Kostenloses eBook

LERNEN HTTP

Free unaffiliated eBook created from
Stack Overflow contributors.

#http

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: http

It is an unofficial and free HTTP ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official HTTP.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit HTTP

Bemerkungen

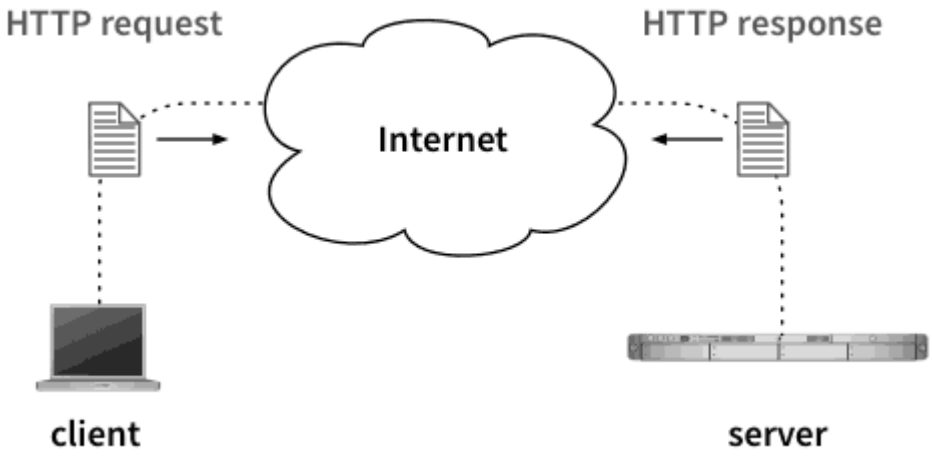
Hypertext Transfer Protocol (HTTP) verwendet ein Clientanforderungs- / Serverantwortmodell. HTTP ist ein zustandsloses Protokoll. Das bedeutet, dass der Server nicht die Informationen oder den Status jedes Benutzers für die Dauer mehrerer Anforderungen aufbewahren muss. Aus Leistungsgründen und zur Vermeidung von TCP-Verbindungslatenz können jedoch Techniken wie persistente, parallele oder Pipelined-Verbindungen verwendet werden.

Versionen

Ausführung	Anmerkungen)	Voraussichtliches Veröffentlichungsdatum
HTTP / 0,9	"Wie umgesetzt"	1991-01-01
HTTP / 1.0	Erste Version von HTTP / 1.0, letzte Version, die nicht in einen RFC aufgenommen wurde	1992-01-01
HTTP / 1.0 r1	Erster offizieller RFC für HTTP	1996-05-01
HTTP / 1.1	Verbesserungen bei der Verbindungsverarbeitung, Unterstützung für namenbasierte virtuelle Hosts	1997-01-01
HTTP / 1.1 r1	Eindeutige Verwendung von Schlüsselwörtern bereinigt, mögliche Probleme mit der Nachrichtenrahmung behoben	1999-06-01
HTTP / 1.1 r2	Generalüberholung	2014-06-01
HTTP / 2	Erste Spezifikation für HTTP / 2	2015-05-01

Examples

HTTP-Anfragen und Antworten



HTTP beschreibt, wie ein HTTP-Client, z. B. ein Webbrowser, eine HTTP-Anforderung über ein Netzwerk an einen HTTP-Server sendet, der dann eine HTTP-Antwort an den Client zurücksendet.

Die HTTP-Anforderung ist normalerweise entweder eine Anforderung für eine Online-Ressource, z. B. eine Webseite oder ein Bild, kann jedoch auch zusätzliche Informationen enthalten, z. B. Daten, die in ein Formular eingegeben werden. Die HTTP-Antwort ist normalerweise eine Darstellung einer Online-Ressource, z. B. einer Webseite oder eines Bildes.

HTTP / 1.0

HTTP / 1.0 wurde in RFC 1945 beschrieben .

HTTP / 1.0 verfügt nicht über einige Funktionen, die heute de-facto für das Web erforderlich sind, z. B. der `Host` für virtuelle Hosts.

