configurato sempre all'interno di un *container*. Normalmente questo può essere di tipo `Directory` se si vuole effettuare la selezione in base al nome di un file sul filesystem, o di tipo `Location` se si vuole effettuare la selezione in base al nome della risorsa espresso nella URI. Ad esempio nel file di configurazione di default per i domini virtuali installato da Debian si usa una sintassi del tipo:

default

```
<Directory />
    Options SymLinksIfOwnerMatch
    AllowOverride None
</Directory>
```

per impostare in maniera generale le funzionalità da utilizzare nell'accesso a tutti i file, si noti come avendo usato la radice dell'albero come oggetto del *container* le impostazioni si applicano anche a tutte le sottodirectory, e quindi a tutti i file.

Oltre alle impostazioni specificate nei *container* Apache implementa un ulteriore meccanismo di configurazione che permette una decentralizzazione attraverso l'uso di file mantenuti direttamente all'interno dell'albero in cui sono situate le risorse. Il nome di questi file viene impostato con la direttiva `AccessFileName`, che può essere specificata o a livello generale per tutto il server o per un singolo dominio virtuale. Il valore di default, che non c'è nessun motivo di cambiare, è `.htaccess`. All'interno di questi file si possono inserire delle direttive come se si trattasse di un *container*,[18] e queste saranno applicate alla directory in cui si è inserito il file e a tutte quelle da essa contenute.

Dato che il contenuto di `.htaccess` può soprassedere, anche se in maniera limitata, la configurazione del server è sempre il caso di usarlo con parsimonia e solo per le situazioni in cui è strettamente necessario, specie se si consente ad altri di modificarne il contenuto, dato che questo può costituire un problema di sicurezza. Occorre quindi valutare sempre se sia effettivamente necessario fornire questi privilegi aggiuntivi agli utenti e considerare i pro ed i contro della scelta.

Inoltre con l'uso di `.htaccess` si peggiorano le prestazioni dato che il server dovrà cercarlo in tutte le directory (comprese quelle di livello superiore), e leggerlo tutte le volte che si richiede un documento. Per questo motivo questa funzionalità, per quanto molto comoda, non andrebbe mai usata fintanto che è possibile eseguire le stesse impostazioni usando il file di configurazio-

[18] in realtà non tutte le direttive che compaiono in un *container* possono essere usate in un file `.htaccess`, ad esempio non è possibile usare `AllowOverride`, dato che serve appunto a disabilitare l'uso degli stessi.

ne principale; in particolare non è mai il caso di mantenere le informazioni di autenticazione all'interno di `.htaccess`.

Lo scopo del meccanismo infatti è quello di permettere una decentralizzazione delle impostazioni tutte le volte che questa è necessaria per semplicità amministrativa, ad esempio per consentire anche agli utenti di personalizzare le proprie configurazioni senza dover ricorrere ad una modifica del file di configurazione generale del server.

La possibilità di utilizzare i file `.htaccess` è controllata dalla direttiva `AllowOverride` che deve essere utilizzata all'interno di un *container* di tipo `Directory`[19] per specificare quali delle configurazioni che si sono impostate possono essere *sovrascritte* dal contenuto di `.htaccess`.

Valore	Significato
`AuthConfig`	consente l'uso di direttive di autenticazione (come `AuthDBMGroupFile`, `AuthDBMUserFile`, `AuthGroupFile`, `AuthName`, `AuthDigestRealmSeed`, `AuthType`, `AuthUserFile`, `Require`, ecc.).
`FileInfo`	consente l'uso di direttive che controllano il tipo di documento (come `AddEncoding`, `AddLanguage`, `AddType`, `DefaultType`, `ErrorDocument`, `LanguagePriority`, ecc.).
`Indexes`	consente l'uso di direttive che controllano l'indicizzazione delle directory (come `AddDescription`, `AddIcon`, `AddIconByEncoding`, `AddIconByType`, `DefaultIcon`, `DirectoryIndex`, `FancyIndexing`, `HeaderName`, `IndexIgnore`, `IndexOptions`, `ReadmeName`, ecc.).
`Limit`	consente l'uso di direttive per il controllo degli accessi (come `Allow`, `Deny` e `Order`).
`Options`	consente l'uso delle direttive che impostano funzionalità specifiche sulle risorse (come `Options` e `XBitHack`).
`All`	consente l'uso di tutte le direttive possibili all'interno di un file `.htaccess`.
`None`	disabilita l'uso dei file `.htaccess`.

Tabella 1.14: Valori degli argomenti della direttiva `AllowOverride`.

La direttiva prende come argomento una lista di valori che indicano diversi gruppi di funzionalità che è possibile impostare all'interno di `.htaccess` *soprassedendo* quelli definiti nel file di configurazione del server. L'elenco dei valori possibili è riportato in tab. 1.14, dove si fa riferimento ad alcune direttive attinenti il cui significato verrà esaminato più avanti. Il valore di default è `All`, che li attiva tutti, mentre l'uso del valore `None` disabilita completamente l'uso `.htaccess`, che non viene neanche cercato evitando così ogni degrado nelle prestazioni.

L'utilizzo dei *container* e dei file `.htaccess` consente grande flessibilità e dettaglio nel controllo delle funzionalità e delle risorse messe a disposizione, ma ha anche un certo grado di complessità ed occorre avere ben presente come si comporta il server quando le direttive impostate all'interno di diversi *container* (e nei file `.htaccess`) si sovrappongono e si vanno ad applicare alle stesse risorse. Il procedimento che determina il risultato finale è allora il seguente:

1. in prima istanza vengono processate le direttive dei *container* di tipo `Directory` che non usano espressioni regolari, a partire da quello che utilizza il pathname più breve per finire con quello applicato al pathname più lungo; se `AllowOverride` lo consente, vengono processati contestualmente gli eventuali contenuti di `.htaccess`. Se due *container* si applicano

[19] trattandosi di un file che contiene le impostazioni da usare per la directory che lo contiene, impostare l'utilizzo di `.htaccess` non ha senso all'interno di una `Location` che fa riferimento solo alle URI, né per nessuno degli altri *container* di tab. 1.12.

alla stessa directory vengono processati nell'ordine in cui sono scritti. Si tenga presente che l'ordine di scrittura vale solo quando l'argomento della direttiva `Directory` è identico, negli altri casi le direttive vengono processate in ordine inverso di genericità (dalla directory di livello più alto verso quelle di livello più basso).
2. in seconda istanza vengono processate le direttive dei *container* di tipo `Directory ~` e di tipo `DirectoryMatch`, cioè che usano espressioni regolari. In caso di sovrapposizione sulle stesse risorse vale l'ordine in cui le si sono scritte.
3. in terza istanza vengono processate le direttive dei *container* di tipo `File` e `FileMatch`, di nuovo in caso di sovrapposizione sulle stesse risorse vale l'ordine in cui le si sono scritte.
4. alla fine vengono processate le direttive dei *container* di tipo `Location`, come negli altri casi (eccettuato `Directory`) se si ha sovrapposizione sulle stesse risorse vale l'ordine in cui le si sono scritte.

Infine, a completamento di questa sequenza, si deve precisare che tutte le direttive impostate all'interno dei *container* di tipo `VirtualHost` vengono processate sempre nella sequenza precedente ma *dopo* tutte le altre impostazioni presenti in *container dello stesso tipo* fatte al di fuori dei `VirtualHost`. Questo significa che le impostazioni eseguite dentro un *Virtual Host* hanno sempre la precedenza rispetto a quelle generiche per le richieste che si applicano a detto dominio virtuale.

Il risultato finale è quello che risulta dal procedimento appena illustrato, tenendo conto che una direttiva processata dopo un'altra ne soprassiederà gli effetti. Questo vuol dire ad esempio che se si effettua una configurazione all'interno di una `Directory` che concerne le stesse risorse configurate all'interno di una `Location`, indipendentemente dall'ordine in cui le si sono scritte, quest'ultima avrà la precedenza. Per illustrare meglio l'effetto dell'ordinamento si consideri allora il seguente esempio:

```
<Files /var/www/files.html>
    Direttiva5
</Files>
<Location /prima/seconda>
    Direttiva6
</Location>
<VirtualHost *>
    <Directory /var/www/prima/seconda>
        Direttiva3
    </Directory>
</VirtualHost>
<DirectoryMatch "^www.*$">
    Direttiva4
</DirectoryMatch>
<Directory /var/www/prima/seconda>
    Direttiva2
</Directory>
<Directory /var/www/>
    Direttiva1
</Directory>
```

in questo caso prima sarà processata `Direttiva1` in quanto si applica alla directory di livello più alto, a cui segue `Direttiva2` e poi, essendo dentro un `VirtualHost`, `Direttiva3`, segue poi `Direttiva4` che è all'interno di un `DirectoryMatch` e `Direttiva5` che è all'interno di un `Files`, chiude `Direttiva6` dato che compare all'interno di una `Location`.

Una volta esaminate le modalità con cui le direttive indicate nei file di configurazione si applicano alle risorse, possiamo passare alla trattazione di quelle più utilizzate. Vedremo la maggior parte di queste nelle sezioni successive, ma una serie di funzionalità comuni sono gestite collettivamente attraverso una unica direttiva, `Options`, che permette di abilitarle o disabilitarle, che tratteremo qui. La direttiva prende come argomenti una lista di valori fra quelli illustrati in tab. 1.15, che indicano quali funzionalità abilitare; un possibile esempio potrebbe essere:

```
Options ExecCGI FollowSymLinks MultiViews
```

La direttiva è strettamente integrata con l'uso dei *container*; se utilizzata direttamente come nell'esempio precedente essa prevede come argomento solo la lista dei valori delle funzionalità da attivare, e queste varranno per tutte le risorse messe a disposizione del server. Qualora invece la si utilizzi all'interno di un *container* le funzionalità saranno attivate solo per le risorse da esso specificate, e se esso è di tipo `Directory` le impostazioni saranno ereditate da una directory a tutte quelle in essa contenute.

Opzione	Significato
`ExecCGI`	permette l'esecuzione di script CGI (vedi sez. 1.5.2), necessita del modulo `mod_cgi`.
`FollowSymLinks`	il server risolve i link simbolici,[20] l'opzione ha effetto solo in un *container* di tipo `Directory`.
`Includes`	attiva l'utilizzo dei *Server Side Include* (vedi sez. 1.5.2.
`IncludesNOEXEC`	attiva i *Server Side Include* ma non consente l'utilizzo al loro interno delle direttive `#exec` che eseguono comandi.
`Indexes`	se si richiede una directory per la quale non è presente il file dichiarato da `DirectoryIndex`, restituisce una indice del contenuto, necessita del modulo `mod_autoindex`.
`SymLinksIfOwnerMatch`	il server risolve un link simbolico soltanto se il file appartiene allo stesso utente e gruppo del link (l'opzione ha effetto solo in un *container* di tipo `Directory`).
`MultiViews`	attiva la negoziazione del contenuto.
`All`	indica tutte le opzioni eccetto `MultiViews`, è il valore di default.

Tabella 1.15: I valori delle opzioni utilizzate con la direttiva `Options`.

Per quanto appena illustrato sulle modalità di applicazione delle direttive, se per una directory si hanno più direttive `Options` assegnate tramite dei *container* di tipo `Directory` varranno quelle indicate nel *container* più specifico che sono processate per ultime, se cioè si ha una configurazione del tipo:

sito.conf

```
<Directory /var/www>
    Options MultiViews
```

[20] il fatto che il server risolva il link simbolico restituendo il file da esso indicato non cambia la modalità di accesso alla risorsa, per la quale vale quanto stabilito nel *container* `Directory` che corrisponde al pathname con la quale è stata richiesta (in sostanza se il link simbolico porta ad una directory diversa per le quali sono abilitate altre funzionalità varranno sempre solo quelle della directory originale).

```
    AllowOverride None
</Directory>
<Directory /var/www/data>
    Options Indexes FollowSymLinks
    AllowOverride None
</Directory>
```

per tutto quello che sta sotto `/var/www/data` varranno le opzioni `Indexes` e `FollowSymLinks` e non `MultiViews`.

La direttiva `Options` però supporta anche una sintassi ulteriore in cui, considerando l'ereditarietà delle configurazioni ottenute dalle directory superiori, si possono aggiungere o togliere opzioni apponendo un segno `+` o `-` al nome della stessa; se allora si ha una configurazione del tipo:

sito.conf

```
<Directory /var/www>
    Options MultiViews
    AllowOverride None
</Directory>
<Directory /var/www/data>
    Options +Indexes +FollowSymLinks
    AllowOverride None
</Directory>
<Directory /var/www/data/images>
    Options -MultiViews
    AllowOverride None
</Directory>
```

in questo caso a `/var/www/data` si applicheranno tutte e tre le opzioni `Indexes`, `FollowSymLinks` e `MultiViews`, mentre a `/var/www/data/images` si applicheranno solo le due opzioni `Indexes` e `FollowSymLinks`, avendo rimosso `MultiViews`.

1.4 Uso dei moduli e principali funzionalità

Come accennato la gran parte delle funzionalità di Apache vengono realizzate attraverso l'uso di opportuni moduli che forniscono sia il supporto che le relative configurazioni. Tratteremo in questa sezione l'uso dei moduli per ampliare le funzionalità di Apache a partire dalla descrizione delle modalità con cui essi stessi vengono gestiti, per passare poi ad una panoramica delle funzionalità fornite dai più importanti, che sono essenziali anche solo per avere una implementazione completa del protocollo HTTP.

1.4.1 La gestione dei moduli

Come illustrato in sez. 1.2.1 delle caratteristiche più interessanti di Apache è la sua modularità; il server cioè è in grado di fornire nuove funzionalità caricando al suo interno dei moduli di espansione anche quando è già in esecuzione, usando quello che viene chiamato il supporto DSO (acronimo di *Dynamic Shared Object*), in cui i moduli possono essere distribuiti separatamente dal server come librerie dinamiche in altrettanti file `.so`. È attraverso questi moduli che si

possono gestire una gran quantità di funzionalità diverse, che vanno dall'interazione di Apache con linguaggi di scripting, alla possibilità di utilizzare il potente motore di per la riscrittura degli indirizzi, al supporto di pressoché infiniti meccanismi di autenticazione di accesso alle risorse.

Come accennato in sez. 1.2.2 con Apache 2 la modularizzazione è stata spinta al limite estremo della implementazione delle funzionalità di base del server attraverso gli MPM; abbiamo già detto però che, almeno fino alla versione 2.2, in questo caso è possibile scegliere soltanto un MPM alla volta per binario (non è possibile cioè caricare dinamicamente un MPM diverso), limite superato con la nuova versione 2.4.

Il meccanismo dei moduli permette di non dover ricompilare Apache ogni volta che si ha bisogno di una funzionalità nuova. Permette anche di cambiare al volo la configurazione, richiedendo semplicemente, grazie alla presenza della direttiva `IfModule`, la rilettura della configurazione, e di avere diverse istanze di Apache che girano sulla stessa macchina con configurazioni profondamente diverse.

A parte gli MPM, che sono un caso particolare, le modalità di utilizzo dei moduli sono due, questi possono essere compilati all'interno del server, e venire attivati e disattivati, oppure possono essere caricati dinamicamente a server attivo da file esterni. Ovviamente quest'ultima è la modalità preferita in quanto permette una maggiore flessibilità nella gestione, ed un minore consumo di memoria.

Come visto in sez. 1.2.3 l'opzione `-l` di ci consente di vedere quali sono i moduli compilati nel server e direttamente inseriti all'interno del binario che viene lanciato; in genere questi sono mantenuti al minimo essenziale, ad esempio con la versione 2.2 di Apache installata su una Debian Squeeze si avrà:

```
anarres:/home/piccardi# apache2 -l
Compiled-in modules:
  core.c
  mod_log_config.c
  mod_logio.c
  prefork.c
  http_core.c
  mod_so.c
```

Quelli elencati nell'esempio precedente sono i moduli compilati all'interno del server, ma come dicevamo è possibile far caricare al server i moduli in fase di avvio o di rilettura del file di configurazione; anche questa funzionalità viene fornita attraverso un modulo, `mod_so.c`, che come nell'esempio, generalmente viene sempre inserito nel binario del server.

Con questo modulo diventa disponibile la direttiva `LoadModule` che permette di caricare un modulo nel server eseguendo il collegamento del codice dello stesso ed attivarne al contempo l'utilizzo. La direttiva vuole come primo argomento un identificatore del modulo e come secondo argomento il file (nella forma di uno *shared object*) che contiene il relativo codice. La convenzione vuole che se il codice sorgente del modulo è ad esempio `mod_`*`nome`*`.c` la stringa identificativa sia *`nome`*`_module` ed il codice sia mantenuto in un file `mod_`*`nome`*`.so` in una opportuna directory.

A partire da Apache 2.0 Debian per la gestione dei moduli ha adottato l'approccio di non di fare riferimento ad un singolo file che indichi quale utilizzare, utilizzando invece il contenuto di una intera directory, come illustrano negli esempi di `Include` citati in sez. 1.3.1.

