EBook Gratuito

APPRENDIMENTO HTTP

Free unaffiliated eBook created from
Stack Overflow contributors.

#http

Sommario

Di

Capitolo 1: Iniziare con HTTP

Osservazioni

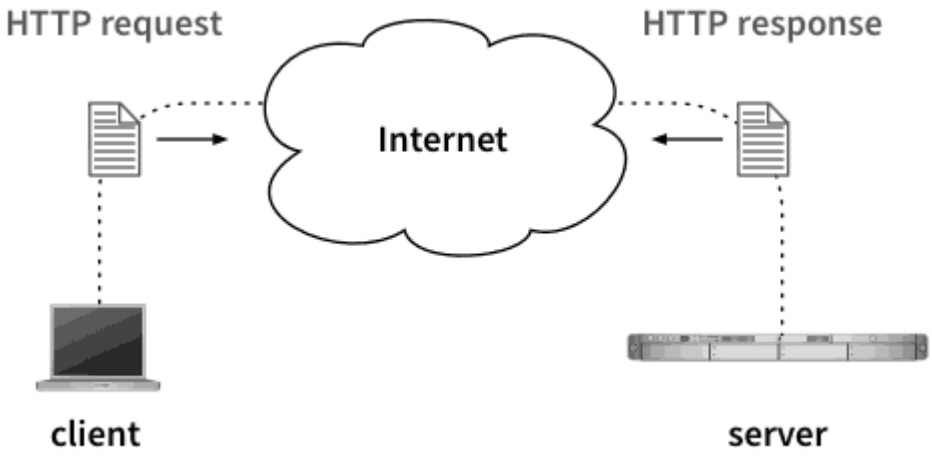
Hypertext Transfer Protocol (HTTP) utilizza un modello client-richiesta / server-risposta. HTTP è un protocollo stateless, il che significa che non richiede al server di conservare informazioni o stato su ciascun utente per la durata di più richieste. Tuttavia, per motivi di prestazioni e per evitare problemi di latenza di connessione del TCP, possono essere utilizzate tecniche come le connessioni Persistente, Parallela o Pipelined.

Versioni

Versione	Gli appunti)	Data di rilascio stimata
HTTP / 0.9	"Come implementato"	1991-01-01
HTTP / 1.0	Prima versione di HTTP / 1.0, ultima versione che non è stata trasformata in una RFC	1992/01/01
HTTP / 1.0 r1	Prima RFC ufficiale per HTTP	1996/05/01
HTTP / 1.1	Miglioramenti nella gestione delle connessioni, supporto per gli host virtuali basati sui nomi	1997-01-01
HTTP / 1.1 r1	Eliminazione dell'uso di parole chiave disambigui, possibili problemi con l'impostazione dei messaggi	1999/06/01
HTTP / 1.1 r2	Revisione importante	2014/06/01
HTTP / 2	Prima specifica per HTTP / 2	2015/05/01

Examples

Richieste e risposte HTTP



HTTP descrive come un client HTTP, come un browser Web, invia una richiesta HTTP tramite una rete a un server HTTP, che quindi invia una risposta HTTP al client.

La richiesta HTTP è in genere una richiesta di una risorsa online, ad esempio una pagina Web o un'immagine, ma può anche includere informazioni aggiuntive, come i dati inseriti in un modulo. La risposta HTTP è in genere una rappresentazione di una risorsa online, ad esempio una pagina Web o un'immagine.

HTTP / 1.0

HTTP / 1.0 è stato descritto in RFC 1945 .

HTTP / 1.0 non ha alcune funzionalità che sono oggi di fatto richieste sul Web, come l'intestazione `Host` per gli host virtuali.

Tuttavia, i client e i server HTTP a volte dichiarano di utilizzare HTTP / 1.0 se hanno un'implementazione incompleta del protocollo HTTP / 1.1 (ad esempio senza codifica di trasferimento chunk o pipelining), o la compatibilità è considerata più importante delle prestazioni (ad esempio quando si connette al proxy locale server).

```
GET / HTTP/1.0
User-Agent: example/1

HTTP/1.0 200 OK
Content-Type: text/plain

Hello
```

HTTP / 1.1

HTTP / 1.1 è stato originariamente specificato nel 1999 in RFC 2616 (protocollo) e RFC 2617 (autenticazione), ma questi documenti sono ormai obsoleti e non dovrebbero essere utilizzati come riferimento:

> Non utilizzare RFC2616. Eliminalo dai tuoi dischi rigidi, dai segnalibri e masterizza (o ricicla in modo responsabile) qualsiasi copia stampata.

- Mark Nottingham, presidente del WG HTTP

Le specifiche aggiornate di HTTP / 1.1, che corrispondono al modo in cui oggi viene implementato HTTP, si trovano nelle nuove RFC 723x:

- RFC 7230: Sintassi dei messaggi e routing
- RFC 7231: semantica e contenuto
- RFC 7232: richieste condizionali
- RFC 7233: richieste di intervallo
- RFC 7234: memorizzazione nella cache
- RFC 7235: autenticazione

HTTP / 1.1 aggiunto, tra le altre caratteristiche:

- codifica di trasferimento chunked, che consente ai server di inviare risposte affidabili di dimensioni sconosciute,
- connessioni TCP / IP persistenti (che erano estensioni non standard in HTTP / 1.0),
- richieste di intervallo utilizzate per riprendere i download,
- controllo della cache.

HTTP / 1.1 ha provato a introdurre il pipelining, che ha consentito ai client HTTP di ridurre la latenza richiesta-risposta inviando più richieste contemporaneamente senza attendere le risposte. Sfortunatamente, questa caratteristica non è mai stata correttamente implementata in alcuni proxy, causando la perdita o l'ordine delle connessioni in pipeline.

```
GET / HTTP/1.0
User-Agent: example/1
Host: example.com

HTTP/1.0 200 OK
Content-Type: text/plain
Content-Length: 6
Connection: close

Hello
```

HTTP / 2

HTTP / 2 (RFC 7540) ha cambiato il formato on-the-wire di HTTP da una semplice intestazione di richiesta e risposta basata su testo al formato di dati binari inviato in frame. HTTP / 2 supporta la compressione delle intestazioni (HPACK).

Questo ha ridotto il sovraccarico delle richieste e ha consentito la ricezione di più risposte contemporaneamente su una singola connessione TCP / IP.

Nonostante i grandi cambiamenti nel formato dei dati, HTTP / 2 utilizza ancora le intestazioni HTTP e le richieste e le risposte possono essere accuratamente tradotte tra HTTP / 1.1 e 2.

HTTP / 0.9

La prima versione di HTTP che è stata creata è 0.9, spesso denominata " HTTP come implementata ". Una descrizione comune di 0.9 è "una sottosezione del protocollo HTTP completo [cioè 1.0]." Tuttavia, questo non riesce a illustrare la disparità di capacità tra 0.9 e 1.0.

Né richieste né risposte in 0.9 feature headers. Le richieste consistono in una singola riga `GET` terminata da CRLF, seguita da uno spazio, seguito dall'URL della risorsa richiesta. Le risposte dovrebbero essere un singolo documento HTML. La fine di detto documento è contrassegnata dalla caduta della connessione lato server. Non ci sono strutture per indicare il successo o il fallimento di un'operazione. L'unica proprietà interattiva è la stringa di ricerca che è strettamente legata al tag HTML `<isindex>` .

L'utilizzo di HTTP / 0.9 è oggi eccezionalmente raro. Occasionalmente viene visto su sistemi embedded come alternativa a tftp .