Per migliorare la gestione Debian prevede inoltre la presenza di due directory; la prima è `/var/apache2/mods-available`, in cui i vari pacchetti che forniscono moduli per Apache inseri-

scono i loro file di configurazione,[21] nella forma `nomemodulo.conf` e `nomemodulo.load`. La seconda è `/var/apache2/mods-enabled`, in cui vengono creati dei link simbolici ai file della directory precedente così da scegliere quali moduli caricare all'avvio, si noti infatti come nell'estratto di configurazione citato vengono inclusi i file da questa directory.

Per gestire tutto quanto in maniera semplificata, Debian installa anche due comandi, `a2enmod` e `a2dismod` che prendono come argomento il nome di un modulo da abilitare o disabilitare; il nome è sempre quello del file presente in `/var/apache2/mods-available` considerato senza estensione. Se non si specifica nessun nome i comandi controllano in `/var/apache2/mods-available` i moduli disponibili e chiedono su quale operare. Se ad esempio si vuole attivare il modulo per la gestione delle sessioni cifrate si dovrà usare il comando:

```
a2enmod ssl
```

Una volta che si sono attivati i moduli se ne possono usare le funzionalità; come accennato in sez. 1.3.1 una delle modalità tipiche per fare questo è all'interno di una direttiva `IfModule` così da evitare errori all'avvio qualora il modulo non fosse presente; già nell'estratto del file di configurazione riportato alla fine di sez. 1.3.2 abbiamo visto come la direttiva sia usata per importare le direttive relative all'MPM `prefork`. Altri esempi si trovano nei vali file di configurazione dei singolo moduli che su Debian vengono utilizzati per la configurazione generica degli stessi, ad esempio si avrà qualcosa del tipo:

/etc/apache2/mods-available/status.conf

```
<IfModule mod_status.c>
    ExtendedStatus On
</IfModule>
```

In questo caso si utilizzano le funzionalità del modulo `mod_status.c` che crea una pagina dinamica (la cui posizione dovrà essere definita con un *container* ad esso dedicato utilizzando la funzionalità degli *handler*, su cui torneremo in sez. 1.5.2) contenente le informazioni relative allo stato del server. Il modulo definisce una direttiva `ExtendedStatus` che quando attivata con l'argomento `On` permette di ottenere un maggior numero di informazioni.

Si tenga presente che gran parte delle funzionalità di Apache che illustreremo in seguito sono disponibili solo attraverso l'uso degli opportuni moduli, un elenco di quelli più importanti, con una breve descrizione delle loro funzionalità, è riportato in tab. 1.16. Per l'elenco completo dei moduli distribuiti ufficialmente con Apache si può fare riferimento alla documentazione relativa ai moduli consultabile a partire da `http://httpd.apache.org/docs/`, si tenga inoltre presente che ogni versione ha i suoi moduli e che loro disponibilità e stabilità dipende anche, a partire da Apache 2, dall'MPM utilizzato.

Infine è da sottolineare che grazie alla semplicità dell'interfaccia di estensione, sono stati sviluppati una enorme quantità di altri moduli anche da soggetti esterni, in grado di fornire le funzionalità più varie. Alcuni di questi, come quelli per la gestione delle comunicazioni cifrate (vedi sez. 1.6.4), sono stati riportati in seguito all'interno del progetto (nel caso citato a partire da Apache 2.0), altri, come quelli per l'esecuzione diretta di programmi scritti in vari linguaggi come il PHP (vedi sez. 1.5.2), continuano ad essere sviluppati esternamente. In generale attraverso

[21] questo ha il vantaggio che il pacchetto sa esattamente quale file ha installato e può rimuoverli senza dover andare a modificare altri file esterni.

Modulo	Funzionalità
mod_log_config	permette di gestire le modalità di produzione dei log e configurare il loro formato.
mod_mime_magic	consente di identificare il tipo di file controllando i primi byte del contenuto dello stesso in maniera analoga al comando file, è pensato come ausilio per mod_mime per i casi in cui quest'ultimo fallisce.
mod_mime	consente di determinare una serie di informazioni riguardo un documento (il tipo di file o la lingua in cui è scritto) sulla base del nome dello stesso. Consente la creazione delle intestazioni necessarie alla negoziazione dei contenuti (quelle di tab. 1.7) o l'associazione ad un *handler* (vedi sez. 1.5.2) che ne gestisca l'elaborazione nel server.
mod_negotiation	implementa la negoziazione dei contenuti e la configurazione delle modalità con cui viene eseguita la scelta del documento più appropriato (vedi sez. 1.5.1).
mod_status	crea una pagina dinamica con le statistiche delle prestazioni del server.
mod_info	crea una pagina con le informazioni di configurazione del server.
mod_autoindex	permette l'indicizzazione automatica delle directory, e definisce le direttive che permettono di controllarne il formato e l'associazione fra tipo di file ed icone.
mod_dir	permette di impostare il file da restituire (in genere index.html) quando il client richiede una directory (vedi sez. 1.4.2).
mod_cgi	permette l'esecuzione degli script CGI (vedi sez. 1.5.2).
mod_actions	permette di associare l'esecuzione di CGI (vedi sez. 1.5.2) ad un tipo di file o al metodo HTTP usato dal client.
mod_userdir	fornisce il contenuto delle pagine degli utenti quando vengono accedute con una URI del tipo /~utente/.
mod_alias	permette mappare una risorsa richiesta da una URI su una directory qualsiasi nel filesystem o di redirigere le richieste ad altri server.
mod_rewrite	esegue lo stesso compito di mod_alias eseguendo una associazione fra URL e file corrispondente usando un motore di riscrittura basato su espressioni regolari (vedi sez. 1.6.3).
mod_access	fornisce un elementare controllo di accesso ai contenuti basato sulle caratteristiche del client, come IP o nome a dominio (vedi sez. 1.6.2).
mod_auth	fornisce un meccanismo di autenticazione degli utenti (vedi sez. 1.6.2), questo modulo gestisce l'autenticazione sulla base del contenuto di alcuni file di testo, esistono altri moduli (come mod_auth_db, mod_auth_digest, ecc.) che consentono di estendere a piacere le modalità di autenticazione.
mod_include	fornisce il supporto dei *Server Side Include* (vedi sez. 1.5.2).
mod_proxy	fornisce le funzionalità di cache e proxy per Apache.
mod_env	permette di controllare le variabili di ambiente che vengono usate nell'esecuzione di script CGI e nei documenti SSI.
mod_setenvif	permette di impostare delle variabili di ambiente sulla base dei dati ricavati dal client.

Tabella 1.16: I principali moduli usati da Apache.

una opportuna scelta dei moduli diventa possibile di estendere in maniera enorme le funzionalità del server.

1.4.2 Le risorse associate alle directory

Abbiamo visto in sez. 1.3.4 come una delle funzionalità la cui disponibilità è gestita dalla direttiva generica `Options` è quella della indicizzazione delle directory. Questa funzionalità entra in gioco tutte le volte che un client richiede una URI corrispondente ad una directory, come nel caso classico un cui si richiede la pagina principale di un sito specificando il solo nome a dominio, in cui la URI corrispondente è semplicemente "`/`".

In casi come questo si pone il problema di quale risorsa fornire come risposta, dato che non è stato richiesto un file specifico. Le modalità più comuni con cui Apache gestisce la situazione sono due: o viene utilizzato uno specifico file (ad esempio `index.html`) presente in quella directory, o viene generata direttamente dal server una pagina che contiene una lista dei file presenti nella directory.

Nel primo caso il modulo `mod_dir` consente di scegliere quale file utilizzare come risposta e si può utilizzare la direttiva `DirectoryIndex` per impostare il nome di questo file. Il valore di default è appunto `index.html`, ma la direttiva permette di specificare una lista di valori che indicano delle possibili alternative, e verrà utilizzato il primo file presente nella directory che corrisponde ad uno di quelli nella lista, secondo l'ordine in cui sono scritti. Una configurazione tipica con questa direttiva è la seguente, presa dal file installato da una Debian Squeeze:

/etc/apache2/mods-available/dir.conf

```
<IfModule mod_dir.c>
    DirectoryIndex index.html index.cgi index.pl index.php index.xhtml index.htm
</IfModule>
```

Si noti che nell'elenco compaiono anche dei file che non necessariamente corrispondono ad una pagina con del contenuto statico, come `index.php` e `index.cgi`. La direttiva infatti indica quale file utilizzare, il contenuto restituito dipenderà anche da come il server utilizzerà lo stesso, generando eventualmente dei contenuti dinamici (torneremo su questo in sez. 1.5.2).

Se pertanto si fosse voluto dare la precedenza a delle pagine scritte in PHP, e usando una di queste come documento di default prima di qualunque altro, si sarebbe dovuto spostare `index.php` in cima alla lista precedente. Se invece si volesse utilizzare un file con un nome diverso o con una estensione diversa si sarebbe dovuto aggiungere lo stesso nella lista.

Qualora nella directory richiesta non sia presente nessuno dei file specificati con `DirectoryIndex` il comportamento di default prevede che server restituisca un errore, a meno che per la directory in questione non sia stata abilitata l'opzione `Indexes`. In tal caso infatti verranno invece utilizzate le funzionalità provviste dal modulo `mod_autoindex` (occorre ovviamente averlo caricato) per generare automaticamente una pagina HTML contenente una lista dei file contenuti nella directory stessa. Questo significa anche che fintanto che esiste un file previsto da `DirectoryIndex`, l'elenco dei file nella directory non sarà mai generato.

Il modulo fornisce inoltre una serie di direttive che permettono di controllare ogni aspetto della generazione della pagina con la lista dei file. Di nuovo estraiamo dal file di configurazione distribuito con una Debian Squeeze un esempio di uso delle varie direttive fornite da `mod_autoindex`:

/etc/apache2/mods-available/autoindex.conf

```
<IfModule mod_autoindex.c>
    IndexOptions FancyIndexing NameWidth=*
    AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip
    AddIconByType (TXT,/icons/text.gif) text/*
    AddIconByType (IMG,/icons/image2.gif) image/*
    AddIconByType (SND,/icons/sound2.gif) audio/*
    AddIconByType (VID,/icons/movie.gif) video/*
    AddIcon /icons/binary.gif .bin .exe
    AddIcon /icons/binhex.gif .hqx
    AddIcon /icons/tar.gif .tar
    AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
    AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
    AddIcon /icons/a.gif .ps .ai .eps
    AddIcon /icons/layout.gif .html .shtml .htm .pdf
    AddIcon /icons/text.gif .txt
    AddIcon /icons/c.gif .c
    AddIcon /icons/p.gif .pl .py
    AddIcon /icons/f.gif .for
    AddIcon /icons/dvi.gif .dvi
    AddIcon /icons/uuencoded.gif .uu
    AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl
    AddIcon /icons/tex.gif .tex
    AddIcon /icons/bomb.gif core
    AddIcon /icons/deb.gif .deb
    AddIcon /icons/back.gif ..
    AddIcon /icons/hand.right.gif README
    AddIcon /icons/folder.gif ^^DIRECTORY^^
    AddIcon /icons/blank.gif ^^BLANKICON^^
    DefaultIcon /icons/unknown.gif
    ReadmeName README.html
    HeaderName HEADER.html
    IndexIgnore .??*
```

Le direttive controllano i vari aspetti della visualizzazione della lista; un elenco delle principali è il seguente, al solito la documentazione completa è fornita sul sito di Apache, ed in particolare all'indirizzo `http://httpd.apache.org/docs/2.2/mod/mod_autoindex.html`:

`AddIconByEncoding` imposta una associazione fra una icona ed un file sulla base del *MIME-encoding* (vedi sez. 1.5.2) dello stesso. Il primo argomento serve ad indicare l'icona da usare; questo può essere espresso o con la URI cui questa è disponibile, o nella forma `(NAME,/uri/to/icon.gif)` in cui si fornisce ai browser non grafici la stringa `NAME` come alternativa all'immagine. Il secondo argomento indica il tipo di codifica del file; ad esempio nel precedente estratto di configurazione si associa una icona (o il testo `CMP`) a tutti i file compressi con una certa codifica,[22] utilizzando la direttiva:

```
AddIconByEncoding (CMP,/icons/compressed.gif) x-gzip x-bzip2
```

`AddIconByType` imposta una associazione fra una icona ed un file sulla base del *MIME-type* (vedi sez. 1.5.2) dello stesso. Il primo argomento è identico alla

[22] nel caso specifico si tratta di file compressi o con `bzip2` o con `gzip`.

precedente `AddIconByEncoding`, il secondo indica il tipo di file in base al *MIME-type*, ad esempio nel precedente estratto di configurazione si associa una icona (o il testo `IMG`) a tutti le immagini utilizzando la direttiva:

```
AddIconByType (IMG,/icons/image2.gif) image/*
```

`AddIcon` imposta una associazione fra una icona ed un file il cui nome termina con la stringa passata come secondo argomento. Il primo argomento indica l'icona da usare nello stesso formato delle precedenti `AddIconByType` e `AddIconByEncoding`. Come secondo argomento si possono usare i due nomi riservati `^^DIRECTORY^^` e `^^BLANKICON^^` per indicare rispettivamente una directory o una riga vuota, ed i nomi dei file possono essere espressi anche con i caratteri jolly; ad esempio nel precedente estratto di configurazione si associa una icona (o il testo `IMG`) a tutti le directory utilizzando la direttiva:

```
AddIcon /icons/folder.gif ^^DIRECTORY^^
```

`DefaultIcon` permette di specificare una icona da associare a tutti i file per i quali non esiste una associazione già definita. Prende come argomento la URI della stessa; ad esempio nel precedente estratto di configurazione si specifica questa icona utilizzando la direttiva:

```
DefaultIcon /icons/unknown.gif
```

`AddDescription` permette di associare ad un file una stringa di descrizione (di un massimo di 23 caratteri) che viene visualizzata in una apposita colonna. Prende come primo argomento la stringa (racchiusa fra virgolette) da usare come descrizione e come secondo argomento il file.

`ReadmeName` specifica il nome del file il cui contenuto (in genere un testo contenente informazioni riguardo la directory) viene inserito automaticamente in coda alla pagina HTML con l'indice. Il nome del file deve essere specificato come argomento, ed è risolto rispetto alla directory di cui si sta facendo l'indice; ad esempio nel precedente estratto di configurazione si specifica questo file utilizzando la direttiva:

```
ReadmeName README.html
```

`HeaderName` specifica il nome del file il cui contenuto viene inserito automaticamente in cima alla pagina, se questo contiene le intestazioni del preambolo HTML se ne deve bloccare la generazione automatica abilitando l'opzione `SuppressHTMLPreamble` (vedi tab. 1.17). Il nome del file deve essere specificato come argomento, ed è risolto rispetto alla directory di cui si sta facendo l'indice; ad esempio nel precedente estratto di configurazione si specifica questo file utilizzando la direttiva:

Opzione	Significato
`DescriptionWidth`	assegnata (con `DescriptionWidth=n`) ad un numero imposta la dimensione in caratteri della descrizione associata ad un file con `AddDescription`, se si usa il valore $*$ la dimensione è impostata automaticamente.
`FancyIndexing`	attiva la generazione di indici complessi dotati di decorazioni e informazioni aggiuntive.
`FoldersFirst`	fa sì che le directory appaiano sempre in cima alla lista, prima dei file.
`IconHeight`	usata insieme ad `IconWidth` fa sì che il server includa gli attributi di altezza e larghezza delle immagini nel codice HTML dell'indice della pagina, il valore di default è quello delle icone distribuite con il server, se ne può usare un altro assegnando un valore all'opzione (con `IconHeight=pixel`).
`IconWidth`	usata insieme ad `IconHeight`, ha lo stesso significato ed usa la stessa sintassi.
`IconsAreLinks`	associa anche le icone ai link al file nella pagina HTML.
`IgnoreCase`	ordina alfabeticamente i file della lista senza distinguere fra maiuscole e minuscole.
`NameWidth`	assegnata (con `NameWidth=n`) ad un numero imposta la dimensione della colonna contenente il nome di un file a quel numero di caratteri, se si usa il valore $*$ la dimensione è impostata automaticamente sulla base della più lungo dei nomi dei file.
`ScanHTMLTitles`	abilita una funzionalità di scansione del contenuto dei file per determinare una descrizione da usare quando non ce n'è una disponibile tramite `AddDescription`.
`SuppressColumnSorting`	disabilita la capacità di creare degli ordinamenti della lista in base alle caratteristiche mostrate nelle varie colonne (accessibili con i link sui titoli delle stesse).
`SuppressDescription`	disabilita la stampa delle descrizioni dei file.
`SuppressHTMLPreamble`	sopprime la generazione delle intestazioni del preambolo della pagine HTML in modo da usare quelle fornite dal file di intestazione personalizzato specificato con `HeaderName`.
`SuppressLastModified`	disabilita la stampa del tempo di ultima modifica.
`SuppressSize`	disabilita la stampa della dimensione del file.
`TrackModified`	riporta i dati dei file nelle intestazioni HTTP `Last-Modified` e `ETag` associate alla pagina.

Tabella 1.17: I valori delle opzioni utilizzate con la direttiva `IndexOptions`.

```
HeaderName HEADER.html
```

`IndexIgnore` specifica una lista di file da ignorare quando viene eseguita la lista del contenuto della directory. Prende come argomenti la lista dei file, che possono essere indicati anche con dei caratteri jolly. A differenza di altre direttive se la si invoca più volte i nuovi file sono aggiunti alla lista (e non sovrascrivono il valore precedente); ad esempio nel precedente estratto di configurazione si indica di non mostrare tutti file *invisibili*[23] utilizzando la direttiva:

```
IndexIgnore .??*
```

[23] nel senso della convenzione di avere un "."come primo carattere del nome.

IndexOptions è la direttiva principale che consente di controllare gli aspetti generali della pagina con la lista dei file, prende come argomento una lista di opzioni fra quelle riportate in tab. 1.17. La direttiva supporta l'utilizzo dei modificatori `+` e `-` in maniera analoga ad `Options` per aggiungere o togliere opzioni impostate in directory sovrastanti. Buona parte di queste direttive hanno senso solo se si utilizza l'opzione `FancyIndexing` che consente di generare delle pagine HTML con una struttura complessa in grado di utilizzare tutte le funzionalità controllate dalle precedenti direttive.

Nell'esempio preso dall'estratto del file di configurazione sono riportate solo delle impostazioni generali valide per tutto il server, è comunque possibile usare le direttive appena illustrate anche all'interno di container di tipo `Directory` e affini[24] e `VirtualHost` e nei file `.htaccess` per delle configurazioni specifiche per casi particolari.

1.4.3 La gestione dei file di log

Benché la problematica sia rilevante per l'amministrazione di qualunque servizio, nel caso di Apache la gestione dei file di log assume un'importanza particolare, dato che questi consentono di tenere traccia di tutte le transazioni HTTP che il server esegue. È così possibile controllare nei dettagli l'attività dello stesso, ed oltre a rilevare gli eventuali errori si possono ottenere sia le statistiche di utilizzo che i dati riguardanti l'accesso, che sono informazioni di grande utilità nella gestione di un sito web.

In generale si distinguono due tipi di messaggi prodotti dal server, che devono essere registrati nei file di log: quelli di errore e quelli di accesso. Nel primo caso i messaggi devono essere generati comunque e la funzionalità viene fornita direttamente dal nucleo di Apache, nel secondo caso invece diventa possibile un controllo estremamente dettagliato delle informazioni registrate grazie alle funzionalità di `mod_log_config`.

Dato che la funzionalità fa parte del nucleo di Apache i messaggi di errore vengono generati comunque; la gestione della loro registrazione è affidata alle seguenti direttive:

ErrorLog indica il nome del file su cui salvare i messaggi di errore generati dal server. Se si usa un pathname relativo esso è preso a partire dalla directory indicata da `ServerRoot`; dato che in genere questa è sotto `/etc` e non è scrivibile in genere si specifica un pathname assoluto per far scrivere il file sotto `/var/log/`. Nel caso di Debian ad esempio si usa `/var/log/apache2/error.log`. La direttiva supporta anche l'uso della forma `syslog:facility` al posto del nome del file per inviare i messaggi di errore al sistema del *syslog*, nella *facility* specificata (per una trattazione del *syslog* si veda la sez. 3.3.3 di [AGL]).

LogLevel associata alla precedente permette di stabilire la quantità di informazioni da inviare sui log specificati da `ErrorLog`. Sono definiti gli stessi livelli utilizzati dal sistema del *syslog* (al solito si può fare riferimento al testo citato o alla pagina di manuale di `syslog.conf`); il valore di default è `warn`.

[24] si intende con questo i *container* `Directory`, `Location` e `Files` che operano tutti su selezioni di file e risorse.

Normalmente è opportuno utilizzare `ErrorLog` a livello di configurazione generale del server per raccogliere in un solo luogo tutti i messaggi di errore, è però possibile usare queste stesse direttive per creare dei file separati per ciascun *Virtual Host* presenti nel sistema, inserendo le stesse direttive all'interno di un *container* del suddetto tipo.[25] Un esempio di messaggio di errore è il seguente:

```
/var/log/apache2/error.log
[Wed Aug 25 15:35:08 2004] [error] [client 127.0.0.1] File does not exist: /var/
www/status-server
```

Questi messaggi prevedono sempre tre campi delimitati fra parentesi quadre: prima la data ed l'ora dell'errore, poi il livello di gravità, seguito dall'indirizzo IP da cui si è collegato il client la cui richiesta ha generato l'errore; a questi segue il testo del messaggio vero e proprio che spiega le cause dell'errore stesso, nell'esempio precedente la richiesta di un file che non esiste. Si tenga presente che in questi messaggi vengono sempre riportati i pathname assoluti rispetto al filesystem su cui gira il server.

Per la gestione dei log di accesso è invece è possibile utilizzare una configurazione specifica che permetta di stabilire che tipo di messaggi far generare al server. Di nuovo l'uso dei moduli ci consente una grande flessibilità ed in questo caso modulo più importante è `mod_log_config`, che definisce le direttive principali per la gestione dei log di accesso, ed un meccanismo molto potente che consente di selezionare le informazioni che devono essere registrate.

Come per `ErrorLog` anche le direttive per la registrazione delle informazioni di accesso possono essere specificate o a livello generale per tutto il server o per i singoli *Virtual Host*; un elenco delle principali è il seguente:

`TransferLog` definisce un file su cui verranno registrati i dati di accesso; la direttiva permette solo di usare il formato di default (quello stabilito dalla più recente impostazione effettuata con `LogFormat`) e prende come unico argomento o il nome di un file (se relativo viene risolto rispetto a `ServerRoot`) o il carattere "`|`" seguito dal pathname di un programma che sarà eseguito e riceverà i dati in ingresso.[26] Un esempio di questa direttiva potrebbe essere il seguente:

```
TransferLog /var/log/apache/access.log
```

`CustomLog` è la principale direttiva di registrazione, rispetto alla precedente `TransferLog` permette di specificare il formato dei dati e di eseguire registrazioni condizionali (in base al valore delle variabili di ambiente gestite da `mod_setenvif`). La direttiva prevede un primo argomento identico a quello di `TransferLog`, a cui segue un secondo argomento che indica una etichetta associata ad un certo formato dei file di log, definito con una opportuna direttiva `LogFormat`.[27] Infine è possibile specificare una condizione nella forma `env=variabile` che

[25] il problema in questo caso è che se si ha un gran numero di *Virtual Host* si deve aprire un gran numero di file, col rischio di esaurire le risorse a disposizione.

[26] il programma viene lanciato per conto dell'utente che ha avviato Apache, che di norma è l'amministratore, pertanto ci si espone ad un grosso rischio di sicurezza.

[27] in realtà è possibile specificare anche direttamente una stringa di formato, ma usare `LogFormat` rende la configurazione molto più leggibile.

consente di eseguire o meno la registrazione di un certa richiesta sulla base della presenza o meno nell'ambiente di una certa variabile (queste vengono impostate a parte dalla direttiva `SetEnvIf`, come vedremo in sez. 1.5.2), la condizione può anche essere invertita se specificata come `env=!valore`. Un esempio di questa direttiva potrebbe essere il seguente:

```
CustomLog /var/log/apache/access.log combined
```

`LogFormat` il formato dei dati registrati nei log viene configurato da questa direttiva, questa può assumere due forme con uno o due argomenti. Nel primo caso l'argomento specifica il formato delle informazioni da registrare, e quello diventa il formato di default (e sarà utilizzato da `TransferLog`. Se invece si specificano due argomenti il secondo deve essere una stringa che fa da una etichetta per identificare il formato definito nel primo argomento, che poi potrà essere riutilizzata nelle direttive `CustomLog`. Un esempio di questa direttiva è il seguente, che definisce il cosiddetto *Common Log Format*:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
```

Per usare in maniera efficace le capacità di registrazione di Apache occorre approfondire il formato del primo argomento della direttiva `LogFormat` che specifica quali dati saranno registrati. L'argomento deve essere passato come stringa delimitata da virgolette, pertanto per poter utilizzare queste ultime all'interno delle stringa si dovrà proteggerle con il carattere "`\`"; questo consente anche di inserire all'interno della stringa anche caratteri speciali come la tabulazione ed il ritorno a capo con sequenze del tipo di `\t` e `\n`. Gli altri caratteri sono convertiti letteralmente mentre il carattere "`%`" viene usato per identificare delle direttive di formattazione che vengono tradotte in una serie di dati relativi alla richiesta.

Un elenco delle principali informazioni che è possibile estrarre dalle transazioni e registrare sui log con le direttive di formattazione è riportato in tab. 1.18; per l'elenco completo si può fare riferimento alla documentazione di `mod_log_config` (disponibile su `http://httpd.apache.org/docs/2.2/mod/mod_log_config.html`).

Si tenga presente che oltre alla forma illustrata nella prima colonna di tab. 1.18 una direttiva di formattazione può assumere anche una più complessa che consente di inserire i dati della richiesta cui fa riferimento a seconda dei codici di ritorno della stessa; in questo caso si avrà una sintassi del tipo:

```
"%!200,304,302{Referer}i"
```

In questo caso al carattere "`%`" si fa seguire una lista dei possibili codici di ritorno, separati da virgole, per i quali si vuole estrarre l'informazione; la lista può anche essere negata apponendovi il carattere "`!`". Nel caso citato si otterrà ad esempio che il contenuto dell'intestazione `Referer:` sarà inserito nel log solo per le richieste che non ricevono una risposta valida (gli stati elencati sono quelli per i quali si riceve la risorsa richiesta) mentre per quelle che non soddisfano la condizione verrà restituito il carattere "`-`".

Un esempio di vari formati usati nella registrazione dei dati di accesso è quello che si ha nel seguente estratto, ripreso del file di configurazione di default installato da una Debian Squeeze, in cui si è preso (al solito eliminando i commenti) la sezione dedicata alle direttive per i log:

Direttiva	Significato
%a	l'indirizzo IP da cui si è ricevuta la richiesta.
%A	l'indirizzo IP su cui si è ricevuta la richiesta.
%b	la dimensione della risorsa inviata (secondo quanto riportato dall'intestazione Content-Lenght) in byte (se nulla viene riportato un -).
%{VAR}e	il contenuto della variabile di ambiente VAR.
%f	il nome del file.
%h	nome della macchina remota (o se manca la risoluzione l'indirizzo IP).
%H	il protocollo delle richiesta.
%{Header}i	il valore dell'intestazione specificata da Header nella richiesta inviata al server.
%l	l'utente che ha eseguito la richiesta (se ottenibile con il programma identd).
%m	il metodo HTTP usato nella richiesta.
%{Header}o	il valore dell'intestazione specificata da Header nella risposta inviata dal server.
%r	la prima riga della richiesta.
%s	il codice di ritorno (vedi tab. 1.8) per le richieste che vengono redirette internamente viene usato il codice della richiesta originale, se si usa %>s quello dell'ultima.
%t	data e ora (in formato standard)
%T	il tempo usato per rispondere alla richiesta in secondi.
%u	l'utente come risultante dalla procedura di autenticazione, il valore può essere fittizio se l'autenticazione è fallita.
%U	la URI richiesta dal client.

Tabella 1.18: Principali direttive di formattazione usate negli argomenti di LogFormat.

apache2.conf

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" \"%{forensic-id}n\" %T %v" full
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" \"%{forensic-id}n\" %P %T" debug
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" \"%{forensic-id}n\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{forensic-id}n"" forensic
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
CustomLog /var/log/apache/access.log combined
```

Con l'uso di una opportuna configurazione è possibile far registrare al server serie di informazioni legate ad ogni richiesta HTTP (data ed ora, indirizzo del client, browser utilizzato, riferimenti, nome del file, ...). L'analisi dei log permette permetterà poi di ottenere una serie di statistiche; per questo sono infatti disponibili applicazioni dedicate, come Awstats (http://awstats.sourceforge.net/) o Webalizer (http://www.mrunix.net/webalizer/).

Si noti come nella configurazione di esempio sia stato utilizzato il formato etichettato come combined. Questo formato, detto *Combined Log Format*, è molto comune e molto utilizzato dai programmi che generano statistiche perché fornisce molte informazioni. Possiamo ad esempio valutare non soltanto gli accessi ai singoli siti, ma la loro provenienza, il numero di pagine visitate, e il modo in cui si è arrivati a quella pagine. Un esempio del risultato ottenibile con questo formato è il seguente:

/var/log/apache2/error.log

```
127.0.0.1 - - [25/Aug/2004:15:34:05 +0200] "GET /server-status HTTP/1.1" 404 285
 "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7) Gecko/20040823 Firefox/0.9
.3" "-"
```

dove si trovano varie informazioni, fra cui appunto quelle relative al browser utilizzato.

Per la gestione dei log, che su server altamente utilizzati raggiungono spesso dimensioni consistenti, occorre predisporre molto spazio su disco, magari in una partizione apposita e dotarsi di strumenti che ci permettano di archiviare comodamente i vari file. Altrimenti possiamo scegliere di usare altri strumenti creati per la gestione dei log dei server web, ad esempio Cronolog, che permette di avere i log suddivisi in base al mese o alla settimana a cui si riferiscono, il tutto in modo completamente automatico.

1.5 La gestione dei contenuti

In questa sezione tratteremo in maggiore dettaglio le modalità con cui si possono controllare i contenuti che vengono forniti dal server in risposta alla richiesta di una risorsa. Vedremo in particolare la gestione della negoziazione dei contenuti e come Apache supporti una architettura generica che consente di controllare sia le modalità con cui possono essere generati contenuti dinamici (i cosiddetti *handler*) che quelle con cui possono essere effettuate operazioni sui dati in uscita ed in ingresso (i filtri).

1.5.1 Tipi di file e negoziazione dei contenuti

Una delle caratteristiche di HTTP/1.1 accennata in sez. 1.1.3 è quella di fornire delle indicazioni sul tipo di file che viene restituito ad una richiesta. Per far questo un server deve essere in grado di classificare i file. Apache è in grado di farlo un maniera estremamente flessibile attraverso il modulo `mod_mime`. La classificazione avviene sulla base di due caratteristiche astratte che vengono associate al file, la lingua in cui è scritto ed il tipo di contenuto dello stesso.

Come già accennato in sez. 1.1.3 per indicare il tipo di file viene usato lo standard MIME (acronimo che sta per *Multipurpose Internet Mail Extensions*). Questa è una standardizzazione che prevede una descrizione del contenuto di un file sulla base di una stringa testuale che ne indica una tipologia generica ed una codifica specifica, nella forma `type/encoding`. La stringa viene spesso chiamata il *MIME-type* del file, mentre si chiama *MIME-encoding* la descrizione della codifica. Un esempio di alcuni *MIME-type* è riportato in tab. 1.19, in genere una lista più completa viene mantenuta nel file `/etc/mime.types`.[28]

La definizione dell'insieme dei *MIME-type*, così come di altre caratteristiche dei file come i nomi delle varie codifiche dei set di caratteri ASCII estesi, come `ISO-8859-1` o `ISO-8859-9` (i cosiddetti *MIME-charset*) e le sigle che identificano le localizzazioni, come `it` o `en-us` (i cosiddetti *MIME-lang*) viene curata dalla IANA e le liste aggiornate con tutte queste informazioni vengono pubblicate su Internet all'indirizzo `http://www.iana.org/assignments/media-types/`.

Il modulo `mod_mime` consente di impostare le modalità con cui Apache gestisce l'associazione fra le proprietà dei file (lingua, set di caratteri e tipo) ed i file stessi, così che il server possa

[28] il file è generico e non attiene alla configurazione di Apache, dato che sono molti i programmi che fanno riferimento all'uso dei *MIME-type*, a partire da quelli relativi alla posta elettronica da cui essi traggono origine.

MIME-type	Significato
text/plain	semplice file di testo.
text/html	testo in formato HTML.
image/jpeg	immagine in formato JPEG.
image/png	immagine in formato PNG.
application/octet-stream	tipo generico: una serie di byte.
application/pdf	un documento PDF.
audio/midi	un file audio in formato MIDI.
audio/ogg	un file audio in formato Ogg Vorbis.
video/mpeg	un file video in formato MPEG.

Tabella 1.19: Alcuni esempio dei più comuni *MIME-type*.

fornire i corretti valori per le intestazioni usate per descrivere il tipo di file (le varie `Content-*` di tab. 1.7) e per gestire la negoziazione dei contenuti.

In genere questa associazione viene eseguita in base all'*estensione* del file, cioè alla parte finale del nome dello stesso che segue il carattere ".". A differenza del DOS però il numero di caratteri dell'estensione non è fisso né limitato ed inoltre i file possono avere anche più estensioni, sempre separate da punti, in tal caso l'ordine in cui le si specificano è irrilevante.

Ad esempio `index.html.it` viene considerato un file con estensione `.it` e `.html` e verrà selezionato tutte le volte che si fa riferimento ad una qualunque di queste due estensioni all'interno di una direttiva, inoltre dal punto di vista del controllo sulle estensioni `index.it.html` è del tutto equivalente a `index.html.it` dato che entrambi hanno estensione `.it` e `.html`.