Leggi Iniziare con HTTP online: https://riptutorial.com/it/http/topic/984/iniziare-con-http

Capitolo 2: Autenticazione

Parametri

Parametro	Dettagli
Stato della risposta	`401` se il server di origine richiede l'autenticazione, `407` se un proxy intermedio richiede l'autenticazione
Intestazioni di risposta	`WWW-Authenticate` dal server di origine, `Proxy-Authenticate` da un proxy intermedio
Richiedi intestazioni	`Authorization` per autorizzazione contro un server di origine, `Proxy-Authorization` contro un proxy intermedio
Schema di autenticazione	`Basic` per l'autenticazione di base, ma altri come `Digest` e `SPNEGO` possono essere utilizzati. Vedere il Registro di sistema di autenticazione HTTP .
Regno	Un nome dello spazio protetto sul server; un server può avere più spazi di questo tipo, ciascuno con un nome e meccanismi di autenticazione distinti.
Credenziali	Per `Basic` : nome utente e password separati da due punti, codificati base64; ad esempio, `username:password` base64-encoded è `dXNlcm5hbWU6cGFzc3dvcmQ=`

Osservazioni

L'autenticazione di base è definita nella RFC2617 . Può essere utilizzato per autenticare contro il server di origine dopo aver ricevuto un `401 Unauthorized` e un server proxy dopo un `407 (Proxy Authentication Required)` . Nelle credenziali (decodificate), la password inizia dopo i primi due punti. Pertanto, il nome utente non può contenere due punti, ma la password può.

Examples

Autenticazione di base HTTP

L'autenticazione di base HTTP fornisce un meccanismo diretto per l'autenticazione. Le credenziali sono inviate in testo normale, quindi non sono sicure per impostazione predefinita.
L'autenticazione riuscita procede come segue.

Il client richiede una pagina per cui l'accesso è limitato:

```
GET /secret
```

Il server risponde con il codice di stato `401 Unauthorized` e richiede al client di autenticarsi:

```
401 Unauthorized
WWW-Authenticate: Basic realm="Secret Page"
```

Il client invia l'intestazione `Authorization` . Le credenziali sono `username:password` base64 codificata:

```
GET /secret
Authorization: Basic dXNlcm5hbWU6cGFzc3dvcmQ=
```

Il server accetta le credenziali e risponde al contenuto della pagina:

```
HTTP/1.1 200 OK
```

Leggi Autenticazione online: https://riptutorial.com/it/http/topic/3286/autenticazione

Capitolo 3: Codici di stato HTTP

introduzione

In HTTP, i codici di stato sono un meccanismo leggibile dalla macchina che indica il risultato di una richiesta precedentemente emessa. Da RFC 7231, sec. 6 : "L'elemento codice di stato è un codice intero a tre cifre che fornisce il risultato del tentativo di comprendere e soddisfare la richiesta."

La grammatica formale consente ai codici di essere compresi tra `000` e `999` . Tuttavia, solo il range da `100` a `599` ha assegnato un significato.

Osservazioni

HTTP / 1.1 definisce un numero di codici di stato HTTP numerici che appaiono nella riga di stato - la prima riga di una risposta HTTP - per riassumere ciò che il client dovrebbe fare con la risposta.

La prima cifra di un codice di stato definisce la classe della risposta:

- `1xx` informativo
- Richiesta client `2xx` riuscita
- `3xx` Richiesta reindirizzata : ulteriori azioni necessarie, ad esempio una nuova richiesta
- Errore client `4xx` - non ripetere la stessa richiesta
- Errore del server `5xx` - forse riprovare

In pratica, non è sempre facile scegliere il codice di stato più appropriato.

Examples

500 Errore interno del server

Un **errore di server interno HTTP 500** è un messaggio generico che indica che il server ha riscontrato qualcosa di inaspettato. Le applicazioni (o il server Web sovrastante) dovrebbero utilizzare un 500 quando si verifica un errore nell'elaborazione della richiesta, ovvero viene generata un'eccezione o una condizione della risorsa impedisce il completamento del processo.

Esempio di riga di stato:

```
HTTP/1.1 500 Internal Server Error
```

404 non trovato

HTTP 404 Not Found significa che il server non è stato in grado di trovare il percorso utilizzando l'URI richiesto dal client.

```
HTTP/1.1 404 Not Found
```

Molto spesso, il file richiesto è stato cancellato, ma a volte può essere una errata configurazione della radice del documento o una mancanza di permessi (anche se le autorizzazioni mancanti attivano più spesso HTTP 403 Proibito).

Ad esempio, Microsoft IIS scrive 404.0 (0 è lo stato secondario) nei suoi file di registro quando il file richiesto è stato cancellato. Ma quando la richiesta in arrivo viene bloccata dalle regole di filtro richieste, scrive 404.5-404.19 per registrare i file in base a quale regola blocca la richiesta. Un riferimento al codice di errore più dettagliato è disponibile sul supporto Microsoft .

Negare l'accesso ai file protetti

Usa 403 Proibito quando un client ha richiesto una risorsa inaccessibile a causa dei controlli di accesso esistenti. Ad esempio, se la tua app ha una route `/admin` che dovrebbe essere accessibile solo agli utenti con diritti amministrativi, puoi utilizzare 403 quando un utente normale richiede la pagina.

```
GET /admin HTTP/1.1
Host: example.com
```

```
HTTP/1.1 403 Forbidden
```

Richiesta di successo

Invia una risposta HTTP con il codice di stato `200` per indicare una richiesta corretta. La riga di stato della risposta HTTP è quindi:

```
HTTP/1.1 200 OK
```

Il testo di stato `OK` è solo informativo. Il corpo della risposta (payload del messaggio) dovrebbe contenere una rappresentazione della risorsa richiesta. Se non vi è alcuna rappresentazione, non si dovrebbe utilizzare alcun contenuto.

Risposta a una richiesta condizionale per il contenuto memorizzato nella cache

Invia uno stato di risposta **304 non modificato** dal server invia in risposta a una richiesta client che contiene intestazioni `If-Modified-Since` e `If-None-Match` , se la risorsa richiesta non è stata modificata.

Ad esempio, se una richiesta di un client per una pagina Web include l'intestazione `If-Modified-Since: Fri, 22 Jul 2016 14:34:40 GMT` e la pagina non è stata modificata da allora, rispondere con la riga di stato `HTTP/1.1 304 Not Modified`

Top 10 Codice di stato HTTP

2xx successo

- **200 OK** : risposta standard per richieste HTTP riuscite.
- **201 Creato** : la richiesta è stata soddisfatta, con conseguente creazione di una nuova risorsa.
- **204 Nessun contenuto** : il server ha elaborato correttamente la richiesta e non restituisce alcun contenuto.

3xx Reindirizzamento

- **304 non modificato** : indica che la risorsa non è stata modificata dalla versione specificata dalle intestazioni della richiesta `If-Modified-Since` o `If-None-Match` .

Errore client 4xx

- **400 Richiesta** non valida - Il server non può o non vuole elaborare la richiesta a causa di un errore apparente del client (ad es. Sintassi di richiesta non valida, dimensioni troppo grandi, frame di messaggi di richieste non valide o instradamento di richieste ingannevoli).
- **401 Non autorizzato** - *Simile a 403 Proibito* , ma specificamente per l'uso quando è richiesta l'autenticazione e non è stato eseguito o non è stato ancora fornito. La risposta deve includere un campo di intestazione `WWW-Authenticate` contenente una richiesta di sfida per la risorsa richiesta.
- **403 Proibito** : la richiesta era valida, ma il server si rifiuta di rispondere. L'utente potrebbe essere registrato ma non dispone delle autorizzazioni necessarie per la risorsa.
- **404 non trovato** - La risorsa richiesta non è stata trovata ma potrebbe essere disponibile in futuro. Sono consentite richieste successive da parte del cliente.
- **409 Conflict** - Indica che la richiesta non può essere elaborata a causa di conflitti nella richiesta, come un conflitto di modifica tra più aggiornamenti simultanei.

5xx Errore del server

- **Errore interno 500 server** - Un messaggio di errore generico, fornito quando si è verificata una condizione imprevista e non è più adatto alcun messaggio specifico.