Le estensioni vengono usate per associare un file ad un tipo di contenuto (come `.html`), alla lingua con cui il contenuto è stato scritto (come `.it`) ed anche al tipo di caratteri usati (ad esempio `.jis`. Si tenga presente infine che nelle direttive una estensione può essere specificata sia con il punto iniziale che senza, e che non c'è differenza fra lettere maiuscole o minuscole.

Come accennato in sez. 1.4.1 per la gestione delle associazioni fra file e *MIME-type* oltre a `mod_mime` si può usare anche il modulo `mod_mime_magic` che cerca di determinare automaticamente il *MIME-type* di un file controllandone il contenuto.[29] L'elenco delle principali direttive disponibili con questi due moduli è riportato di seguito, si tenga presente che tutte queste direttive (tranne le prime due) si possono usare, oltre che nella configurazione generica del server, in *container* di tipo `Directory` e affini, nei `VirtualHost` ed nei file `.htaccess`:

`MimeMagicFile` è l'unica direttiva definita da `mod_mime_magic`, e prende come argomento il file che contiene la mappa delle corrispondenze fra i cosiddetti *magic number* (il valore dei primi byte di un file, usato spesso per identificarne il contenuto) ed i *MIME-type* ad essi associati. La direttiva può essere usata solo a livello di configurazione del server o in un `VirtualHost`; un esempio, estratto dal file di configurazione di default usato da Debian, è il seguente:

```
<IfModule mod_mime_magic.c>
    MIMEMagicFile /usr/share/misc/file/magic.mime
</IfModule>
```

[29] per quanto il meccanismo di riconoscimento sia stato ottimizzato l'uso di questo modulo può risultare piuttosto pesante su un server con molto carico, dato che richiede l'ispezione del contenuto dei singoli file.

TypesConfig definisce da quale file leggere le definizioni dei *MIME-types* e le associazioni di default con le relative estensioni. La direttiva si applica solo a livello di configurazione generale del server, e prende come unico argomento il nome del file da utilizzare; nel file di configurazione standard installato da Debian questa ha il valore:

```
TypesConfig /etc/mime.types
```

AddLanguage permette di associare un insieme di documenti identificato da una certa estensione con una lingua (che si suppone quella in cui gli stessi sono scritti). La direttiva prende come primo argomento l'identificativo della lingua e come secondo argomento l'estensione. Si tenga presente che un file può essere associato anche a più lingue se utilizza più estensioni. Un esempio di uso di questa direttiva è:

```
AddLanguage en .en
AddLanguage it .it
```

AddCharset permette di associare un insieme di file con un set di caratteri di default, lo si usa in genere abbinato ad `AddLanguage` per definire un particolare set di caratteri da usare per quella lingua; la direttiva prende come primo argomento l'identificativo del set di caratteri[30] e come secondo argomento l'estensione del file. Un esempio è:

```
AddCharset ISO-8859-2  .iso-pl
AddCharset ISO-2022-JP .jis
```

AddType permette di associare un insieme di file identificato da una certa estensione ad un *MIME-type*, la si usa in genere per aggiungere delle corrispondenze non contemplate in `/etc/mime.types`. La direttiva prende come primo argomento la stringa che identifica il *MIME-type* (nel formato riportato nella prima colonna di tab. 1.19) e come argomenti successivi le estensioni cui associarli. Un esempio è:

```
AddType application/x-tar .tgz
AddType image/bmp .bmp
```

AddEncoding permette di associare una codifica ad un insieme di file identificato da una estensione; lo si usa in genere per i file compressi che già hanno associato un *MIME-type* relativo al contenuto originale, e che dopo la compressione che vengono identificati tramite una ulteriore estensione (il caso classico sono i `.tar.gz`). La direttiva prende come primo argomento un *MIME-encoding* e come argomenti successivi le estensioni cui associarlo. Un esempio è

```
AddEncoding x-compress Z
AddEncoding x-gzip gz tgz
```

[30] questi sono standardizzati insieme ai *MIME-type* e si trovano anche essi elencati nella pagina citata in precedenza.

DefaultType definisce quale valore usare per l'intestazione `Content-Type` tutte le volte che viene richiesto un documento per il quale non è stato possibile determinare altrimenti il *MIME-type*. La direttiva prende come argomento una stringa che indica il *MIME-type* analoga a quelle riportate nella prima colonna di tab. 1.19. Un esempio è:

```
DefaultType text/plain
```

DefaultLanguage definisce a quale lingua devono essere associati tutti i file per i quali questa non risulta definita; si possono così ad esempio marcare i contenuti di intere directory come appartenenti ad una certa lingua senza dover rinominare i file per aggiungere le opportune estensioni. La direttiva richiede un unico parametro indicante la lingua.

Il modulo `mod_mime` si limita ad impostare le associazioni fra i file e le loro caratteristiche astratte, ma per eseguire la *content negotiation* cui accennavamo in sez. 1.1.1 Apache necessita di un altro modulo, `mod_negotiation`. Si ricordi che la *content negotiation* è quel meccanismo per cui in base alle preferenze dichiarate dal client con le varie intestazioni `Accept*` (vedi tab. 1.5) il server risponde scegliendo, fra diverse alternative dello stesso documento, quella che più si avvicina a quella richiesta.

Il meccanismo della *content negotiation* è descritto in dettaglio nella documentazione di Apache, ed in particolare in una sezione ad esso espressamente dedicata che è reperibile all'indirizzo `http://httpd.apache.org/docs/content-negotiation.html`. Per quanto riguarda la configurazione si tenga presente che per attivare questa funzionalità si deve usare l'opzione `MultiViews` di `Options` (che non è attiva di default).

L'unica direttiva rilevante fornita da `mod_negotiation` è `LanguagePriority` che prende come argomento una lista di lingue, ordinata per priorità decrescente; un esempio dell'uso della direttiva, come impostato nel file di configurazione installato da Debian, è il seguente:

/etc/apache2/mods-available/negotiation.conf

```
<IfModule mod_negotiation.c>
    LanguagePriority en da nl et fr de el it ja pl pt pt-br lb ca es sv
</IfModule>
```

che stabilisce a quale delle varie lingue disponibili il server deve dare una preferenza quando il client non ne esprime una per conto suo.

Sulla base delle precedenti spiegazioni sul significato delle varie direttive, diventa allora possibile interpretare il significato del seguente estratto del file di configurazione di default installato da Debian in cui si sono riportate le parti in cui viene effettuata la impostazione delle associazioni fra i singoli file e le rispettive lingue e `MIME-type`:

/etc/apache2/mods-available/mime.conf

```
<IfModule mod_mime.c>
TypesConfig /etc/mime.types
AddType application/x-compress .Z
AddType application/x-gzip .gz .tgz
AddType application/x-bzip2 .bz2
AddLanguage am .amh
...
```

```
AddLanguage zh-TW .zh-tw
AddCharset us-ascii    .ascii .us-ascii
AddCharset ISO-8859-1  .iso8859-1  .latin1
...
AddCharset shift_jis   .shift_jis .sjis
AddHandler type-map var
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
</IfModule>
```

1.5.2 *Handler*, contenuti dinamici e CGI

Il lettore attento avrà notato che nell'estratto di configurazione di `mod_mime` visto in coda alla sezione precedente compare una direttiva, `AddHandler`, che non abbiamo ancora trattato. Anche questa è una delle direttive definite da `mod_mime` che, come accennato in sez. 1.4.1, supporta una ulteriore funzionalità rispetto a quelle trattate finora: quella che permette di associare un file ad un *handler*.[31]

Un *handler* non è altro che una rappresentazione delle modalità con cui il server esegue le sue operazioni quando gli viene richiesta una risorsa. Come accennato all'inizio di sez. 1.3.4 quando un client richiede una risorsa il caso più comune è quello in cui gli viene restituito il contenuto del file ad essa corrispondente. Questo è il lavoro svolto dal cosiddetto `default-handler`, che non solo dovrà recuperare il contenuto del file, ma dovrà anche inviarlo al client all'interno di una transazione HTTP, e generare quindi anche i codici di stato e tutte le intestazioni necessarie per comporre una risposta corretta.

Un'altra delle caratteristiche di Apache che gli conferiscono una grande flessibilità è che attraverso gli opportuni moduli possono essere installati degli ulteriori *handler* in grado di eseguire dei compiti più complessi, come l'esecuzione di script CGI (acronimo di *Common Gateway Interface*) o l'interpretazione diretta di script PHP, che tratteremo più avanti. Ogni *handler* è identificato da un nome ed associato ad un modulo che lo implementa, tranne ovviamente il `default-handler` che fa parte delle funzionalità fornite con il nucleo di Apache. In tab. 1.20 si è riportato l'elenco degli *handler* presenti nella distribuzione standard di Apache, insieme ai moduli che li implementano e ad una breve descrizione delle loro funzionalità.

Si tenga presente comunque che questi sono solo un piccolo sottoinsieme degli *handler* effettivamente disponibili, dato che molti progetti esterni ad Apache e buona parte dei linguaggi di scripting usati per le creazione di pagine dinamiche utilizzano questo meccanismo per far fornire direttamente al server le funzionalità da essi supportate.

[31] dovendo ancora spiegare cosa sono gli *handler* non aveva senso trattare `AddHandler` e le altre direttive collegate a questa funzionalità.

[33] in realtà le intestazioni `Date` e `Server` sono comunque aggiunte, non vengono invece generate le altre che devono essere scritte direttamente all'interno del file; questo inoltre deve prevedere una intestazione `Status` che specifichi il codice di stato delle risposta.

[34] le *image map* sono quei file che, all'interno delle pagine HTML, permettono di usare una immagine come mappa su cui si può cliccare per ottenere per seguire dei collegamenti; una mappa definisce quali sezioni delle immagini corrispondono ad un certo collegamento.

[34] le *type map* sono file che vengono usati dal meccanismo della *content negotiation* per presentare le possibili varianti associate alla stessa risorsa, il loro formato è descritto nella documentazione di `mod_negotiation`.

Handler	Modulo	Funzionalità
`default-handler`	interno	trasmette il contenuto del file richiesto all'interno di una transazione HTTP, provvedendo tutte le intestazioni necessarie.
`send-as-is`	`mod_asis`	trasmette direttamente il contenuto del file richiesto così come è, senza aggiungere le intestazioni.[32]
`cgi-script`	`mod_cgi`	esegue il file richiesto come script CGI.
`imap-file`	`mod_imap`	processa la risorsa richiesta come una *image map*.[33]
`server-info`	`mod_info`	associa alla risorsa una pagina dinamica contenente una panoramica completa sulla configurazione del server.
`server-status`	`mod_status`	associa alla risorsa una pagina dinamica contenente le informazioni sulle attività e le prestazioni del server.
`type-map`	`mod_negotiation`	permette di processare una *type map*.[34]

Tabella 1.20: Gli *handler* interni della distribuzione standard di Apache.

Le direttive fornite da `mod_mime` per la gestione degli *handler* sono tre; a queste però si possono aggiungere le direttive messe a disposizione da `mod_actions` che permettono all'utente di definire dei propri *handler* associandoli all'esecuzione di opportuni script CGI. Le direttive in questione sono:

`AddHandler` permette di associare un insieme di file identificato da una estensione all'*handler* che deve essere utilizzato quando uno di essi viene richiesto. La direttiva prende come primo argomento il nome dell'*handler* da utilizzare (che può essere uno di quelli interni di tab. 1.20 o uno di quelli definiti con una direttiva `Action`) e come argomenti successivi le estensioni cui applicarlo. La direttiva si può usare sia a livello generale per tutto il server, che all'interno di *container* di tipo `Directory` e affini, nei `VirtualHost` e nei file `.htaccess`. Allora per definire quali file trattare come script CGI si può usare:

```
AddHandler cgi-script .cgi .sh .pl
```

`RemoveHandler` permette di rimuovere l'associazione ad un *handler* qualunque dei i file con una certa estensione, la direttiva si può usare solo all'interno di *container* di tipo `Directory` e affini, nei `VirtualHost` e nei file `.htaccess`; ad esempio la si può usare nei file `.htaccess` per cancellare le associazioni ereditate dalle directory sovrastanti o dalla configurazione del server. La direttiva prende come argomenti l'elenco di estensioni per le quali rimuovere l'associazione.

`SetHandler` permette di impostare un *handler* di default. La direttiva prende come argomento il nome dell'*handler* o la parola chiave `None` che permette di cancellare l'effetto di una precedente `SetHandler`. La direttiva si può usare solo all'interno di *container* di tipo `Directory` e `Location` e nei file `.htaccess`, ed imposta l'*handler* che verrà utilizzato per tutte le risorse da essi identificate. Un esempio è quello in cui si abilita l'accesso alle statistiche del server fornite da `mod_status` con:

```
<Location /server-status>
  SetHandler server-status
</Location>
```

che, associando alla URI `/server-status` l'omonimo *handler* consente di ottenere la pagina con le statistiche.

`Action` permette attivare automaticamente l'utilizzo di un opportuno script CGI in corrispondenza di una certa "*azione*". Questa può essere un *MIME-type*, nel qual caso lo script sarà attivato tutte le volte che si richiede una risorsa di quel tipo, oppure una stringa. In questo secondo caso essa assume il significato di un nuovo *handler* definito dall'utente; in questo modo lo script sarà invocato per tutte le risorse a cui si è associato il nuovo *handler* con una direttiva `AddHandler`. La direttiva prende come primo argomento l'*azione* e come secondo la URI che identifica lo script CGI da usare per quella azione. Un esempio potrebbe essere:

```
Action image/gif /cgi-bin/images.cgi
Action my-handler /cgi-bin/handler.cgi
AddHandler my-handler .my
```

`Script` permette di attivare automaticamente l'esecuzione di uno script CGI in risposta a richieste effettuate con un certo metodo HTTP. La direttiva prende come argomenti una lista di metodi (espressi con i nomi di tab. 1.2 e 1.3) terminata dalla URI dello script da eseguire.

Una delle principali estensioni che usa gli *handler* è quella dei CGI (sigla di *Common Gateway Interface*) che definisce una interfaccia tramite la quale il server può invocare un programma esterno che generi i contenuti necessari. I CGI (in genere si tratta di script, ma possono essere anche programmi binari) vengono gestiti dal modulo `mod_cgi` che implementa l'*handler* `cgi-script`. Quando si richiede una risorsa gestita con questo *handler* il server esegue il programma da essa indicato, e restituisce al client il risultato dell'elaborazione da esso svolta.

Apache tratta automaticamente come CGI tutti i file che hanno un *MIME-type* di tipo `application/x-httpd-cgi` o che siano stati associati all'*handler* `cgi-script`, questo può essere ottenuto al solito con le direttive `AddType` e `AddHandler` associando i file in base alle loro estensioni, con un qualcosa del tipo:

```
AddHandler cgi-script .cgi .sh .pl
```

La configurazione usuale però prevede che gli script vengano mantenuti in una o più directory a loro riservate, dichiarate tramite la direttiva `ScriptAlias` (torneremo sugli *alias* e le redirezioni in sez. 1.6.3, dove tratteremo anche l'uso di questa direttiva); tutti i file contenuti in tali directory vengono automaticamente considerati dei CGI ed eseguiti.

Se si vogliono usare degli script CGI mantenuti al di fuori di una di queste directory occorrerà abilitarne esplicitamente l'esecuzione con la direttiva `Options` attivando per essi l'opzione `ExecCGI`; il server infatti di default non esegue nulla che non sia all'interno di una delle directory dichiarate con `ScriptAlias`, anche se i file hanno *MIME-type* `application/x-httpd-cgi` o sono associati all'*handler* `cgi-script`. Per questo motivo una delle configurazioni più comuni per l'uso degli script CGI è:

/etc/apache2/sites-available/default

```
ScriptAlias /cgi-bin /usr/lib/cgi-bin
```

Tutto questo viene fatto perché l'uso dei CGI può essere molto delicato dal punto di vista della sicurezza. Di per sé Apache si è sempre dimostrato un programma piuttosto sicuro, con un numero piuttosto limitato di vulnerabilità (gli interessati possono trovare un elenco su `http://httpd.apache.org/security_report.html`) ma ogni volta che gli si fa eseguire un programma esterno a quelle di Apache si aggiungono le vulnerabilità di quest'ultimo ed i problemi dovuti a script CGI un po' troppo trascurati dal punto di vista della sicurezza sono innumerevoli.

Per questo consentire automaticamente l'esecuzione di CGI ovunque essi si trovino non è ma una buona idea (anche se sono eseguiti con i privilegi limitati del server); restringerne l'uso a quelli presenti in una sola directory (usualmente `/usr/lib/cgi-bin`) di proprietà dell'amministratore rende molto più difficile ad un attaccante la possibilità di installare ed eseguire un CGI malevolo e consente all'amministratore un controllo molto più stretto su chi è autorizzato ad usarli.

Questo ci ricorda anche una ulteriore precauzione da prendere; dato che il server eseguirà gli script CGI per conto dell'utente e del gruppo impostati con le direttive `User` e `Group` occorrerà che essi siano eseguibili da tali utenti. Inoltre se gli script operano su altri file occorrerà di nuovo che i sopra nominati utente e gruppo abbiano privilegi di accesso sufficienti, la mancanza di questi privilegi è una delle cause più comuni di malfunzionamento.

Una volta eseguito, per generare dei contenuti, uno script CGI deve semplicemente scriverli sullo standard output; la sola precauzione richiesta è quella di far precedere il tutto dalla dichiarazione del *MIME-type* nello stesso formato della intestazione `Content-type`. In sostanza è cura del programma fornire questa intestazione (e la riga vuota successiva) ed a seguito di essa il contenuto del documento, il server si limiterà a reinviare il tutto al client all'interno della risposta.

Se la generazione in uscita è relativamente semplice, molto più complessa è la gestione del passaggio di dati in ingresso allo script CGI. Le modalità scelte dalla *Common Gateway Interface* sono sostanzialmente due, o attraverso lo standard input o attraverso delle variabili di ambiente, che sarà compito dello script analizzare e interpretare.

La prima modalità viene utilizzata quando il client esegue una richiesta con un metodo HTTP `POST` (detta per questo *POST request*);[35] in questo caso i dati inviati vengono opportunamente impacchettati dal server in un formato in cui un campo è associato ad un valore con il carattere "`=`" ed i vari campi vengono suddivisi tramite il carattere "`&`", mentre i caratteri speciali (come lo spazio ed i due precedenti) vengono specificato dal loro valore decimale preceduto da un "`%`". Si avrà allora qualcosa del tipo:

```
nome=Simone&cognome=Piccardi&hobby=penne%20stilografiche
```

e lo script li potrà leggere dallo standard input.

La seconda modalità, detta *GET request*, è quella in cui questa stessa stringa viene fornita come parte finale della URL della richiesta, separata dalla URI effettiva attraverso l'uso del

[35] queste richieste in cui si inviano dei dati sono generate dalle apposite direttive `FORM` del linguaggio HTML, queste prevedono anche la selezione della modalità di invio, in genere si usano le richieste di tipo POST per inviare quantità maggiori di dati, e per evitare che i dati della richiesta compaiano nella URL.

carattere "?". In questo caso il metodo è usato è un normale `GET`, da cui il nome ed in tal caso il server inserisce la stessa stringa all'interno della variabile di ambiente `QUERY_STRING`. Oltre a questa il server passa comunque allo script una serie di variabili di ambiente ricavate dalla richiesta, le principali delle quali sono riportate in tab. 1.21.

Variabile	Significato
`REMOTE_ADDR`	indirizzo da cui si è ricevuta la richiesta.
`REQUEST_METHOD`	il metodo usato nella richiesta che ha lanciato lo script.
`QUERY_STRING`	la eventuale stringa con i dati della FORM.
`SERVER_NAME`	nome del server.

Tabella 1.21: Principali variabili di ambiente passate ad uno script CGI.

Le variabili di ambiente sono molto importanti in quanto permettono di passare delle informazioni utili per prendere delle decisioni su cosa fare all'interno della risposta di un CGI ad un singola richiesta, per questo è possibile controllarne il contenuto e definirne altre anche in fase di configurazione, attraverso le direttive che tratteremo più avanti in sez. 1.6.1.

Un secondo impiego comune degli *handler* è quello del supporto per l'esecuzione diretta di linguaggi interpretati. Fra questi il più comune è probabilmente il PHP (acronimo ricorsivo per *PHP Hypertext Preprocessor*), un linguaggio di scripting rilasciato con licenza libera molto adatto per lo sviluppo Web e facilmente integrabile nell'HTML. Il PHP è utilizzabile su tutti i principali sistemi operativi e server web, ed è possibile avere accesso ad una quantità enorme di moduli per gestire qualunque tipo di necessità, dalla semplice gestione del testo, all'XML, all'interfaccia con i database SQL, ecc.

Nonostante possa essere usato anche per altri obiettivi, il campo di maggiore utilizzo di PHP è lo scripting server-side. Per evitare di dovere lanciare l'interprete ogni volta che si esegue uno script PHP è possibile utilizzare un un modulo apposito, distribuito insieme al linguaggio, che fa si che lo stesso venga eseguito direttamente da Apache.

Per far questo occorrerà ovviamente installare il modulo, cosa che dipende dalla propria distribuzione e poi occorrerà modificare opportunamente la configurazione di Apache perché questo venga utilizzato. Nel caso di Debian il pacchetto che contiene il modulo è `libapache-mod-php5`, ed una volta installato per abilitarlo basterà usare il comando:

```
a2enmod php5
```

Per gestire correttamente gli script PHP in modo che questi siano automaticamente interpretati occorrerà poi usare la direttiva `SetHandler` per indicare ad Apache di usare l'*handler* fornito dal modulo per tutti gli script PHP, nel caso di Debian questo viene fatto con le direttive:

/etc/apache2/mods-available/php5.conf

```
<IfModule mod_php5.c>
    <FilesMatch "\.ph(p3?|tml)$">
        SetHandler application/x-httpd-php
    </FilesMatch>
    <FilesMatch "\.phps$">
        SetHandler application/x-httpd-php-source
    </FilesMatch>
    # To re-enable php in user directories comment the following lines
    # (from <IfModule ...> to </IfModule>.) Do NOT set it to On as it
```

```
    # prevents .htaccess files from disabling it.
    <IfModule mod_userdir.c>
        <Directory /home/*/public_html>
            php_admin_value engine Off
        </Directory>
    </IfModule>
</IfModule>
```

a questo punto tutte le volte che sarà richiesto al server una pagina PHP questo eseguirà automaticamente lo script in essa contenuto.

1.5.3 Filtri, SSI e compressione dei contenuti

A partire dalla versione 2.0 Apache è stato dotato di una infrastruttura per processare in maniera generica i dati in ingresso ed in uscita dal server, attraverso i cosiddetti *filtri*. L'infrastruttura è descritta in dettaglio nella documentazione di Apache all'indirizzo `http://httpd.apache.org/docs/2.2/filter.html`, e consente di stabilire in maniera estremamente dettagliata e flessibile le operazioni che è possibile compiere.

Manca trattazione filtri, mettere qui.

Due delle funzionalità più significative svolte tramite i *filtri* sono quella delle pagine SSI (acronimo di *Server Side Include*) e la compressione automatica dei contenuti, gestite rispettivamente dai moduli `mod_include` e `mod_deflate`. Vengono gestite attraverso i filtri anche funzionalità più avanzate come la cifratura con `mod_ssl` che vedremo in sez. 1.6.4.

Le pagine SSI sono una forma elementare di pagine dinamiche in cui si possono includere alcuni comandi nelle pagine web, usando una particolare sintassi dei commenti HTML. Quando un client richiede uno di questi file esso viene analizzato ed i comandi inseriti in esso vengono interpretati da `mod_include` che si incarica di eseguire le operazioni richieste e restituire il risultato finale.

Le operazioni che si possono compiere con i *Server Side Include* sono sostanzialmente l'inclusione di file esterni, la definizione di variabili e la capacità di lanciare dei programmi. Non approfondiremo ulteriormente l'argomento dato che stiamo trattando della configurazione del server, ma una buona introduzione agli SSI è fornita insieme alla documentazione di Apache, e la si può consultare all'indirizzo `http://httpd.apache.org/docs/2.2/howto/ssi.html`; i dettagli sulle operazioni supportate sono comunque riportati anche nella documentazione di `mod_include`.

Per poter utilizzare delle pagine SSI occorre anzitutto abilitarne l'uso con la direttiva `Options`. Come visto in sez. 1.3.4 questa prevede a tale riguardo due opzioni, `Includes` e `IncludesNOEXEC` che consentono l'uso dei *Server Side Include* sui file cui esse vengono applicate. Nel primo caso si abiliterà l'uso di tutte le funzionalità delle pagine SSI, nel secondo si escluderà la possibilità di lanciare programmi esterni.

Il fatto di averne abilitato l'uso non comporta però che tutti i file vengano analizzati per eseguire i comandi, la funzionalità infatti viene fornita in forma di *filtro*, per cui occorrerà istruire il server su quali sono le pagine SSI. Come si può notare nella parte finale della configurazione di `mod_mime` in sez. 1.5 un modo per fare questo è dare a questi file una loro estensione (in genere si usa `.shtml`) e poi usare la direttiva `AddOutputFilter` per associarli al filtro `INCLUDES`. Questo è quello che viene fatto anche nella configurazione standard usata da Debian, in cui si hanno le seguenti istruzioni:

/etc/apache2/mods-enabled/mime.conf

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

in cui prima si associano i file `.shtml` al *MIME-type* `text/html` dato che comunque si tratta di file che contengono codice HTML, e poi si dice che detti file devono essere gestiti dal server attraverso il loro *filtro* specifico.[36]

Il precedente meccanismo funziona regolarmente ma necessita dell'uso di una estensione specifica, cosa che può risultare non molto gradita se si deve aggiungere la funzionalità a delle pagine esistenti e ci si vede costretti a cambiargli nome e a modificare tutti i file che fanno riferimento ad esse. Così capita spesso che per semplificarsi la vita si scelga di trattare come SSI tutti i file `.html` (nel precedente esempio basterebbe aggiungere questa estensione alla direttiva `AddOutputFilter`), il che però comporta uno spreco di risorse (ed un peggioramento delle prestazioni) in quanto il server a quel punto deve analizzare tutti i file.

Esiste allora una soluzione più elegante che è quella dell'uso della direttiva `XBitHack`, questa fa si che il server applichi automaticamente il filtro `INCLUDES` a tutti i documenti associati al *MIME-type* `text/html` che hanno il permesso di esecuzione per il proprietario attivo. La direttiva prende un solo argomento, che può essere `On`, che abilita la funzionalità, `Off` che la disabilita (il default) e `Full` che è analoga a `On` ma che se trova attivo il permesso di esecuzione per il gruppo modifica l'intestazione `Last-modified` al tempo di ultima modifica del file, in modo che i proxy possano mantenere correttamente le copie di cache.[37]

Da finire. Manca `mod_deflate`.

1.6 Le configurazioni avanzate

Tratteremo in questa sezione alcuni argomenti di configurazione più avanzata, come la gestione delle variabili di ambiente interne al server, l'autenticazione ed il controllo degli accessi, la redirezione delle richieste e la riscrittura degli indirizzi e l'uso del protocollo HTTPS per una trasmissione dei dati cifrata e sicura.

1.6.1 Le variabili di ambiente

Le variabili di ambiente rivestono una notevole importanza in vari aspetti della configurazione di Apache. Il loro utilizzo infatti non riguarda solo i CGI come visto in sez. 1.5.2, esse sono utilizzate anche dai comandi dei *Server Side Include* (vedi sez. 1.5.3), per esegure delle configurazioni condizionali (come quella che consente cosa registrare nei file di log cui abbiamo accennato in sez. 1.4.3), nel controllo degli accessi e nelle regole di riscrittura.

[36] siccome in precedenza la gestione dei *Server Side Include* era effettuata utilizzando l'*handler* `server-parsed`, per compatibilità si può anche continuare ad usare la direttiva `AddHandler` per associarli detto *handler*.

[37] il problema con gli SSI è che siccome questi creano il risultato dinamicamente il documento risulta sempre nuovo; in certi casi questo è corretto, se ad esempio si sono usati comandi che hanno un risultato diverso, ma in genere non è detto che sia così: gli SSI infatti sono usati principalmente per semplificare la gestione dei siti mettendo le parti comuni in file che vengono inclusi; in questo caso il risultato sarebbe identico e l'uso di `Full`, mantenendo il tempo di ultima modifica uguale a quello del file che ha generato la risposta, consente al proxy di accorgersi che il documento ottenuto non è cambiato e di evitare una inutile ripetizione della richiesta.

Per questo motivo Apache prevede vari moduli per la gestione delle variabili di ambiente; il più semplice è `mod_env` che ne consente una manipolazione elementare con le prime tre direttive del seguente elenco; ma le funzionalità più interessanti sono quelle fornite da `mod_setenvif` che permette di creare variabili di ambiente sulla base delle caratteristiche delle richieste del client, che poi possono essere utilizzate per configurazioni condizionali. Le direttive fornite da questi moduli sono le seguenti:

`PassEnv` specifica una lista di variabili di ambiente che possono essere passate dall'ambiente della shell che ha lanciato Apache agli script CGI o alle pagine SSI. La direttiva prende una lista di nomi delle variabili di ambiente.

`SetEnv` imposta una variabile di ambiente, che sarà passata a script CGI e pagine SSI. La direttiva prende come primo argomento il nome della variabile e come secondo argomento il valore; si ricordi che gli argomenti sono separati da spazi, se il valore è una stringa che contiene uno spazio va espressa con le virgolette.

`UnsetEnv` permette di rimuovere una o più variabili dall'ambiente. La direttiva prende come argomenti una lista delle variabili da rimuovere.

`BrowserMatch` permette di definire delle variabili di ambiente in base al valore dell'intestazione `User-Agent` ottenuta nella richiesta del client. Il primo argomento è una espressione regolare che esprime un criterio di corrispondenza applicato alla stringa ottenuta dal client, se la corrispondenza è trovata verranno eseguite le definizioni espresse in seguito (in una lista separata da spazi).

Queste hanno tre forme: se si specifica un nome sarà definita una corrispondente variabile di ambiente, se si appone al nome il carattere "`!`" la variabile di ambiente verrà rimossa e con una assegnazione si da un valore specifico ad una variabile di ambiente; un possibile esempio è il seguente:

```
BrowserMatch ^Mozilla forms jpeg=yes browser=netscape
BrowserMatch MSIE !javascript
```

dove nella prima riga si definisce la variabile *forms* e si assegna un valore a `jpeg` e a `browser` mentre nella seconda riga si rimuove la variabile `javascript`.

Si tenga conto infine che se si ha più di una corrispondenza le istruzioni vengono eseguite nella sequenza in cui le si sono scritte, e che istruzioni successive possono sovrascrivere quelle precedenti.

`BrowserMatchNoCase` analoga alla precedente, ma esegue il controllo sulla corrispondenza senza distinguere fra maiuscole e minuscole.

SetEnvIf è una generalizzazione della precedente `BrowseMatch` in cui il controllo di corrispondenza viene eseguito sul contenuto di una qualsiasi delle intestazioni fornite dal client. La direttiva prende come primo argomento il nome di una qualunque delle intestazioni nella richiesta del client (vedi tab. 1.5) oppure una delle caratteristiche della richiesta elencate in tab. 1.22. Se non corrisponde a nessuna di queste allora viene cercato fra i nomi delle variabili di ambiente già definite. In questo modo si può usare la direttiva per eseguire controlli sulla base del risultato di corrispondenze trovate in precedenza.

Variabile	Significato
`Remote_Host`	il nome a dominio di chi ha eseguito la richiesta (se riesce la risoluzione inversa).
`Remote_Addr`	l'indirizzo IP di chi ha eseguito la richiesta.
`Request_Method`	il metodo della richiesta.
`Request_Protocol`	il protocollo usato nella richiesta (vedi tab. 1.1).
`Request_URI`	la URI usata nella richiesta.

Tabella 1.22: Valori usati da `SetEnvIf` per identificare alcune caratteristiche di una richiesta.

Gli argomenti successivi sono identici a quelli di `BrowseMatch`; il secondo è una espressione regolare con cui controllare il valore del contenuto della caratteristica indicata dal primo, cui seguono le assegnazioni delle variabili nella stessa forma vista in precedenza.

SetEnvIfNoCase è sostanzialmente identica alla precedente `SetEnvIf`, ma esegue i controlli di corrispondenza senza distinguere fra maiuscole e minuscole.

Con queste direttive non solo si possono definire variabili specifiche per passare agli script CGI e alle pagine SSI tutta una serie di informazioni; il comportamento del server stesso può essere modificato attraverso alcune variabili di ambiente riservate, in modo di adattare meglio le risposte alle funzionalità (o più spesso ai limiti) dei client che hanno eseguito una richiesta. Un elenco di queste variabili riservate e del loro effetto è riportato in tab. 1.23.

1.6.2 Controllo degli accessi e meccanismi di autenticazione

Apache supporta varie modalità per imporre delle restrizioni d'accesso alle risorse che il server mette a disposizione. La forma di controllo degli accessi più semplice è quella fornita da `mod_access`, con il quale si possono discriminare gli accessi in base agli indirizzi IP, ai domini di provenienza delle richieste e, basandosi sulle funzionalità provviste da `SetEnvIf` illustrate in sez. 1.5.2, su una qualunque altra caratteristica delle richieste del client.

Il modulo supporta tre direttive, che possono essere usate sia in *container* di tipo `Directory` e affini che nei file `.htaccess`; inoltre se si vogliono imporre dei controlli solo per alcuni dei metodi HTTP (di default valgono per tutti) si possono inserire le direttive all'interno di un *container* di tipo `Limit`. Dette direttive sono:

Order permette di definire la modalità di accesso utilizzata di default stabilendo in quale ordine vengono valutate le direttive `Allow` e `Deny` presenti nello stesso *container*.