Leggi Codici di stato HTTP online: https://riptutorial.com/it/http/topic/2577/codici-di-stato-http

Capitolo 4: Codifiche di risposta e compressione

Examples

Compressione HTTP

Il corpo del messaggio HTTP può essere compresso (poiché HTTP / 1.1). O dal server comprime la richiesta e aggiunge un'intestazione `Content-Encoding` o da un proxy e aggiunge un'intestazione `Transfer-Encoding` .

Un client può inviare un'intestazione di richiesta `Accept-Encoding` per indicare quali codifiche accetta.

Le codifiche più comunemente usate sono:

- gzip - deflate algorithm (LZ77) con checksum CRC32 implementato nel programma di compressione del file "gzip" (RFC1952)
- deflate - formato dati "zlib" (RFC1950), algoritmo di svuotamento (ibrido LZ77 e Huffman) con checksum Adler32

Più metodi di compressione

È possibile comprimere un corpo del messaggio di risposta HTTP più di una volta. I nomi della codifica dovrebbero quindi essere separati da una virgola nell'ordine in cui sono stati applicati. Ad esempio, se un messaggio è stato compresso tramite deflate e quindi gzip, l'intestazione dovrebbe essere simile a:

```
Content-Encoding: deflate, gzip
```

Più intestazioni di `Content-Encoding` sono valide anche se non raccomandate:

```
Content-Encoding: deflate
Content-Encoding: gzip
```

compressione gzip

Il client invia prima una richiesta con un'intestazione `Accept-Encoding` che indica che supporta gzip:

```
GET / HTTP/1.1\r\n
Host: www.google.com\r\n
Accept-Encoding: gzip, deflate\r\n
\r\n
```

Il server può quindi inviare una risposta con un corpo di risposta compresso e un'intestazione

`Content-Encoding` che specifica che è stata utilizzata la codifica gzip ::

```
HTTP/1.1 200 OK\r\n
Content-Encoding: gzip\r\n
Content-Length: XX\r\n
\r\n
... compressed content ...
```

Leggi Codifiche di risposta e compressione online:
https://riptutorial.com/it/http/topic/5046/codifiche-di-risposta-e-compressione

Capitolo 5: HTTP per API

Osservazioni

Le API HTTP utilizzano un ampio spettro di verbi HTTP e in genere restituiscono risposte JSON o XML.

Examples

Crea una risorsa

Non tutti sono d'accordo su quale sia il metodo più semanticamente corretto per la creazione di risorse. Pertanto, la tua API potrebbe accettare richieste `POST` o `PUT` o entrambe.

Il server dovrebbe rispondere con `201 Created` se la risorsa è stata creata correttamente. Scegli il codice di errore più appropriato se non lo fosse.

Ad esempio, se fornisci un'API per creare record dipendenti, la richiesta / risposta potrebbe essere simile a questa:

```
POST /employees HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "Charlie Smith",
    "age": 38,
    "job_title": "Software Developer",
    "salary": 54895.00
}
```

```
HTTP/1.1 201 Created
Location: /employees/1/charlie-smith
Content-Type: application/json

{
    "employee": {
        "name": "Charlie Smith",
        "age": 38,
        "job_title": "Software Developer",
        "salary": 54895.00
        "links": [
            {
                "uri": "/employees/1/charlie-smith",
                "rel": "self",
                "method": "GET"
            },
            {
                "uri": "/employees/1/charlie-smith",
                "rel": "delete",
                "method": "DELETE"
            },
```

```
            {
                "uri": "/employees/1/charlie-smith",
                "rel": "edit",
                "method": "PATCH"
            }
        ]
    },
    "links": [
        {
            "uri": "/employees",
            "rel": "create",
            "method": "POST"
        }
    ]
}
```

Includendo i `links` campi JSON nella risposta consentono al client di accedere alle risorse correlate alla nuova risorsa e all'applicazione nel suo complesso, senza dover prima conoscere i propri URI o metodi.

Modifica una risorsa

La modifica o l'aggiornamento di una risorsa è uno scopo comune per le API. Le modifiche possono essere ottenute inviando le richieste `POST` , `PUT` o `PATCH` alla rispettiva risorsa. Sebbene al `POST` sia permesso di aggiungere dati alla rappresentazione esistente di una risorsa, si raccomanda di usare `PUT` o `PATCH` perché trasmettono una semantica più esplicita.

Il server dovrebbe rispondere con `200 OK` se l'aggiornamento è stato eseguito, o `202 Accepted` se non è ancora stato applicato. Scegli il codice di errore più appropriato se non può essere completato.

Aggiornamenti completi

`PUT` ha la semantica di sostituire la rappresentazione corrente con il carico utile incluso nella richiesta. Se il payload non è dello stesso tipo di rappresentazione della rappresentazione corrente della risorsa da aggiornare, il server può decidere quale approccio adottare. RFC7231 definisce che il server può entrambi

- Riconfigurare la risorsa di destinazione in modo che rifletta il nuovo tipo di supporto
- Trasforma la rappresentazione PUT in un formato coerente con quello del resouce prima di salvarlo come nuovo stato della risorsa
- Rifiuta la richiesta con una risposta di `415 Unsupported Media Type` che indica che la risorsa di destinazione è limitata a uno specifico (set) di tipi di supporto.

Una risorsa di base contenente una rappresentazione HAL JSON come ...

```
{
    "name": "Charlie Smith",
    "age": 39,
    "job_title": "Software Developer",
```

```
    "_links": {
        "self": { "href": "/users/1234" },
        "employee": { "href": "http://www.acmee.com" },
        "curies": [{ "name": "ea", "href": "http://www.acmee.com/docs/rels/{rel}", templated":
true}],
        "ea:admin": [
            "href": "/admin/2",
            "title": "Admin"
        ]
    }
}
```

... potrebbe ricevere una richiesta di aggiornamento come questa

```
PUT /users/1234 HTTP/1.1
Host: http://www.acmee.com
Content-Type: "application/json; charset=utf-8"
Content-Length: 85
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)

{
    "name": "Charlie Gold-Smith",
    "age": 40,
    "job_title": "Senior Software Developer"
}
```

Il server può ora sostituire lo stato della risorsa con il corpo della richiesta specificato e anche modificare il tipo di contenuto da `application/hal+json` a `application/json` o convertire il payload JSON in una rappresentazione HAL JSON e quindi sostituire il contenuto della risorsa con quello trasformato o rifiutare la richiesta di aggiornamento a causa di un tipo di supporto inaplicabile con una risposta di `415 Unsupported Media Type` .

C'è una differenza tra la sostituzione diretta del contenuto o la prima trasformazione della rappresentazione nel modello di rappresentazione definito e quindi la sostituzione del contenuto esistente con quello trasformato. Una successiva richiesta `GET` restituirà la seguente risposta su una sostituzione diretta:

```
GET /users/1234 HTTP/1.1
Host: http://www.acmee.com
Accept-Encoding: gzip, deflate
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
ETag: "e0023aa4e"

{
    "name": "Charlie Gold-Smith",
    "age": 40,
    "job_title": "Senior Software Developer"
}
```

mentre l'approccio di trasformazione e quindi di sostituzione restituirà la seguente rappresentazione:

```
GET /users/1234 HTTP/1.1
```

```
Host: http://www.acmee.com
Accept-Encoding: gzip, deflate
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
ETag: e0023aa4e

{
    "name": "Charlie Gold-Smith",
    "age": 40,
    "job_title": "Senior Software Developer",
    "_links": {
        "self": { "href": "/users/1234" },
        "employee": { "href": "http://www.acmee.com" },
        "curies": [{ "name": "ea", "href": "http://www.acmee.com/docs/rels/{rel}", templated":
true}],
        "ea:admin": [
            "href": "/admin/2",
            "title": "Admin"
        ]
    }
}
```

Effetti collaterali

Nota che `PUT` può avere effetti collaterali anche se è definito come operazione idempotent! Questo è documentato in RFC7231 come

> Una richiesta PUT applicata alla risorsa di destinazione **può avere effetti collaterali su altre risorse** . Ad esempio, un articolo potrebbe avere un URI per identificare "la versione corrente" (una risorsa) che è separata dagli URI che identificano ogni particolare versione (diverse risorse che a un certo punto condividevano lo stesso stato della risorsa della versione corrente). Una richiesta PUT riuscita sull'URI della "versione corrente" potrebbe pertanto creare una nuova risorsa di versione oltre a modificare lo stato della risorsa di destinazione e potrebbe anche causare l'aggiunta di collegamenti tra le risorse correlate.