Variabile	Significato
`downgrade-1.0`	tratta forzatamente la richiesta come se fatta con HTTP/1.0 anche se il client usa un valore successivo.
`force-gzip`	se è attivo `mod_deflate` la direttiva forza la compressione delle risposte qualunque sia la richiesta del browser.
`force-no-vary`	rimuove il campo `Vary`, che non viene interpretato correttamente da alcuni client, dalle risposte.
`force-response-1.0`	forza una risposta con HTTP/1.0.
`no-gzip`	disabilita comunque la compressione dei dati con `mod_deflate` e la fornitura di contenuti codificati con `mod_negotiation`.
`nokeepalive`	disabilita le connessioni permanenti indipendentemente da quanto richiesto negli header.
`prefer-language`	con `mod_negotiation` attivo fornisce sempre la variante di linguaggio indicata dal valore.
`redirect-carefully`	richiede al server una maggior cura nell'invio di risposte di redirezione, nasce per rispondere ai problemi di alcuni client nel gestire le redirezioni (in particolare il client DAV di Windows).
`suppress-error-charset`	rimuove la specificazione del set di caratteri usato nel testo usato nelle redirezioni, cosa che con alcuni client problematici porta ad usare una versione non corretta della pagina su cui si viene rediretti.

Tabella 1.23: Variabili di ambiente riservate per la modifica del comportamento di Apache.

La direttiva prende un unico argomento che stabilisce l'ordine. Questo può avere i valori `Deny,Allow` in cui vengono valutate per prime le direttive `Deny` e l'accesso è garantito di default, oppure `Allow,Deny` in cui vengono valutate per prima le direttive `Allow` e l'accesso è negato di default. Nel primo caso ogni client che non corrisponde ad una direttiva `Deny` o corrisponde ad una direttiva `Allow` può accedere;[38] nel secondo caso invece ogni client che non corrisponde ad una direttiva `Allow` o corrisponde ad una direttiva `Deny` non può accedere.[39] Si tenga presente che le direttive vengono valutate a seconda del container in cui sono applicate secondo l'ordine illustrato in sez. 1.3.4, per cui se utilizzata in una `Location` la direttiva (essendo processata per ultima) soprassiederà quanto impostato all'interno di una `Directory`.

`Allow` consente di stabilire chi può avere accesso alle risorse definite nel *container* in cui la si è utilizzata. La direttiva richiede sempre come primo argomento la parola chiave `from`, mentre gli argomenti successivi esprimono la lista di soggetti che possono accedere; la parola chiave `all` consente l'accesso a tutti,[40] si può invece specificare un sottoinsieme di indirizzi nelle forme seguenti:

- un nome a dominio (ad esempio `truelite.it`) nel qual caso le richieste provenienti da macchine all'interno di quel dominio saranno esaudite. Il controllo è fatto per la presenza all'interno di quel dominio (cioè nel caso dell'esempio

[38] questo permette di bloccare su un insieme ampio con `Deny` e poi definire accessi più ristretti con successive direttive `Allow`.

[39] qui invece di default si bloccano tutti gli accessi e si può definire chi accede con `Allow` e imporre ulteriori restrizioni su quanto consentito con `Allow` tramite delle successive `Deny`.

[40] compatibilmente con quanto stabilito con eventuali ulteriori direttive `Order` e `Deny`.

`monk.truelite.it` avrà accesso, mentre `falsotruelite.it` no). In questo caso il server farà due risoluzioni, la prima per ricavare dall'indirizzo IP della macchina il nome della stessa, e la seconda per verificare se detto nome corrisponde davvero all'indirizzo; soltanto se queste hanno successo e se nome ed indirizzo corrispondono l'accesso sarà consentito.

- un indirizzo IP in forma *dotted decimal*; in questa forma si si possono specificare anche delle sottoreti scrivendo parzialmente l'indirizzo; ad esempio `192.168.0.` indica tutta la sottorete `192.168.0.0/24`.
- una rete indicata sia con una coppia indirizzo/netmask (ad esempio con la notazione `172.16.0.0/255.255.0.0`) che con la notazione CIDR (ad esempio con `192.168.2.0/23`).

Infine si può usare una variabile di ambiente impostata con la direttiva `SetEnvIf`, in questo caso si consente l'accesso solo qualora la variabile di ambiente richiesta sia definita, e la sintassi della condizione sarà nella forma `env=VARIABILE`.

`Deny` consente di stabilire chi non deve avere accesso alle risorse definite nel *container* in cui la si è utilizzata; gli argomenti sono identici ai precedenti già descritti per `Allow`.

Una serie di esempi di utilizzo di queste direttive per consentire diverse modalità di accesso, estratti dal contenuto dei file di configurazione installati di default su Debian, sono i seguenti:

/etc/apache2/mods-available/status.conf

```
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from 127.0.0.1 ::1
</Location>
```

in cui si blocca l'accesso alle statistiche del server solo a partire dal *localhost*, utilizzando `deny,allow` per bloccare gli accessi di e poi `Allow` consentirli solo dagli indirizzi prescelti, oppure il seguente:

/etc/apache2/sites-available/default

```
<IfModule mod_alias.c>
    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory /usr/lib/cgi-bin/>
        AllowOverride None
        Options ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
    </Directory>
</IfModule>
```

in cui si permette di accedere agli script CGI a chiunque, impostando come ordine `allow,deny` che prevede un accesso di default, ribadito con la direttiva `Allow` che di nuovo lo consente a tutti.

Come accennato si può ottenere un controllo molto più sofisticato basandosi sugli header HTTP attraverso l'utilizzo della direttiva `SetEnvIf`; in questo caso si potrà avere una configurazione del tipo:

```
SetEnvIf Referer "^http://www.dominio.tld/" local_referal
# Permette l'accesso ai browser che non mandano informazioni sul Referer
SetEnvIf Referer "^$" local_referal
<Directory /var/www/immagini>
   Order Deny,Allow
   Deny from all
   Allow from env=local_referal
</Directory>
```

in questo modo ad esempio, possiamo evitare di far utilizzare le immagini del nostro sito da altri facendo in modo che possano essere utilizzate solo dalle nostre pagine. Lo stesso metodo (con l'uso della direttiva `BrowseMatch`) può essere usato per restringere l'accesso ad alcuni `User-Agent`, come qualche *web-spider* particolare,[41] oppure per restringere l'accesso ad una directory di pagine scritte per un determinato browser.[42]

Il controllo degli accessi disponibile con `mod_access` permette solo una discriminazione sulla base degli indirizzi da cui si ricevono le richieste; il protocollo HTTP prevede però anche la possibilità di autenticare le richieste. In particolare con Apache la richiesta di una risorsa protetta causa una risposta con un codice 401 (vedi tab. 1.8), che nei client che supportano l'autenticazione comporta in genere la richiesta di immissione di username e password che poi possono essere inviati al server nelle opportune intestazioni, in modo che questo possa provvedere ad una autenticazione.

Il punto essenziale da capire in questo meccanismo è che il protocollo HTTP è senza stati, non esiste cioè niente a livello di protocollo che definisca una forma di *collegamento* dell'utente al server per cui se l'autenticazione è necessaria le relative informazioni devono essere inviate per qualunque richiesta si faccia. Dato che è oltremodo scomodo dover reimmettere le informazioni per ogni richiesta è in genere cura del client eseguire questa operazione automaticamente una volta che le si sono immesse la prima volta. Questo però comporta anche che non esiste niente di analogo ad un *logout*, ed è di nuovo compito del client decidere come e per quanto tempo deve mantenere memorizzate le informazioni di autenticazione.

Le modalità di autenticazione sono sostanzialmente due, `Basic` e `Digest`, dai rispettivi valori della direttiva `AuthType` che permette di scegliere quale usare. La differenza consiste sostanzialmente nel fatto che mentre nel primo caso le password vengono trasmesse in chiaro al server nelle intestazioni, nel secondo caso sulla rete passa soltanto un *digest* della password che non consente di ricavarne il valore.

In realtà dal punto di vista della sicurezza dell'autenticazione entrambi i metodi sono inefficaci, dato che la trasmissione dei dati avviene comunque in chiaro per cui chiunque potrebbe intercettarli. Allora, anche se nel secondo caso non si dispone della password in chiaro, essendo comunque il confronto fatto con il digest, il possesso di quest'ultimo (che può essere intercettato)

[41] si chiamano *web-spider* quei programmi, in genere utilizzati dai motori di ricerca, che eseguono scansioni dei siti web, usualmente per acquisirne e classificarne i contenuti.

[42] si tenga presente che, oltre ad essere una pessima politica restringere inutilmente l'accesso al proprio sito, essendo le intestazioni fornite direttamente dal client, è relativamente facile per quest'ultimo falsificarle.

darebbe ugualmente accesso alle risorse protette. Per questo motivo l'unico modo per avere la sicurezza dell'autenticazione è quello di usare il supporto per SSL/TSL (vedi sez. 1.6.4), in cui tutti i dati vengono cifrati.

Per richiedere l'accesso con autenticazione il server dispone di due direttive di base, `AuthName` e `AuthType` da utilizzare all'interno di container di tipo `Directory` o affini, o anche nei file `.htaccess`, nel qual caso occorrerà che per la directory in questione sia stata consentita la sovrascrittura delle politiche di autenticazione con:

```
AllowOverride AuthConfig
```

La prima direttiva definisce il nome associato all'ambito (il cosiddetto *realm*) su cui vale l'autenticazione che deve essere unico per ciascun gruppo di credenziali. È questo nome che viene usato dai client per indicare agli utenti quali credenziali devono utilizzare; esso viene specificato come unico argomento della direttiva e se contiene spazi deve essere protetto con le virgolette. La direttiva `AuthType` invece specifica, come accennato, la modalità di autenticazione, e prevede un unico argomento che può essere una delle due parole chiave `Basic` o `Digest`, che sono le uniche due modalità attualmente supportate.

Per poter poi eseguire effettivamente l'autenticazione Apache deve ricorrere ad un opportuno modulo di supporto; il più comune è `mod_auth`, che definisce una ulteriore serie di direttive volte a specificare le modalità con cui si potrà accedere ai dati relativi alle credenziali degli utenti che nel caso sono mantenuti su semplici file di testo analoghi a quelli utilizzati nel sistema; queste direttive sono:

`AuthUserFile` permette di specificare il file in cui vengono mantenute le informazioni relative agli utenti (i loro nomi e le relative password). La direttiva prende come argomento il pathname al file su cui sono memorizzate dette informazioni; se il pathname non è assoluto questo viene risolto a partire dalla `ServerRoot`.

`AuthGroupFile` permette di specificare il file in cui vengono mantenute le informazioni relative ai gruppi (i loro nomi e gli utenti che vi appartengono). Come per `AuthUserFile` prende come argomento il pathname del file.

L'uso di di queste direttive presuppone che si disponga dei file in un formato appropriato; nel caso del file dei gruppi il formato è banale; ogni riga definisce un gruppo, essa deve iniziare con il nome del gruppo seguito da ":" a cui segue una lista dei nomi degli utenti separati da spazi; un possibile esempio del contenuto di questo file è il seguente:

htgroup.file

```
admin: piccardi cgabriel
author: csurchi
```

Il formato del file degli utenti è invece più complesso, il server infatti richiede che le password siano fornite in forma cifrata sia che si utilizzi la autenticazione semplice che quella in *digest*. Per questo motivo con Apache vengono distribuiti degli appositi programmi, `htpasswd` e `htdigest`, che possono essere utilizzati per creare e manipolare questi file.

Il comando `htpasswd` prevede a seconda dei casi da uno a tre argomenti. A meno di non usare l'opzione `-n` (che fa scrivere il risultato sullo standard output) il comando richiede come primo

Opzione	Significato
`-b`	richiede l'immissione della password direttamente sulla riga di comando.
`-c`	crea il file delle password, se questo esiste già viene troncato e ricreato da capo.
`-n`	scrive i risultati del comando sullo standard output.
`-m`	crea la password usando l'algoritmo MD5, che è utilizzabile da Apache su tutte le piattaforme .
`-d`	crea la password usando la funzione `crypt` di Unix (in sostanza un DES).
`-s`	crea la password usando l'algoritmo SHA.
`-p`	salva la password come testo in chiaro.

Tabella 1.24: Le principali opzioni del comando `htpasswd`.

argomento il nome del file, seguito dal nome dell'utente; se poi si specifica anche l'opzione `-b` la password deve essere specificata direttamente (come ultimo argomento) sulla riga di comando, altrimenti viene letta dal terminale senza essere mostrata.

Se si usa l'opzione `-c` (incompatibile con `-n`) il file viene creato, e vi viene inserito l'utente specificato; altrimenti l'utente viene aggiunto se non è già presente, o ne viene modificata la password altrimenti. Le varie opzioni del comando (comprese quelle relative all'algoritmo con cui sono generate le password) sono riportate in tab. 1.24. Così ad esempio per creare il file ed inserire il primo utente si eseguirà:

```
monk:~# htpasswd -c /etc/apache/passwd piccardi
New password:
Re-type new password:
Adding password for user piccardi
```

mentre poi per aggiungerne un altro si potrà eseguire:

```
monk:~# htpasswd /etc/apache/passwd csurchi
New password:
Re-type new password:
Adding password for user csurchi
```

ed il file conterrà per ciascuna riga un nome utente e la corrispondente password cifrata separati dal carattere "`:`", un esempio di questo file potrebbe essere qualcosa del tipo:

/etc/apache/passwd

```
piccardi:Gin6Rlb7mGvoQ
csurchi:d82x4sT/niKDk
```

Il comando `htdigest` invece prevede come unica opzione `-c` che ha lo stesso significato che assume per `htpasswd`, il comando richiede sempre tre argomenti, il primo per indicare il file su cui operare, il secondo per identificare il *realm* cui appartiene l'utente, ed infine il nome di quest'ultimo.

Quale dei due diversi formati per i file degli utenti occorra utilizzare dipende dalla modalità di autenticazione scelta. Dato che i file usati da `mod_auth` contengono informazioni sensibili (anche se le password sono cifrate è possibile tentare un attacco a dizionario) è sempre opportuno porli al di fuori dell'albero dei documenti serviti da Apache onde evitare che qualcuno possa scaricarli;

non vanno posti neanche nelle directory per le quali si restringono gli accessi, perché un utente autorizzato finirebbe con l'ottenere anche i dati degli altri utenti.

Argomento	Significato
`user`	richiede che l'utente sia uno di quelli elencati negli argomenti successivi (prevede una lista di nomi di utente separati da spazi).
`group`	richiede che l'utente appartenga al gruppo specificato.
`valid-user`	richiede che ci sia un qualunque utente autenticato.
`file-owner`	richiede che l'utente corrisponda all'utente di sistema proprietario del file.
`file-group`	richiede che l'utente appartenga al gruppo con lo stesso nome del gruppo proprietario del file.

Tabella 1.25: I valori definiti come primo argomento della direttiva `Require`.

Una volta impostati i file dai quali si possono ottenere le credenziali di autenticazione, una ulteriore direttiva, `Require`, permette di specificare ulteriormente quali fra gli utenti autenticati possono avere accesso ad una specifica risorsa. In questo modo è possibile utilizzare gli stessi file con nomi utente e password, ma restringere ulteriormente l'accesso ad alcune risorse del nostro server.

La direttiva deve essere specificata insieme alle altre e prende come primo argomento il nome della proprietà con la quale viene eseguita la selezione ulteriore, ad esempio con `user` si possono scegliere quali degli utenti possono accedere (specificandone la lista negli argomenti successivi), mentre con `valid-user` si richiede la semplice presenza di un utente autenticato; un elenco dei possibili argomenti per `Require` è illustrato in tab. 1.25.

Un esempio di configurazione di due diversi tipi di *container* per l'utilizzo delle funzionalità di autenticazione provviste da `mod_auth` è allora il seguente:

```
<Location /private>
    AuthName "Area riservata"
    AuthType "Basic"
    AuthUserFile /etc/apache/passwd
    AuthGroupFile /etc/apache/group
    Require valid-user
</Location>
...
<Directory /var/www/truelite.it/simone>
    AuthName "Materiale privato"
    AuthType "Digest"
    AuthUserFile /etc/apache/passwd
    AuthGroupFile /etc/apache/group
    Require user piccardi
</Directory>
```

Infine in tutti i casi in cui si vuole utilizzare un sistema di autenticazione basato su nome utente e password ed allo stesso tempo eseguire un un controllo su IP o nome della macchina di provenienza della richiesta, è possibile utilizzare la direttiva `Satisfy` che prende come unico argomento uno dei due valori `any` o `all`. In questo caso qualora si siano specificate delle restrizioni

sia tramite `Allow` che tramite `Require` si può richiedere che l'accesso sia consentito a chi ha superato almeno uno dei due controlli con `any`, o che debbano essere superati entrambi con `all`.

1.6.3 Alias, redirezione e riscrittura degli indirizzi

Abbiamo già accennato all'inizio di sez. 1.3.4 come le URI delle richieste HTTP non corrispondano necessariamente ad file. Le URI infatti esprimono un pathname nell'albero astratto dei documenti web messi a disposizione del server, e anche se il caso più comune è quello in cui questi sono direttamente corrispondenti a dei file in una sezione dell'albero del filesystem (quella definita da `DocumentRoot`) non sempre è così. Abbiamo già visto in sez. 1.5.2 come ad esempio sia possibile ottenere le statistiche del server facendo riferimento a delle URI (come `/server-status`) che non corrispondono a dei file ma sono generate dinamicamente, o come una sezione delle URI possa essere usate per passare dei parametri da uno script CGI.

Una delle altre funzionalità messe a disposizione di Apache è quella di rimappare a piacere diverse parti dell'albero dei documenti web su quello dei file, in modo che sia possibile configurare il server in maniera estremamente flessibile per quanto riguarda l'associazione fra un documento richiesto da un client con una URI e la risorsa locale (file, script o altro) cui su vuole accedere. Per far questo sono disponibili tre diversi meccanismi:

aliasing alla richiesta del client di una specifica risorsa nella URI, il server fa corrispondere in modo trasparente una diversa risorsa locale.

redirecting alla richiesta di una risorsa il client riceve una notifica sul nuovo indirizzo della stessa e viene redirezionato per eseguire di nuovo la richiesta.

rewriting la URL fornita dal client viene parzialmente o completamente riscritta, e la risorsa corrispondente viene restituita.

Le funzionalità che permettono di realizzare l'*aliasing* sono fornite da `mod_alias`; le direttive che possono essere usate a questo scopo sono elencate di seguito e possono essere applicate o a livello di server o per un *Virtual Host*:

`Alias` permette di mantenere di far accedere documenti posti in qualsiasi directory nel filesystem locale, anche al di fuori della `DocumentRoot`, in sostanza si può mappare una sezione del pathname espresso nella URI in una directory sul filesystem locale. La direttiva prende come primo argomento la sezione di URI da rimappare e come secondo argomento la directory dove questa deve essere mappata. Un possibile esempio è il seguente:

```
Alias /icons/ /usr/share/apache/icons/
Alias /images/ /usr/share/images/
```

in cui ogni risorsa che nella URI fa riferimento a `/icons/` viene cercate nella directory locale `/usr/share/apache/icons/`. Si tenga presente che se il primo parametro termina con una `/` il server richiede che essa sia presente, nel caso in esempio questo significa che la URI `/images` non verrà rimappata, mentre `/images/mozilla.png` si.

AliasMatch è equivalente a `Alias` ed ha gli stessi effetti, ma invece di richiedere come primo argomento una sezione di pathname, supporta l'uso di espressioni regolari, e consente il riutilizzo di ogni sezione che ha avuto corrispondenza[43] nel secondo argomento. In questo modo è possibile rimappare gruppi di file in maniera molto flessibile (anche se non quanto con `mod_rewrite`). Un esempio che riproduce lo stesso effetto del precedente di `Alias` è:

```
AliasMatch ^/icons(.*) /usr/share/apache/icons$1
AliasMatch ^/images(.*) /usr/share/images$1
```

ScriptAlias la direttiva definisce una directory su cui sono mantenuti gli script CGI (si ricordi quanto detto in sez. 1.5.2). È identica ad `Alias`, ed ha gli stessi effetti, in più marca tutti i file presenti nella destinazione come script CGI e ne abilita l'esecuzione con l'*handler* `cgi-script`.

ScriptAliasMatch è la corrispettiva di `AliasMatch` per `ScriptAlias`, permette la selezione in base ad una espressione regolare; ad esempio se i CGI scritti in Perl sono identificati dall'estensione `.pl` e sono posti in una directory separata si potranno vedere direttamente sotto `/cgi-bin` con una impostazione del tipo:

```
ScriptAliasMatch /cgi-bin/(.*)\.pl$ /usr/lib/cgi-bin/perl/$1.pl
```

Un esempio dell'uso di queste direttive si trova nel seguente estratto del file di configurazione del dominio virtuale di default, installato da Debian (da cui al solito si sono rimossi i commenti):

default

```
...
    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory /usr/lib/cgi-bin/>
        AllowOverride None
        Options ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
    </Directory>
...
    Alias /doc/ "/usr/share/doc/"
    <Directory "/usr/share/doc/">
        Options Indexes MultiViews FollowSymLinks
        AllowOverride None
        Order deny,allow
        Deny from all
        Allow from 127.0.0.0/255.0.0.0 ::1/128
    </Directory>
...
```

[43] secondo la sintassi classica delle stesse (vedi sez.**??**) in cui quanto corrisponde ad una sezione marcata fra parentesi tonde può essere in seguito riutilizzato con le espressioni `$1`, `$2`, ecc.

e si noti come, essendo le directory specificate con le direttive `Alias` al di fuori della `DocumentRoot`, a ciascuna di esse segua un *container* di tipo `Directory` in cui vengono effettuate le opportune impostazioni di base per coprire le risorse situate nella destinazione dell'*alias*.

Un secondo gruppo di direttive fornite da `mod_alias` sono quelle che possono essere usate per la *redirezione*; mentre per l'*aliasing* il cambiamento della collocazione della risorsa è completamente trasparente al client, con questo meccanismo possiamo notificare al client un cambiamento di URL, usando gli opportuni codici di ritorno HTTP (vedi sez. 1.1.4).

Valore	Significato
`permament`	redirezione permanente (associato al codice di stato 301).
`temp`	redirezione temporanea (associato al codice di stato 302).
`seeother`	la risorsa è stata rimpiazzata (associato al codice di stato 303).
`gone`	risorsa non più disponibile (associato al codice di stato 410).

Tabella 1.26: I valori di controllo per lo stato della redirezione da usare come primo argomento della direttiva `Redirect`.

In questo caso la direttiva principale che abbiamo a disposizione è `Redirect`, questa prende sempre tre argomenti, il primo indica il tipo di redirezione (in sostanza quale codice di stato deve essere restituito al client) il secondo la URI della risorsa, ed il terzo la URL completa a cui si viene rediretti, nella forma:

```
Redirect [stato] URI URL
```

in tab. 1.26 si sono riportati i valori possibili per il primo argomento (se non lo si specifica si sottintende il valore `temp`); un possibile esempio di uso di questa direttiva è il seguente:

truelite.conf
```
Redirect permanent /notizie http://www.truelite.it/news/old/
```

in cui si redirigono su una nuova URL le richieste indirizzate ad una sezione del sito che si è spostata.

Oltre a `Redirect` l'uso di `mod_alias` mette a disposizione una serie di altre direttive di redirezione, due di queste, `RedirectPermanent` e `RedirectTemp`, sono equivalenti all'uso di `Redirect` con gli omonimi valori per il primo argomento; l'unica che introduce delle funzionalità aggiuntive è `RedirectMatch`, che ha scopo analogo a quello di `Redirect` ma esegue la redirezione sfruttando delle espressioni regolari; anche `RedirectMatch` prende tre argomenti; il primo argomento è identico a quello di `Redirect` ed usa gli stessi valori di tab. 1.26, il secondo è una espressione regolare che seleziona le risorse per le quali effettuare la redirezione, la cui destinazione è espressa dal terzo, con una sintassi analoga a quella di `AliasMatch`. Così se si sono spostati i file compressi in una directory a parte si potrà effettuare una redirezione con qualcosa del tipo:

esempio.conf
```
RedirectMatch temp (.*)\.(zip|gz)$ http://www.dominio.tld/archivi/$1.$2
```

Il terzo meccanismo, quello del *rewriting*, è simile come concetto all'*aliasing* in quanto avviene in maniera trasparente al client, ma è molto più potente in quanto permette di rimappare

qualsiasi richiesta su una corrispondente risorsa locale. Le funzionalità vengono messe a disposizione da un apposito modulo, `mod_rewrite`, che attiva il cosiddetto *motore di riscrittura*; questo permette di eseguire manipolazioni estremamente complesse[44] attraverso una serie di regole e condizioni ad esse allegate, utilizzando criteri che coinvolgono anche parametri diversi, come l'autenticazione, l'indirizzo di provenienza, o il browser utilizzato.

La prima direttiva da utilizzare per l'uso di `mod_rewrite` è `RewriteEngine`, che attiva o disattiva il motore di riscrittura a seconda del valore del suo argomento che può essere solo `on` o `off`; viene in genere utilizzata per evitare di dover commentare tutte le regole di riscrittura quando le si vogliono disabilitare.

Le direttive principali per l'uso di `mod_rewrite` sono due, `RewriteRule` e `RewriteCond`. In genere infatti il motore di riscrittura viene utilizzato attraverso una serie di criteri di selezione, specificati con `RewriteCond`, che identificano le ulteriori proprietà delle richieste per le quali si applicano le regole di riscrittura specificate dalla susseguente `RewriteRule`. Una forma comune dell'uso delle direttive di `mod_rewrite` è pertanto:

```
RewriteCond  espressione_da_controllare espressione_di_controllo
RewriteCond  espressione_da_controllare espressione_di_controllo
...
RewriteRule URL_da_riscrivere URL_riscritta  [flags]
...
```

Il meccanismo di scansione delle regole di riscrittura, illustrato in fig. 1.7, come si può notare è molto complesso; una prima difficoltà sta nel fatto che storicamente le condizioni vengono scritte prima della regola di riscrittura cui si applicano, ma in realtà vengono controllate soltanto se la risorsa richiesta è fra quelle indicate da quest'ultima. Le regole poi vengono processate nell'ordine in cui sono scritte, e le regole successive alla prima vedranno non la risorsa originariamente richiesta, ma il risultato delle precedenti riscritture, ed occorre tenerne conto.

Inoltre una singola regola di riscrittura ha effetto solo se tutte le condizioni espresse nelle corrispondenti `RewriteCond` vengono soddisfatte (se non ve ne sono ha effetto immediato) che di nuovo vengono processate sequenzialmente. Infine sia nella selezione operata da `RewriteRule` che nelle condizioni specificate da `RewriteCond` si possono usare delle espressioni regolari ed in ogni argomento delle suddette direttive si possono richiamare i risultati di corrispondenze relative a espressioni controllate in precedenza, complicando all'inverosimile la situazione.

La direttiva `RewriteCond` prende due argomenti; il primo argomento è una stringa che indica un testo a cui si applica l'espressione di controllo data dal secondo argomento. Oltre al testo semplice, che viene interpretato letteralmente, il primo argomento supporta l'utilizzo di una serie di espressioni addizionali che vengono espanse ed il cui valore viene poi confrontato con il criterio di corrispondenza espresso nel secondo argomento, queste sono:

`$N` il valore dell'`N`-simo *subpattern* (con `N` compreso fra 1 e 9) usato nell'espressione di selezione della susseguente `RewriteRule` a cui la condizione fa riferimento.

`%N` il valore dell'`N`-simo *subpattern* (con `N` compreso fra 1 e 9) usato all'interno di espressioni di controllo nelle precedenti condizioni.

[44] la complessità però ha un costo: come riportato nell'introduzione alla sua documentazione sul sito di Apache, `mod_rewrite` viene considerato "*voodo*".

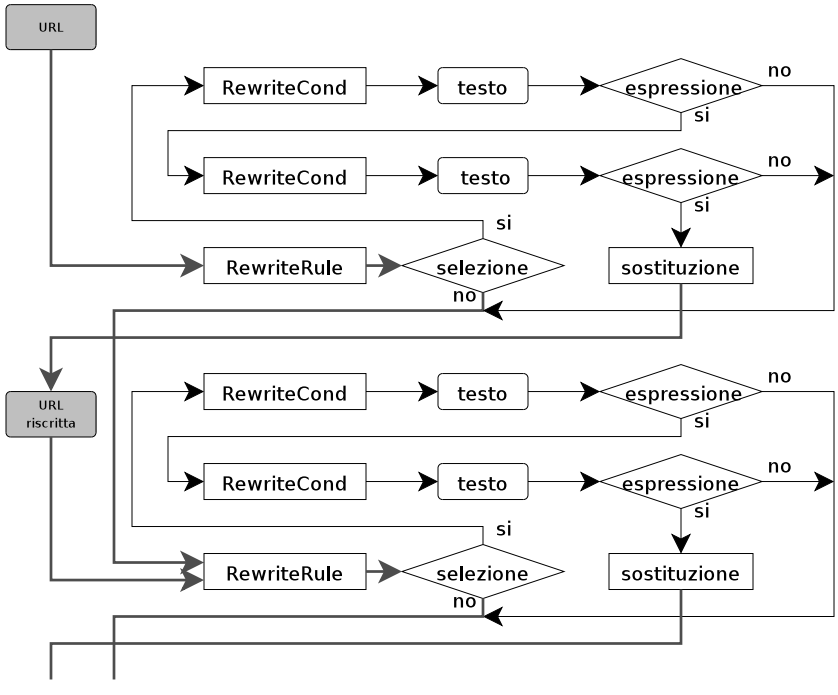


Figura 1.7: Schema del motore di riscrittura delle URL di `mod_rewrite`.

`%{VAR}` il valore di una delle variabili definite internamente dal server, un breve elenco delle più importanti sono state riportate in tab. 1.27; per l'elenco completo si consulti la documentazione online di `mod_rewrite` già citata in precedenza.[45]

`${map:key}` il valore ottenuto dalla mappa `map` (definita da una direttiva `RewriteMap`) in corrispondenza della chiave con valore `key`.

Il secondo argomento di `RewriteCond` è di norma una espressione regolare che viene applicata al risultato dell'espansione del primo argomento; la condizione ha successo qualora ci sia corrispondenza. Oltre alle espressioni regolari classiche però la direttiva prevede anche l'uso di alcune estensioni specifiche. La prima di queste è l'uso del carattere "`!`" come prefisso, che permette di invertire la condizione e chiedere la non corrispondenza con l'espressione regolare usata; oltre a questa si possono usare le ulteriori sintassi illustrate in tab. 1.28.

La direttiva supporta anche un terzo argomento opzionale, da specificare fra parentesi quadre, che permette di modificare il comportamento della condizione. Questo può prendere solo due valori; con `NC` si richiede che tutte le corrispondenze siano *case insensitive* mentre con `OR` si

[45] oltre a quella citata esiste un sintassi estesa per alcune variabili particolari, ad esempio con `%{ENV:nome}` si richiede il valore della corrispondente variabile di ambiente, mentre con `%{HTTP:header}` si richiede la corrispondente intestazione generica del protocollo; anche questi dettagli si trovano nella documentazione di `mod_rewrite`.

Valore	Significato
HTTP_USER_AGENT	il valore dell'intestazione Referer della richiesta.
HTTP_REFERER	valore dell'intestazione Referer della richiesta.
HTTP_HOST	il valore dell'intestazione Host della richiesta.
REMOTE_ADDR	l'indirizzo IP della richiesta.
REMOTE_HOST	il nome della macchina che ha eseguito la richiesta.
REMOTE_USER	il nome dell'utente (ottenuto dalle informazioni di autenticazione) che ha eseguito la richiesta.
TIME	il tempo di sistema.
TIME_DAY	il giorno del mese.
TIME_HOUR	l'ora.
THE_REQUEST	il testo completo della richiesta HTTP (come illustrato in fig. 1.4) .

Tabella 1.27: Alcune variabili del server utilizzabili nelle condizioni di controllo di RewriteCond.

possono combinare più direttive fra loro con un OR logico (cioè richiedere che ne sia soddisfatta almeno una) invece che con l'AND implicito nel meccanismo illustrato in fig. 1.7.

Espressione	Significato
<testo	esegue un confronto lessicale con il testo ottenuto dal primo argomento per vedere se è minore (in ordine alfabetico) della stringa testo.
>testo	esegue un confronto lessicale con il testo ottenuto dal primo argomento per vedere se è maggiore (in ordine alfabetico) della stringa testo.
=testo	esegue un confronto lessicale del testo ottenuto dal primo argomento che deve corrispondere esattamente alla stringa testo.
-d	tratta il testo ottenuto dal primo argomento come il nome di una directory e ne richiede l'esistenza.
-f	tratta il testo ottenuto dal primo argomento come il nome di un file e ne richiede l'esistenza.
-s	tratta il testo ottenuto dal primo argomento come il nome di un file e ne richiede l'esistenza e che non abbia dimensione nulla.
-l	tratta il testo ottenuto dal primo argomento come il nome di un file e richiede che sia un link simbolico.
-F	controlla se il testo ottenuto dal primo argomento corrisponde al nome di un file accessibile nella configurazione corrente del server.
-U	controlla se il testo ottenuto dal primo argomento corrisponde ad una URL valida nella configurazione corrente del server.

Tabella 1.28: Espressioni speciali utilizzabili come secondo argomento di RewriteCond.