La produzione di voci di registro aggiuntive non è considerata come un effetto collaterale di solito poiché non è certamente uno stato di una risorsa in generale.

Aggiornamenti parziali

RFC7231 menziona questo per quanto riguarda gli aggiornamenti parziali:

> Gli aggiornamenti del contenuto parziale sono possibili indirizzando una risorsa identificata separatamente con stato che si sovrappone a una parte della risorsa più grande o utilizzando un metodo diverso che è stato definito in modo specifico per gli aggiornamenti parziali (ad esempio, il metodo PATCH definito in RFC5789).

Gli aggiornamenti parziali possono quindi essere eseguiti in due versioni:

- Avere una risorsa incorporare più risorse secondarie più piccole e aggiornare solo una

rispettiva risorsa secondaria anziché la risorsa completa tramite `PUT`
- Utilizzare `PATCH` e `PATCH` al server cosa aggiornare

Aggiornamento parziale con stato di sovrapposizione

Se una rappresentazione utente deve essere parzialmente aggiornata a causa di uno spostamento di un utente in un'altra posizione, invece di aggiornare direttamente l'utente, la risorsa correlata deve essere aggiornata direttamente, il che si riflette in un aggiornamento parziale della rappresentazione dell'utente.

Prima dello spostamento un utente aveva la seguente rappresentazione

```
GET /users/1234 HTTP/1.1
Host: http://www.acmee.com
Accept: application/hal+json; charset=utf-8
Accept-Encoding: gzip, deflate
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
ETag: "e0023aa4e"

{
    "name": "Charlie Gold-Smith",
    "age": 40,
    "job_title": "Senior Software Developer",
    "_links": {
        "self": { "href": "/users/1234" },
        "employee": { "href": "http://www.acmee.com" },
        "curies": [{ "name": "ea", "href": "http://www.acmee.com/docs/rels/{rel}", templated":
true}],
        "ea:admin": [
            "href": "/admin/2",
            "title": "Admin"
        ]
    },
    "_embedded": {
        "ea:address": {
            "street": "Terrace Drive, Central Park",
            "zip": "NY 10024"
            "city": "New York",
            "country": "United States of America",
            "_links": {
                "self": { "href": "/address/abc" },
                "google_maps": { "href": "http://maps.google.com/?ll=40.7739166,-73.970176" }
            }
        }
    }
}
```

Mentre l'utente si sta spostando in una nuova posizione, aggiorna le informazioni sulla sua posizione in questo modo:

```
PUT /address/abc HTTP/1.1
Host: http://www.acmee.com
Content-Type: "application/json; charset=utf-8"
Content-Length: 109
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

```
{
    "street": "Standford Ave",
    "zip": "CA 94306",
    "city": "Pablo Alto",
    "country": "United States of America"
}
```

Con la semantica trasformazione-prima della sostituzione per il tipo di media non corrispondente tra la risorsa indirizzo esistente e quella nella richiesta, come descritto sopra, la risorsa indirizzo è ora aggiornata e ha l'effetto che su una successiva richiesta `GET` sulla risorsa utente viene restituito il nuovo indirizzo per l'utente

```
GET /users/1234 HTTP/1.1
Host: http://www.acmee.com
Accept: application/hal+json; charset=utf-8
Accept-Encoding: gzip, deflate
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
ETag: "e0023aa4e"

{
    "name": "Charlie Gold-Smith",
    "age": 40,
    "job_title": "Senior Software Developer",
    "_links": {
        "self": { "href": "/users/1234" },
        "employee": { "href": "http://www.acmee.com" },
        "curies": [{ "name": "ea", "href": "http://www.acmee.com/docs/rels/{rel}", templated":
true}],
        "ea:admin": [
            "href": "/admin/2",
            "title": "Admin"
        ]
    },
    "_embedded": {
        "ea:address": {
            "street": "Standford Ave",
            "zip": "CA 94306",
            "city": "Pablo Alto",
            "country": "United States of America"
            "_links": {
                "self": { "href": "/address/abc" },
                "google_maps": { "href": "http://maps.google.com/?ll=37.4241311,-122.1524475"
}
            }
        }
    }
}
```

Patching dei dati parziali

`PATCH` è definito in RFC5789 e non fa direttamente parte delle specifiche HTTP di per sé. Un errore comune è che l' invio solo dei campi che dovrebbero essere parzialmente aggiornati è sufficiente all'interno di una richiesta `PATCH` . La specifica pertanto afferma

> Il metodo PATCH richiede che una serie di modifiche descritte nell'entità richiesta venga applicata alla risorsa identificata dall'URI della richiesta. Il set di modifiche è rappresentato in un formato chiamato "documento patch" identificato da un tipo di supporto.

Ciò significa che un client deve calcolare i passaggi necessari necessari per trasformare la risorsa dallo stato A allo stato B e inviare queste istruzioni al server.

Un popolare tipo di supporto basato su JSON per l'applicazione delle patch è JSON Patch .

Se l'età e il titolo di lavoro del nostro utente campione cambiano e deve essere aggiunto un campo aggiuntivo che rappresenta il reddito dell'utente, un aggiornamento parziale usando `PATCH` usando la Patch JSON può assomigliare a questo:

```
PATCH /users/1234 HTTP/1.1
Host: http://www.acmee.com
Content-Type: application/json-patch+json; charset=utf-8
Content-Length: 188
Accept: application/json
If-Match: "e0023aa4e"

[
    { "op": "replace", "path": "/age", "value": 40 },
    { "op": "replace", "path": "/job_title", "value": "Senior Software Developer" },
    { "op": "add", "path": "/salery", "value": 63985.00 }
]
```

`PATCH` può aggiornare più risorse contemporaneamente e richiede di applicare le modifiche atomicamente, il che significa che devono essere applicate tutte le modifiche o nessuna che apporta un carico di transazione sull'implementatore dell'API.

Un aggiornamento riuscito potrebbe restituire qualcosa di simile

```
HTTP/1.1 200 OK
Location: /users/1234
Content-Type: application/json
ETag: "df00eb258"

{
    "name": "Charlie Smith",
    "age": 40,
    "job_title": "Senior Software Developer",
    "salary": 63985.00
}
```

sebbene non sia limitato solo a `200 OK` codici di risposta `200 OK` .

Per evitare aggiornamenti intermedi (le modifiche effettuate tra il recupero precedente dello stato di rappresentazione e l'aggiornamento) devono essere utilizzate l'intestazione `ETag` , `If-Match` o `If-Unmodified-Since` .

Gestione degli errori

La specifica su `PATCH` consiglia la seguente gestione degli errori:

genere	Codice di errore
Documento patch malformato	`400 Bad Request`
Documento di patch non supportato	`415 Unsupported Media Type`
Richiesta non processabile, vale a dire se il resoure diventasse non valido applicando la patch	`422 Unprocessable Entity`
Risorsa non trovata	`404 Not Found`
Stato in conflitto, ovvero un rinominare (spostare) un campo che non esiste	`409 Conflict`
Modifica in conflitto, cioè se il client utilizza un'intestazione `If-Match` o `If-Unmodified-Since` che ha avuto esito negativo. Se non era disponibile alcuna precondizione, è necessario restituire il secondo codice di errore	`412 Precondition Failed` o `409 Conflict`
Modifica simultanea, cioè se la richiesta deve essere applicata prima dell'accettazione, ulteriori richieste di `PATCH`	`409 Conflict`

Elimina una risorsa

Un altro uso comune delle API HTTP è quello di eliminare una risorsa esistente. Questo di solito dovrebbe essere fatto usando le richieste `DELETE` .

Se la cancellazione ha avuto successo, il server dovrebbe restituire `200 OK` ; un appropriato codice di errore se non lo fosse.

Se il nostro dipendente Charlie Smith ha lasciato l'azienda e ora vogliamo cancellare i suoi record, potrebbe essere simile a questo:

```
DELETE /employees/1/charlie-smith HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    'links': [
        {
            'uri': '/employees',
            'rel': 'create',
            'method': 'POST'
        }
    ]
}
```

Elenca le risorse

L'ultimo uso comune delle API HTTP è ottenere un elenco di risorse esistenti sul server. Gli elenchi come questo dovrebbero essere ottenuti utilizzando le richieste `GET` , poiché *recuperano* solo i dati.

Il server deve restituire `200 OK` se è in grado di fornire l'elenco o un codice di errore appropriato, in caso contrario.

Elencare i nostri dipendenti, quindi, potrebbe assomigliare a questo:

```
GET /employees HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    'employees': [
        {
            'name': 'Charlie Smith',
            'age': 39,
            'job_title': 'Software Developer',
            'salary': 63985.00
            'links': [
                {
                    'uri': '/employees/1/charlie-smith',
                    'rel': 'self',
                    'method': 'GET'
                },
                {
                    'uri': '/employees/1/charlie-smith',
                    'rel': 'delete',
                    'method': 'DELETE'
                },
                {
                    'uri': '/employees/1/charlie-smith',
                    'rel': 'edit',
                    'method': 'PATCH'
                }
            ]
        },
        {
            'name': 'Donna Prima',
            'age': 30,
            'job_title': 'QA Tester',
            'salary': 77095.00
            'links': [
                {
                    'uri': '/employees/2/donna-prima',
                    'rel': 'self',
                    'method': 'GET'
                },
                {
                    'uri': '/employees/2/donna-prima',
                    'rel': 'delete',
                    'method': 'DELETE'
```

```
                },
                {
                    'uri': '/employees/2/donna-prima',
                    'rel': 'edit',
                    'method': 'PATCH'
                }
            ]
        }
    ],
    'links': [
        {
            'uri': '/employees/new',
            'rel': 'create',
            'method': 'PUT'
        }
    ]
}
```

Leggi HTTP per API online: https://riptutorial.com/it/http/topic/3423/http-per-api

Capitolo 6: Memorizzazione nella cache delle risposte HTTP

Osservazioni

Le risposte vengono memorizzate nella cache separatamente per ogni URL e ogni metodo HTTP.

Il caching HTTP è definito in RFC 7234 .

Glossario

- **fresco** - stato di una risposta memorizzata nella cache, che non è ancora scaduta. In genere, una nuova risposta può *soddisfare le* richieste senza la necessità di contattare il server da cui ha origine la risposta.
- **stantio** - stato di una risposta memorizzata nella cache, che ha superato la data di scadenza. In genere, le risposte obsolete non possono essere utilizzate per *soddisfare* una richiesta senza verificare con il server se è ancora valida.
- **soddisfare** - risposta in cache *soddisfa* una richiesta quando tutte le condizioni della domanda corrisponde la risposta in cache, ad esempio, hanno lo stesso metodo HTTP e URL, la risposta è fresco o la richiesta permette risposte stantie, intestazioni di richiesta corrispondono intestazioni elencate nella risposta di `Vary` intestazione, ecc .
- **riconvalida** - verifica se una risposta memorizzata nella cache è recente. Questo di solito è fatto con una *richiesta condizionale* contenente `If-Modified-Since` o `If-None-Match` e lo stato della risposta `304` .
- **cache privata** - cache per un singolo utente, ad esempio in un browser web. Le cache private possono memorizzare risposte personalizzate.
- **cache pubblica** - cache condivisa tra molti utenti, ad esempio in un server proxy. Tale cache può inviare la stessa risposta a più utenti.

Examples

Risposta della cache per tutti per 1 anno

```
Cache-Control: public, max-age=31536000
```

`public` significa che la risposta è la stessa per tutti gli utenti (non contiene alcuna informazione personalizzata). `max-age` è in secondi da ora. 31536000 = 60 * 60 * 24 * 365.

Questo è raccomandato per le risorse statiche che non si intende mai modificare.

Risposta personalizzata in cache per 1 minuto

```
Cache-Control: private, max-age=60
```

`private` specifica che la risposta può essere memorizzata nella cache solo per l'utente che ha richiesto la risorsa e non può essere riutilizzata quando altri utenti richiedono la stessa risorsa. Questo è appropriato per le risposte che dipendono dai cookie.

Interrompere l'uso delle risorse memorizzate nella cache senza prima verificare con il server

```
Cache-Control: no-cache
```

Il client si comporterà come se la risposta non fosse stata memorizzata nella cache. Questo è appropriato per risorse che possono cambiare in modo imprevedibile in qualsiasi momento e che gli utenti devono sempre vedere nell'ultima versione.

Le risposte con `no-cache` saranno più lente (latenza elevata) a causa della necessità di contattare il server ogni volta che vengono utilizzate.

Tuttavia, per risparmiare larghezza di banda, i client *possono* ancora memorizzare tali risposte. Le risposte con `no-cache` non verranno utilizzate per soddisfare le richieste senza contattare il server ogni volta per verificare se è possibile riutilizzare la risposta memorizzata nella cache.

Richiedi risposte da non memorizzare affatto

```
Cache-control: no-store
```

Indica ai clienti di non memorizzare nella cache la risposta in alcun modo e di dimenticarla al più presto possibile.

Questa direttiva è stata originariamente progettata per dati sensibili (oggi dovrebbe essere usato HTTPS), ma può essere utilizzata per evitare l'inquinamento delle cache con risposte che non possono essere riutilizzate.

È appropriato solo in casi specifici in cui i dati di risposta sono sempre diversi, ad esempio un endpoint API che restituisce un numero casuale elevato. In caso contrario, è possibile utilizzare `no-cache` e revalidation per avere un comportamento di risposta "non memorizzabile", pur conservando una certa larghezza di banda.

Intestazioni obsolete, ridondanti e non standard

- `Expires` : specifica la data in cui la risorsa diventa obsoleta. Si basa su server e client che hanno orologi precisi e che supportano correttamente i fusi orari. `Cache-control: max-age` ha la precedenza su `Expires` ed è generalmente più affidabile.
- `post-check` direttive `post-check` e `pre-check` sono estensioni di Internet Explorer non standard che consentono l'utilizzo di risposte obsolete. L'alternativa standard è `stale-while-revalidate` .
- `Pragma: no-cache` - obsoleto nel 1999 . `Cache-control` posto dovrebbe essere usato il `Cache-control`

.

Modifica delle risorse memorizzate nella cache

Il metodo più semplice per ignorare la cache è modificare l'URL. Questo è usato come best practice quando l'URL contiene una versione o un checksum della risorsa, ad es

```
http://example.com/image.png?version=1
http://example.com/image.png?version=2
```

Questi due URL verranno memorizzati nella cache separatamente, quindi anche se `…?version=1` stato memorizzato nella cache *per sempre* , una nuova copia potrebbe essere immediatamente recuperata come `…?version=2` .

Si prega di non utilizzare URL casuali per bypassare le cache. Usa `Cache-control: no-cache` o `Cache-control: no-store` . Se le risposte con URL casuali vengono inviate senza la direttiva `no-store` , verranno memorizzate inutilmente nella cache e verranno inviate fuori dalla cache più risposte utili, degradando le prestazioni dell'intera cache.

Leggi Memorizzazione nella cache delle risposte HTTP online:
https://riptutorial.com/it/http/topic/3296/memorizzazione-nella-cache-delle-risposte-http

Capitolo 7: Origine incrociata e controllo di accesso

Osservazioni

La condivisione delle risorse tra origini è progettata per consentire richieste dinamiche tra domini, spesso utilizzando tecniche come AJAX . Mentre lo scripting fa la maggior parte del lavoro, il server HTTP deve supportare la richiesta usando le intestazioni corrette.

Examples

Cliente: invio di una richiesta di condivisione delle risorse tra origini (CORS)

Una *richiesta di* origine incrociata deve essere inviata includendo l'intestazione `Origin` . Questo indica da dove è originata la richiesta. Ad esempio, una richiesta di origine incrociata da `http://example.com` a `http://example.org` potrebbe essere simile a questa:

```
GET /cors HTTP/1.1
Host: example.org
Origin: example.com
```

Il server utilizzerà questo valore per determinare se la richiesta è autorizzata.

Server: risposta a una richiesta CORS

La risposta a una richiesta CORS deve includere un'intestazione `Access-Control-Allow-Origin` , che impone quali origini sono autorizzate a utilizzare la risorsa CORS. Questa intestazione può assumere uno dei tre valori:

- Un'origine In questo modo solo le richieste provenienti da *tale origine* .
- Il personaggio `*` . Ciò consente richieste da *qualsiasi origine* .
- La stringa `null` . Ciò *non* consente *richieste CORS* .

Ad esempio, alla ricezione di una richiesta CORS dall'origine `http://example.com` , se `example.com` è un'origine autorizzata, il server restituirà questa risposta:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: example.com
```

Una risposta di qualsiasi origine consentirebbe anche questa richiesta, vale a dire:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
```

Consentire credenziali utente o sessione

Consentendo alle credenziali dell'utente o alla sessione dell'utente di essere inviata con una richiesta CORS, il server può conservare i dati utente attraverso le richieste CORS. Ciò è utile se il server deve controllare se l'utente è loggato prima di fornire dati (ad esempio, eseguendo un'azione solo se un utente ha effettuato l'accesso - ciò richiederebbe la richiesta CORS di essere inviata con credenziali).

Questo può essere ottenuto lato server per le richieste preflight, inviando l'intestazione `Access-Control-Allow-Credentials` in risposta alla richiesta di preflight `OPTIONS` . Prendi il seguente caso di una richiesta CORS per `DELETE` una risorsa:

```
OPTIONS /cors HTTP/1.1
Host: example.com
Origin: example.org
Access-Control-Request-Method: DELETE
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: example.org
Access-Control-Allow-Methods: DELETE
Access-Control-Allow-Credentials: true
```

`Access-Control-Allow-Credentials: true` line indica che la seguente richiesta `DELETE` CORS può essere inviata con le credenziali dell'utente.

Richieste di verifica preliminare

Una richiesta CORS di base può utilizzare uno dei due soli metodi:

- OTTENERE
- INVIARE

e solo alcune intestazioni selezionate. Le richieste POST CORS possono inoltre scegliere tra solo tre tipi di contenuto.

Per evitare questo problema, le richieste che desiderano utilizzare altri metodi, intestazioni o tipi di contenuto devono prima emettere una richiesta di *preflight* , che è una richiesta `OPTIONS` che include intestazioni di richiesta di controllo dell'accesso. Ad esempio, questa è una richiesta di preflight che controlla se il server accetta una richiesta `PUT` che include un'intestazione `DNT` :

```
OPTIONS /cors HTTP/1.1
Host: example.com
Origin: example.org
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: DNT
```

Server: risposta alle richieste di verifica preliminare

Quando un server riceve una richiesta di verifica preliminare, deve verificare se supporta il metodo

e le intestazioni richieste e inviare una risposta che indica la sua capacità di supportare la richiesta, nonché qualsiasi altro dato consentito (come le credenziali).

Questi sono indicati nelle intestazioni Allow-control Allow. Il server può anche inviare un'intestazione `Max-Age` controllo accessi, indicando per quanto tempo è possibile memorizzare la risposta di preflight.

Questo è ciò che un ciclo richiesta-risposta per una richiesta di preflight può avere:

```
OPTIONS /cors HHTP/1.1
Host: example.com
Origin: example.org
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: DNT
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: example.org
Access-Control-Allow-Methods: PUT
Access-Control-Allow-Headers: DNT
```

Leggi Origine incrociata e controllo di accesso online:
https://riptutorial.com/it/http/topic/3424/origine-incrociata-e-controllo-di-accesso

Capitolo 8: Richieste HTTP

Parametri

Metodo HTTP	Scopo
`OPTIONS`	Recupera informazioni sulle opzioni di comunicazione (metodi disponibili e intestazioni) disponibili sull'URI della richiesta specificata.
`GET`	Recupera i dati identificati dall'URI della richiesta o i dati prodotti dallo script disponibili all'URI della richiesta.
`HEAD`	Identico a `GET` tranne per il fatto che nessun corpo del messaggio verrà restituito dal server: solo intestazioni.
`POST`	Inviare un blocco di dati (specificato nel corpo del messaggio) al server per l'aggiunta al reso specificato nell'URI della richiesta. Più comunemente usato per l'elaborazione dei moduli.
`PUT`	Memorizza le informazioni incluse (nel corpo del messaggio) come una risorsa nuova o aggiornata sotto l'URI della richiesta specificata.
`DELETE`	Elimina, o accoda per l'eliminazione, la risorsa identificata dall'URI della richiesta.
`TRACE`	Essenzialmente un comando echo: un server HTTP funzionante e conforme deve inviare l'intera richiesta come corpo di una risposta 200 (OK).

Osservazioni

Il metodo `CONNECT` è riservato dalle specifiche delle definizioni dei metodi da utilizzare con i proxy che sono in grado di passare da modalità proxy e tunneling (ad esempio per il tunneling SSL).

Examples

Invio manuale di una richiesta HTTP minima tramite Telnet

Questo esempio dimostra che HTTP è un protocollo di comunicazioni Internet basato su testo e mostra una richiesta HTTP di base e la corrispondente risposta HTTP.

È possibile utilizzare Telnet per inviare manualmente una richiesta HTTP minima dalla riga di comando, come indicato di seguito.

1. Avvia una sessione Telnet sul server web `www.example.org` sulla porta 80:

```
telnet www.example.org 80
```

Telnet segnala di essere connesso al server:

```
Connected to www.example.org.
Escape character is '^]'.
```

2. Immettere una riga di richiesta per inviare un percorso URL di richiesta GET `/` , utilizzando HTTP 1.1

```
GET / HTTP/1.1
```

3. Immettere una riga del campo dell'intestazione HTTP per identificare la parte del nome host dell'URL richiesto, che è richiesta in HTTP 1.1

```
Host: www.example.org
```

4. Inserisci una riga vuota per completare la richiesta.

   Il server Web invia la risposta HTTP, che appare nella sessione Telnet.

La sessione completa è la seguente. La prima riga della risposta è la *riga di stato HTTP* , che include il codice di stato 200 e il testo di stato *OK* , che indicano che la richiesta è stata elaborata correttamente. Questo è seguito da un numero di campi di intestazione HTTP, una riga vuota e la risposta HTML.