Le direttiva RewriteRule è quella che definisce ogni regola di riscrittura; come accennato il primo argomento è una espressione regolare che seleziona le URL per le quali viene utilizzato il motore di riscrittura. Questa è una espressione regolare in standard POSIX con una eccezione, l'uso del carattere "!" per selezionare le URL che *non* corrispondono.[46]

[46]si faccia attenzione se si usa questa funzionalità a non utilizzare riferimenti successivi (con $N) al contenuto di eventuali *subpattern*: avendo scelto le URL che non corrispondono questi ovviamente non avranno nessun

Il secondo argomento della direttiva è il valore della URL riscritta (si ricordi che la riscrittura avviene solo se tutte le condizioni sono soddisfatte), ed in questo si possono utilizzare le stesse espressioni per il primo argomento di `RewriteCond`. Infine si può usare il valore speciale "`-`" per indicare di non eseguire nessuna sostituzione.[47]

Infine la direttiva supporta un terzo argomento opzionale che esprime una lista (separata da virgole e racchiusa in parentesi quadre) di flag che permettono di modificare le modalità con cui la regola viene applicata. Questi possono essere indicati sia in forma breve che estesa; un elenco dei principali flag disponibili è illustrato in tab. 1.29, per l'elenco completo si faccia riferimento alla documentazione di `mod_rewrite`.

Flag		Significato
R	redirect	sostituisce tutta la URI, e forza una redirezione, il flag può (se specificato nella forma `R=CODE`) specificare il codice HTTP della redirezione (vedi sez. 1.1.4) che altrimenti è pari a 301 (redirezione temporanea).
F	forbidden	forza una risposta di proibizione (codice HTTP 403, vedi sez. 1.1.4), serve a bloccare delle URL selezionate grazie alle varie `RewriteCond`.
G	gone	forza un codice HTTP 410 marcando le URL corrispondenti come riferimenti a risorse non più esistenti.
L	last	termina immediatamente il processo di riscrittura, non eseguendo le eventuali `RewriteRule` successive. Si utilizza per evitare che le URL riscritta venga ulteriormente riscritta successivamente da altre regole.
N	next	fa ripartire da capo il processo di riscrittura rieseguendo il procedimento a partire dalla prima regola (attenzione a non creare cicli infiniti).
C	chain	*incatena* la regola alla successiva, in caso di successo non cambia nulla, ma in caso di fallimento la regola successiva non viene eseguita.
NC	nocase	esegue il controllo di corrispondenza in modalità *case insensitive*.

Tabella 1.29: Principali flag utilizzabili come terzo argomento di `RewriteRule`.

Vediamo allora alcuni esempi dell'uso del motore di riscrittura, il primo caso è una estensione generalizzata delle funzionalità di *aliasing*, ad esempio si possono redirigere le richieste delle icone su una directory diversa con una qualcosa del tipo di:

```
RewriteRule /icons/(.*) /usr/share/apache/icons/$1 [R=permanent,NC]
```

Un altro esempio è il seguente in cui si utilizzano le variabili di tab. 1.27 per effettuare la redirezione delle connessioni eseguite da un certo utente quando è in una sottorete ad una sezione diversa dell'albero:

```
RewriteCond %{REMOTE_ADDR} ^192\.168\.1\.
RewriteCond %{REMOTE_USER} piccardi
RewriteRule ^/$ /locale/ [L]
```

contenuto.

[47] può sembrare assurdo, ma questo è usato in combinazione con il flag `C` per eseguire ricerche su più pattern.

ed in questo caso la regola di riscrittura viene eseguita soltanto se l'indirizzo IP del client e l'utente remoto corrispondono a quanto richiesto; il precedente esempio può essere modificato per avere la redirezione per una sola delle due condizione riscrivendolo come:

```
RewriteCond %{REMOTE_ADDR} ^192.168\.1\.1 [OR]
RewriteCond %{REMOTE_USER} piccardi
RewriteRule ^/$ /locale/ [L]
```

Infine possiamo gestire direttamente con le regole di riscrittura una situazione in cui ci sono un gran numero di domini virtuali, evitando l'uso di `VirtualHost` e procedendo alla riscrittura diretta delle richieste con qualcosa del tipo:

```
RewriteCond %{HTTP_HOST} ^www\.(.+)$
RewriteRule ^/(.+)$ /var/www/virtualhosts/%1/$1 [L]
```

in cui prima si estrae dalla richiesta l'indirizzo richiesto, e poi si riscrive la URL per indirizzare correttamente il client alla directory in cui si ci sono i rispettivi file; in questo modo ad esempio si otterrà che l'indirizzo `www.truelite.it` vada a puntare ai file contenuti nella directory `/var/www/virtualhosts/truelite.it`.

Come accennato in precedenza una delle possibili modalità di sostituzione che si possono usare all'interno di una `RewriteCond` è quello dell'uso di opportune mappe, con la sintassi `${map:key}`, che consentano una mappatura manuale fra tanti diversi possibili valori per la chiave `key` che sarà sostituita da un corrispondente valore ottenuto dalla mappa stessa. Una mappa si crea con la direttiva `RewriteMap`; questa prende come primo argomento il nome della mappa (`map` nel caso dell'esempio) che definisce e come secondo argomento la specificazione del file che contiene la mappa nella forma:

```
tipo:/pathname/file
```

dove `tipo` indica il tipo di mappa e può essere uno dei valori di tab. 1.30, cui deve seguire il pathname del file da cui i dati della mappa vengono ottenuti.

1.6.4 Gestione del protocollo HTTPS

Una delle caratteristiche di base del protocollo HTTP è che i dati vengono trasmessi in chiaro sulla rete; fintanto che il protocollo viene usato per accedere ad informazioni pubbliche questo non è un problema, ma tutte le volte che si deve accedere in maniera privata, o autenticare un utente, si presenta il problema di come evitare che i dati possano essere intercettati da terzi.

Per questo motivo a suo tempo Netscape creò un apposito protocollo, denominato SSL (da *Secure Socket Layer*, che consente di creare un canale cifrato sul quale fare passare i dati che usualmente si farebbero passare in chiaro su un normale socket.[48] L'immediata applicazione di questo protocollo è stata quella di creare HTTPS, che non è altro che il normale protocollo HTTP fatto passare su SSL. La standardizzazione di questo protocollo ha provveduto ad assegnare ad esso la porta 443, mentre il suo utilizzo da parte di un browser viene segnalato richiedendo una URL che utilizzi come indicatore di protocollo la stringa `https://`.

[48] per una trattazione estesa del protocollo e delle relative problematiche si consulti la sez. 2.1 di [SGL].

Tipo	Descrizione
`txt`	File di testo, che contiene una corrispondenza `chiave valore` per ciascuna riga. Il formato prevede che tutto quello che segue il carattere "`#`" sia ignorato e che liste di valori possano essere specificate nella forma `valore1|valore2|valore3`.
`dbm`	Analogo al precedente, ma i contenuti vengono mantenuti all'interno di un database DBM che consente una ricerca molto più veloce.
`int`	permette di accedere ad alcune funzioni interne (come `tolower`, per l'elenco si consulti la documentazione citata in precedenza) che operano direttamente sul valore della chiave.
`prg`	esegue come programma il file passato come parametro (che deve essere eseguibile) che comunica con il server attraverso lo standard input (su cui viene scritta la chiave come stringa terminata da un carattere di a capo) e lo standard output (da cui riceve il corrispondente valore nella stessa forma, o la stringa `NULL` qualora non ci sia corrispondenza).

Tabella 1.30: Vari tipi di mappe definibili con la direttiva `RewriteMap`.

Per supportare l'uso di HTTPS con Apache 1.3 si avevano due alternative. La prima era quella di utilizzare una apposita versione del demone, *Apache-SSL*, compilata con il supporto del protocollo, da mettere in ascolto direttamente sulla porta 443. La seconda, la sola rimasta disponibile anche nelle versioni successive ed in uso ancor oggi, era quella di utilizzare un modulo apposito, `mod_ssl`, che fornisce ad Apache il supporto per SSL (v2/v3) e TLS (*Transport Layer Security*).

Lo sviluppo del modulo è partito nel 1998 dal codice scritto per Apache-SSL e con la versione 2 di Apache `mod_ssl` è diventato parte della distribuzione ufficiale e viene installato di default dalle principali distribuzioni ed opera in sostanza come un filtro. Si tenga presente però che il modulo viene solo installato, in genere deve essere attivato esplicitamente con le modalità illustrate in sez. 1.4.1.

Le direttive di `mod_ssl` dono divise in tre classi principali; le *Global directives*, che possono essere utilizzate solo a livello di configurazione generale del server (e quindi al di fuori di ogni *container*), le *Per-Server directives* che devono essere usate sia a livello di configurazione generale che per i singoli *Virtual Host* e le *Per-Directory* che possono essere utilizzate ovunque. Non tratteremo qui i concetti relativi al meccanismo di funzionamento di SSL/TLS e della gestine di chiavi e certificati, che daremo per noti,[49] limitandoci ad esaminare le direttive di Apache per la gestione degli stessi.

La documentazione completa di tutte le direttive è disponibile direttamente insieme ad un "*howto*" specifico insieme al resto della documentazione del server. Un elenco delle principali *Global directives* è riportato di seguito:

`SSLMutex` configura il meccanismo usato per la mutua esclusione in tutte le operazioni che lo necessitano quando queste devono essere sincronizzate fra i vari sottoprocessi creati all'avvio di Apache; il default è `none` che significa nessun meccanismo, alternativamente si può usare la parola

[49] una trattazione dettagliata dell'argomento si trova in sez.2.1 di [SGL].

chiave `sem` che permette di usare i semafori del SysV IPC,[50] oppure la sintassi `file:/path/to/file` per indicare l'uso di un *file di lock*.[51]

`SSLRandomSeed` indica la sorgente utilizzata come seme per il sistema di generazione dei numeri pseudocasuali usati nella cifratura. La direttiva prevede tre argomenti, il primo indica quando deve essere usata la sorgente, se all'avvio (con `startup`) o prima di creare una nuova connessione (con `connect`). Il secondo argomento specifica la sorgente, secondo la sintassi illustrata in tab. 1.31, mentre il terzo, opzionale, indica il numero di byte da richiedere.

Tipo di seme	Descrizione
`builtin`	usa il meccanismo interno, è efficiente; ma usando i dati dello *scoreboard* di Apache non è una sorgente robusta se usato all'avvio quando questo ancora non è presente.
`file:/path/to/source`	usa il file specificato come sorgente, in genere si usa il file di dispositivo fornito dal sistema per la generazione di numeri casuali (nel caso si Linux `/dev/urandom`); è usualmente la scelta usata all'avvio del server.
`exec:/path/to/program`	indica l'uso di un programma esterno per generare il seme, che viene letto dallo standard output dello stesso;
`egd:/path/to/egd-socket`	utilizza come il demone EGD (*Entropy Gathering Daemon*), utilizzato quando il sistema non dispone di una fonte autonoma di numeri casuali.

Tabella 1.31: Vari tipi di semi per la generazione di numeri pseudocasuali usati con la direttiva `SSLRandomSeed`.

`SSLSessionCache` controlla l'uso della cache utilizzate per mantenere le informazioni generali sulle sessioni SSL attive. In questo modo quando un browser esegue una serie di richieste in parallelo (come è comune nella richiesta del contenuto di una pagina) e queste vengono servite da diversi processi fra quelli in ascolto, non è necessario che ciascuno di questi ripeta da zero la creazione della sessione. Il valore di default è `none` che disabilita la funzionalità; in genere lo si specifica nella forma `dbm:/path/to/datafile` indicando il file DBM[52] usato per memorizzare le informazioni.

`SSLPassPhraseDialog` Quando all'avvio del server questo legge i vari certificati e le rispettive chiavi, se queste sono protette da password è necessario disporre di un meccanismo con cui le password possano venire chieste. La direttiva controlla le modalità in cui è possibile effettuare questa operazione, e prevede il valore `builtin`, per utilizzare presentare

[50] una delle interfacce di intercomunicazione fra processi tradizionalmente presenti nei sistemi unix-like, per una trattazione dell'argomento si consulti la sezione 12.2.5 di [GaPiL].

[51] un'altra tecnica di mutua esclusione, meno efficiente, che si basa sulla presenza di un file, in genere sotto `/var/lock`; anche questa è descritta in [GaPiL], alla sezione 12.3.2.

[52] viene usata l'interfaccia DBM per la gestione di database (non relazionali) definita sotto BSD, della quale esistono varie implementazioni, la più usata delle quali è *Berkley DB*, che si può trovare su `http://www.sleepycat.com/`.

la richiesta sul terminale prima che il server se ne distacchi, o la sintassi `exec:/path/to/program` per fare eseguire la richiesta ad un programma esterno.

In generale non è necessario specificare nessuna delle precedenti direttive a livello di server, e nella gran parte delle distribuzioni la loro impostazione di default viene effettuata nel file di configurazione associato al modulo. Per l'uso di HTTPS, infatti le configurazioni vengono normalmente fatte per i singoli domini all'interno dei rispettivi *Virtual Host*, usando le altre direttive che possono essere usate anche a quel livello. Un elenco delle principali è il seguente:

`SSLEngine` attiva l'utilizzo di SSL, e prende un unico argomento i cui valori possono essere `on` o `off` e a partire dalla versione 2.1 del server anche `optional` che abilita il supporto per l'RFC 2817.

`SSLCertificateFile` indica il file che contiene il certificato che identifica il server (in formato PEM). La direttiva può essere usata due volte per poter utilizzare parallelamente il certificato sia in formato DSA che RSA. Non è ovviamente possibile associare due certificati diversi allo stesso *Virtual Host*.

`SSLCertificateKeyFile` qualora il file di certificato specificato con `SSLCertificateFile` non contenga direttamente la chiave privata ad esso associata, questa direttiva indica il file in cui questa viene mantenuta. Si usa in genere un file separato quando si ha una chiave privata non protetta da password, in modo da poter proteggere da lettura questo file e lasciare accessibile a tutti quello che contiene il certificato.

`SSLCertificateChainFile` indica il file contiene un eventuale certificato intermedio e la relativa catena di certificati fino a quello radice (da cui il nome) usato da una *Certification Authority* per firmare il certificato del server, e deve essere specificato per consentire la verifica dello stesso da parte del client.

`SSLCACertificateFile` contiene il file con i certificati delle *Certification Authority* considerate valide per il server. Queste vengono usate in caso si voglia autenticare i client.

`SSLVerifyClient` Si richiede che il client presenti un certificato valido per consentirgli di stabilire una connessione SSL, in questo modo diventa possibile identificare i client. Per quanto sia una modalità molto sicura ed efficace per consentire gli accessi è poco diffusa dovendo affrontare la gestione dei certificati sui client.

`SSLRequireSSL` richiede l'accesso alla risorsa specificata dalla direttiva container in cui si trova questa direttiva sia eseguito con una connessione sotto SSL, generando altrimenti un errore di accesso. E' utile quando si vuole impedire l'accesso in chiaro ad una qualche risorsa.

Allora un esempio tipico di utilizzo delle direttive di `mod_ssl` per fornire ad un sito l'accesso con HTTPS è il seguente, in cui si fa uso della direttiva `VirtualHost` per definire il sito ed all'interno di questo si utilizzano le direttive precedenti per attivare SSL e identificare il certificato e la rispettiva chiave:

default-ssl

```
<VirtualHost *:443>
  ServerName www.truelite.it
  DocumentRoot /var/www/truelite.it/
  <IfModule mod_ssl.c>
    SSLEngine on
    SSLCertificateFile /etc/apache/ssl.crt/server.crt
    SSLCertificateKeyFile /etc/apache/ssl.key/server.key
    SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown
  </IfModule>
</VirtualHost>
```

Si noti anche come si sia usata la direttiva `SetEnvIf` per riconoscere le richieste effettuate da Internet Explorer, caratterizzate dalla presenza della stringa `MSIE` nell'intestazione `User-Agent`. Questo è dovuto al fatto che Explorer presenta una serie di bug che creano problemi con il funzionamento standard di SSL, per questo motivo si definiscono le due variabili `nokeepalive` e `ssl-unclean-shutdown` che permettono di evitarli.

Si tenga presente che per il funzionamento normale dei *Virtual Host* (quello *name-based* attivato da `NameVirtualHost`) si può usare un solo certificato, questo perché il controllo della corrispondenza del nome a dominio del sito con quello presente nel certificato viene effettuato in fase di creazione della connessione SSL fra browser e server. Il meccanismo di `NameVirtualHost` prevede la selezione del *Virtual Host* da usare una volta che il client abbia specificato quale è il sito a cui si rivolge con l'intestazione `Host` del protocollo (si ricordi quanto detto in sez. 1.3.3) cosa che avviene *dopo* che la connessione è stabilita.

Questo pone un problema non banale in quando in genere i certificati vengono rilasciati per un sigolo sito (a parte quelli *wildcard*, che comunque valgono per un singolo dominio), per cui se si hanno due *Virtual Host* relativi a domini diversi, questi non potranno stare sulla stessa connessione, vale a dire che o si dovrà usare un IP diverso (o una porta diversa) per ciascuno di essi e passare cioè ad un sistema *IP-based*.

Indice analitico

Bibliografia

[AGL] Simone Piccardi. *Amministrare GNU/Linux*. http://labs.truelite.it, 2010.

[GaPiL] Simone Piccardi. *Guida alla Programmazione in Linux*. http://gapil.gnulinux.it, 2012.

[SGL] Simone Piccardi. *La sicurezza con GNU/Linux*. http://labs.truelite.it, 2003.