```
$ telnet www.example.org 80
Trying 2606:2800:220:1:248:1893:25c8:1946...
Connected to www.example.org.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.example.org

HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html
Date: Thu, 21 Jul 2016 15:56:05 GMT
Etag: "359670651"
Expires: Thu, 28 Jul 2016 15:56:05 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (lga/1318)
Vary: Accept-Encoding
X-Cache: HIT
x-ec-custom-error: 1
Content-Length: 1270

<!doctype html>
<html>
<head>
    <title>Example Domain</title>
    <meta charset="utf-8" />
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1" />
</head>
<body>
<div>
    <h1>Example Domain</h1>
    <p>This domain is established to be used for illustrative examples in documents. You may
use this
    domain in examples without prior coordination or asking for permission.</p>
    <p><a href="http://www.iana.org/domains/example">More information...</a></p>
</div>
</body>
</html>
```

(elemento di `style` e righe vuote rimosse dalla risposta HTML, per brevità.)

Formato di richiesta di base

In HTTP 1.1, una richiesta HTTP minima è composta da una riga di richiesta e un'intestazione `Host` :

```
GET /search HTTP/1.1 \r\n
Host: google.com \r\n
\r\n
```

La prima riga ha questo formato:

```
Method Request-URI HTTP-Version CRLF
```

`Method` dovrebbe essere un metodo HTTP valido; uno di [1] [2] :

- `OPTIONS`
- `GET`
- `HEAD`
- `POST`
- `PUT`
- `DELETE`
- `PATCH`
- `TRACE`
- `CONNECT`

`Request-URI` indica l'URI o il percorso della risorsa richiesta dal client. Può essere o:

- un URI pienamente qualificato, inclusi schema, host, porta (e) opzionale e percorso; o
- un percorso, nel qual caso l'host deve essere specificato nell'intestazione `Host`

`HTTP-Version` indica la versione del protocollo HTTP che il client sta usando. Per le richieste HTTP 1.1, questo deve sempre essere `HTTP/1.1` .

La riga di richiesta termina con una coppia di feed di ritorno riga di trasporto, solitamente rappresentata da `\r\n` .

Richiedi i campi dell'intestazione

I campi intestazione (di solito chiamati semplicemente "intestazioni") possono essere aggiunti a una richiesta HTTP per fornire ulteriori informazioni con la richiesta. Un'intestazione ha una semantica simile ai parametri passati a un metodo in qualsiasi linguaggio di programmazione che supporti tali elementi.

Una richiesta con intestazioni `Host` , `User-Agent` e `Referer` potrebbe essere simile a questa:

```
GET /search HTTP/1.1 \r\n
Host: google.com \r\n
User-Agent: Chrome/54.0.2803.1 \r\n
Referer: http://google.com/ \r\n
\r\n
```

Un elenco completo delle intestazioni delle richieste HTTP 1.1 supportate può essere trovato nelle specifiche . I più comuni sono:

- `Host` : la parte del nome host dell'URL della richiesta (richiesta in HTTP / 1.1)
- `User-Agent` - una stringa che rappresenta la richiesta dell'utente agente;
- `Referer` : l'URI di cui il cliente è stato indirizzato qui; e
- `If-Modified-Since` : indica una data che il server può utilizzare per determinare se una risorsa è stata modificata e indica che il client può utilizzare una copia memorizzata nella cache, se non è stata così.

Un'intestazione dovrebbe essere formata come `Name: Value CRLF` . `Name` è il nome dell'intestazione, ad esempio `User-Agent` . `Value` è il dato assegnato ad esso e la linea dovrebbe terminare con un CRLF. I nomi delle intestazioni non fanno distinzione tra maiuscole e minuscole e possono utilizzare solo lettere, cifre e caratteri `!#$%&'*+-.^_`|~` (RFC7230 sezione 3.2.6 Componenti valore campo).

Il nome del campo dell'intestazione del `Referer` è un refuso per "referrer", introdotto accidentalmente in RFC1945 .

Corpi dei messaggi

Alcune richieste HTTP possono contenere un corpo del messaggio. Si tratta di dati aggiuntivi che il server utilizzerà per elaborare la richiesta. I corpi dei messaggi vengono spesso utilizzati nelle richieste POST o PATCH e PUT, per fornire nuovi dati che il server deve applicare a una risorsa.

Le richieste che includono un corpo del messaggio dovrebbero sempre includere la sua lunghezza in byte con intestazione `Content-Length` .

Un corpo del messaggio è incluso *dopo* tutte le intestazioni e un doppio CRLF. Un esempio di richiesta PUT con un corpo potrebbe essere simile a questo:

```
PUT /files/129742 HTTP/1.1\r\n
Host: example.com\r\n
User-Agent: Chrome/54.0.2803.1\r\n
Content-Length: 202\r\n
\r\n
This is a message body. All content in this message body should be stored under the
```

```
/files/129742 path, as specified by the PUT specification. The message body does
not have to be terminated with CRLF.
```

`HEAD` richieste `HEAD` e `TRACE` non devono includere un corpo del messaggio.

Leggi Richieste HTTP online: https://riptutorial.com/it/http/topic/2909/richieste-http

Capitolo 9: Risposte HTTP

Parametri

Codice di stato	Frase-motivo - Descrizione
100	**Continua** : il client deve inviare la seguente parte di una richiesta multiparte.
101	**Switching Protocols** - il server sta cambiando la versione o il tipo di protocollo utilizzato in questa comunicazione.
200	**OK** : il server ha ricevuto e completato la richiesta del cliente.
201	**Creato** : il server ha accettato la richiesta e creato una nuova risorsa, disponibile sotto l'URI nell'intestazione `Location` .
202	**Accettato** : il server ha ricevuto e accettato la richiesta del cliente, ma non ha ancora iniziato o completato l'elaborazione.
203	**Informazioni non autorevoli** : il server restituisce dati che potrebbero essere un sotto o superset delle informazioni disponibili sul server originale. Utilizzato principalmente dai proxy.
204	**Nessun contenuto** - usato al posto di 200 (OK) quando non c'è alcun corpo alla risposta.
205	**Resetta il contenuto** - identico a 204 (nessun contenuto), ma il client dovrebbe ricaricare la vista del documento attivo.
206	**Contenuto parziale** : utilizzato al posto di 200 (OK) quando il client ha richiesto un'intestazione `Range` .
300	**Scelte multiple** : la risorsa richiesta è disponibile su più URI e il client deve reindirizzare la richiesta a un URI specificato nell'elenco nel corpo del messaggio.
301	**Spostato in modo permanente** - la risorsa richiesta non è più disponibile in questo URI e il client deve reindirizzare questa e tutte le future richieste all'URI specificato nell'intestazione `Location` .
302	**Trovato** : la risorsa risiede temporaneamente in un URI diverso. Questa richiesta dovrebbe essere reindirizzata sulla conferma dell'utente all'URI nell'intestazione `Location` , ma le richieste future non dovrebbero essere modificate.
303	**Vedi Altro** - molto simile a 302 (Trovato), ma non richiede l'input dell'utente per reindirizzare all'URI fornito. L'URI fornito deve essere recuperato con una

Codice di stato	Frase-motivo - Descrizione
	richiesta GET.
304	**Non modificato** : il client ha inviato un'intestazione `If-Modified-Since` o simile e la risorsa non è stata modificata da quel momento; il client dovrebbe visualizzare una copia memorizzata nella cache della risorsa.
305	**Usa Proxy** : la risorsa richiesta deve essere nuovamente richiesta tramite il proxy specificato nel campo dell'intestazione `Location` .
307	**Redirect temporaneo** - identico a 302 (Trovato), ma i client HTTP 1.0 non supportano 307 risposte.
400	**Richiesta non** valida: il client ha inviato una richiesta non valida contenente errori di sintassi e deve modificare la richiesta per correggerla prima di ripeterla.
401	**Non autorizzato** : la risorsa richiesta non è disponibile senza autenticazione. Il client può ripetere la richiesta utilizzando un'intestazione `Authorization` per fornire i dettagli di autenticazione.
402	**Pagamento richiesto** : codice di stato riservato e non specificato per l'utilizzo da parte di applicazioni che richiedono abbonamenti utente per visualizzare il contenuto.
403	**Proibito** : il server comprende la richiesta, ma rifiuta di soddisfarla a causa dei controlli di accesso esistenti. La richiesta non dovrebbe essere ripetuta.
404	**Non trovato** - non esiste alcuna risorsa disponibile su questo server che corrisponda all'URI richiesto. Può essere usato al posto di 403 per evitare di esporre i dettagli del controllo accessi.
405	**Metodo non consentito** : la risorsa non supporta il metodo di richiesta (verbo HTTP); l'intestazione `Allow` elenca i metodi di richiesta accettabili.
406	**Non accettabile** : la risorsa ha caratteristiche che violano le intestazioni di accettazione inviate nella richiesta.
407	**Autenticazione proxy richiesta** - simile a 401 (non autorizzato), ma indica che il client deve prima autenticarsi con il proxy intermedio.
408	**Timeout richiesta** : il server si aspettava un'altra richiesta dal client, ma nessuna era stata fornita entro un intervallo di tempo accettabile.
409	**Conflitto** : la richiesta non può essere completata perché è in conflitto con lo stato corrente della risorsa.
410	**Andato** - simile a 404 (non trovato), ma indica una rimozione permanente. Nessun indirizzo di inoltro disponibile.

Codice di stato	Frase-motivo - Descrizione
411	**Lunghezza richiesta** : il client non ha specificato un'intestazione `Content-Length` valida e deve farlo prima che il server accetti questa richiesta.
412	**Precondizione non riuscita** : la risorsa non è disponibile con tutte le condizioni specificate dagli header condizionali inviati dal client.
413	**Richiesta Entità troppo grande** - il server non è attualmente in grado di elaborare un corpo del messaggio della lunghezza inviata dal client.
414	**URI di richiesta troppo lungo** : il server rifiuta la richiesta perché l'URI di richiesta è più lungo di quanto il server sia disposto a interpretare.
415	**Tipo di supporto non supportato** : il server non supporta il MIME o il tipo di supporto specificato dal client e non può soddisfare questa richiesta.
416	**Intervallo richiesto non soddisfacente** : il client ha richiesto un intervallo di byte, ma il server non è in grado di fornire il contenuto a tale specifica.
417	**Expectation Failed** : i vincoli specificati dal client nell'intestazione `Expect` che il server non è in grado di soddisfare.
500	**Errore interno del server** : il server ha riscontrato una condizione o un errore imprevisto che impedisce di completare questa richiesta.
501	**Non implementato** : il server non supporta la funzionalità richiesta per completare la richiesta. Solitamente utilizzato per indicare un metodo di richiesta che non è supportato su *alcuna* risorsa.
502	**Bad Gateway** : il server è un proxy e ha ricevuto una risposta non valida dal server upstream durante l'elaborazione di questa richiesta.
503	**Servizio non disponibile** : il server è sottoposto a un carico elevato o in fase di manutenzione e al momento non è in grado di soddisfare questa richiesta.
504	**Timeout gateway** : il server è un proxy e non ha ricevuto una risposta dal server upstream in modo tempestivo.
505	**Versione HTTP non supportata** : il server non supporta la versione del protocollo HTTP con cui il client ha effettuato la richiesta.

Examples

Formato di risposta di base

Quando un server HTTP riceve una richiesta HTTP ben formata, deve elaborare le informazioni

contenute nella richiesta e restituire una risposta al client. Una semplice risposta HTTP 1.1 può apparire come una delle seguenti, solitamente seguita da un numero di campi di intestazione e, eventualmente, da un corpo di risposta:

```
HTTP/1.1 200 OK \r\n
```

```
HTTP/1.1 404 Not Found \r\n
```

```
HTTP/1.1 503 Service Unavailable \r\n
```

Una semplice risposta HTTP 1.1 ha questo formato:

```
HTTP-Version Status-Code Reason-Phrase CRLF
```

Come in una richiesta, `HTTP-Version` indica la versione del protocollo HTTP in uso; per HTTP 1.1 questa deve sempre essere la stringa `HTTP/1.1` .

`Status-Code` è un `Status-Code` a tre cifre che indica lo stato della richiesta del cliente. La prima cifra di questo codice è la *classe di stato* , che colloca il codice di stato in una delle 5 categorie di risposta [1] :

- `1xx` **Informativo** : il server ha ricevuto la richiesta e l'elaborazione continua
- `2xx` **Successo** : il server ha accettato ed elaborato la richiesta
- **Reindirizzamento** `3xx` : è necessario un ulteriore intervento da parte del cliente per completare la richiesta
- **Errori del client** `4xx` : il client ha inviato una richiesta che non era valida o che non può essere soddisfatta
- **Errori del server** `5xx` : la richiesta era valida, ma al momento il server non è in grado di soddisfarla

`Reason-Phrase` è una breve descrizione del codice di stato. Ad esempio, il codice `200` ha una frase di motivazione di `OK` ; il codice `404` ha una frase di `Not Found` . Un elenco completo di frasi di motivazione è disponibile in Parametri, sotto o nelle specifiche HTTP .

La linea termina con una coppia di trasporto a riga di ritorno, solitamente rappresentata da `\r\n` .

Intestazioni aggiuntive

Come una richiesta HTTP, una risposta HTTP può includere intestazioni aggiuntive per modificare o aumentare la risposta che fornisce.

Un elenco completo delle intestazioni disponibili è definito nel §6.2 della specifica . Le intestazioni più comunemente utilizzate sono:

- `Server` , che funziona come un'intestazione di richiesta `User-Agent` per il server;
- `Location` , che viene utilizzata sulle risposte di stato 201 e 3xx per indicare un URI da reindirizzare a; e
- `ETag` , che è un identificativo univoco per questa versione della risorsa restituita per

consentire ai client di memorizzare nella cache la risposta.

Le intestazioni di risposta vengono dopo la riga di stato e, come per le intestazioni delle richieste, sono formate come tali:

```
Name: Value CRLF
```

`Name` fornisce il nome dell'intestazione, come `ETag` o `Location` , e `Value` fornisce il valore che il server sta impostando per quella intestazione. La linea termina con un CRLF.

Una risposta con le intestazioni potrebbe essere simile a questa:

```
HTTP/1.1 201 Created \r\n
Server: WEBrick/1.3.1 \r\n
Location: http://example.com/files/129742 \r\n
```

Corpi dei messaggi

Come per i corpi delle richieste , le risposte HTTP possono contenere un corpo del messaggio. Ciò fornisce dati aggiuntivi che il cliente elaborerà. In particolare, 200 risposte OK a una richiesta GET ben formata dovrebbero sempre fornire un corpo del messaggio contenente i dati richiesti. (Se non ce n'è, 204 No Content è una risposta più appropriata).

Un corpo del messaggio è incluso dopo tutte le intestazioni e un doppio CRLF. Per quanto riguarda le richieste, la sua lunghezza in byte dovrebbe essere data con l'intestazione `Content-Length` . Una risposta positiva a una richiesta GET, quindi, potrebbe essere simile a questa:

```
HTTP/1.1 200 OK\r\n
Server: WEBrick/1.3.1\r\n
Content-Length: 39\r\n
ETag: 4f7e2ed02b836f60716a7a3227e2b5bda7ee12c53be282a5459d7851c2b4fdfd\r\n
\r\n
Nobody expects the Spanish Inquisition.
```

Leggi Risposte HTTP online: https://riptutorial.com/it/http/topic/3077/risposte-http

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con HTTP	Community, DaSourcerer, Kornel, Peter Hilton
2	Autenticazione	DaSourcerer, Peter Hilton, Stefan Kögl
3	Codici di stato HTTP	ArtOfCode, DaSourcerer, Deltik, Kornel, Lex Li, mnoronha, Peter Hilton, Rptk99, Sender, Xevaquor
4	Codifiche di risposta e compressione	Jeff Bencteux, Peter Hilton
5	HTTP per API	ArtOfCode, mnoronha, Peter Hilton, Roman Vottner
6	Memorizzazione nella cache delle risposte HTTP	DaSourcerer, Kornel
7	Origine incrociata e controllo di accesso	ArtOfCode
8	Richieste HTTP	artem, ArtOfCode, Jeff Bencteux, Peter Hilton
9	Risposte HTTP	ArtOfCode, Jeff Bencteux, Peter Hilton