HTTP-Clients und -Server erklären jedoch manchmal immer noch, dass sie HTTP / 1.0 verwenden, wenn das HTTP / 1.1-Protokoll unvollständig implementiert ist (z. B. ohne verschlüsselte Übertragungskodierung oder Pipelining), oder die Kompatibilität wird wichtiger als die Leistung (z. B. beim Herstellen einer Verbindung zu einem lokalen Proxy) Server).

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

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

Hello
```

HTTP / 1.1

Ursprünglich wurde HTTP / 1.1 1999 in RFC 2616 (Protokoll) und RFC 2617 (Authentifizierung) angegeben. Diese Dokumente sind jedoch veraltet und sollten nicht als Referenz verwendet werden:

> Verwenden Sie RFC2616 nicht. Löschen Sie es von Ihren Festplattenlaufwerken,

Lesezeichen und brennen Sie die ausgedruckten Kopien.

- Mark Nottingham, Vorsitzender der HTTP WG

Die aktuelle Spezifikation von HTTP / 1.1, die der heutigen Implementierung von HTTP entspricht, ist in den neuen RFCs 723x enthalten:

- RFC 7230: Nachrichtensyntax und Routing
- RFC 7231: Semantik und Inhalt
- RFC 7232: Bedingte Anforderungen
- RFC 7233: Bereichsanfragen
- RFC 7234: Zwischenspeicherung
- RFC 7235: Authentifizierung

HTTP / 1.1 wurde unter anderem hinzugefügt:

- Chunked Transfer Encoding, die es Servern ermöglicht, Antworten mit unbekannter Größe zuverlässig zu senden.
- persistente TCP / IP-Verbindungen (die keine Standarderweiterung in HTTP / 1.0 waren),
- Bereichsanfragen für die Wiederaufnahme von Downloads,
- Cache-Steuerung.

HTTP / 1.1 versuchte, ein Pipelining einzuführen, mit dem HTTP-Clients die Anforderungs-Antwort-Latenz reduzieren können, indem sie mehrere Anforderungen auf einmal senden, ohne auf Antworten zu warten. Leider wurde diese Funktion in einigen Proxys nie richtig implementiert, was dazu führte, dass Pipeline-Verbindungen gelöscht oder die Antworten neu angeordnet wurden.

```
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) hat das On-the-Wire-Format von HTTP von einfachen textbasierten Anforderungs- und Antwort-Headern in ein binäres Datenformat geändert, das in Frames gesendet wird. HTTP / 2 unterstützt die Komprimierung der Header (HPACK).

Dies reduziert den Overhead von Anforderungen und ermöglicht das gleichzeitige Empfangen mehrerer Antworten über eine einzige TCP / IP-Verbindung.

Trotz großer Änderungen im Datenformat verwendet HTTP / 2 weiterhin HTTP-Header, und Anforderungen und Antworten können genau zwischen HTTP / 1.1 und 2 übersetzt werden.

HTTP / 0,9

Die erste Version von HTTP, die zustande gekommen ist, ist 0.9, oft als " HTTP As Implemented " bezeichnet. Eine gebräuchliche Beschreibung von 0.9 ist "ein Unterabschnitt des vollständigen HTTP-Protokolls [dh 1.0]". Dies zeigt jedoch kaum, dass die unterschiedlichen Kapazitäten zwischen 0,9 und 1,0 liegen.

Weder Anforderungen noch Antworten in 0.9-Feature-Headern. Anforderungen bestehen aus einer einzelnen CRLF-terminierten Zeile von `GET` , gefolgt von einem Leerzeichen, gefolgt von der angeforderten Ressourcen-URL. Es wird erwartet, dass es sich bei den Antworten um ein einzelnes HTML-Dokument handelt. Das Ende dieses Dokuments wird markiert, indem die Verbindung serverseitig getrennt wird. Es gibt keine Möglichkeiten, den Erfolg oder Misserfolg einer Operation anzuzeigen. Die einzige interaktive Eigenschaft ist die Such - `<isindex>` Zeichenfolge , die an den eng verbunden ist `<isindex>` HTML - Tag.

Die Verwendung von HTTP / 0.9 ist heutzutage außergewöhnlich selten. Es wird gelegentlich auf eingebetteten Systemen als Alternative zu TFTP gesehen .

Erste Schritte mit HTTP online lesen: https://riptutorial.com/de/http/topic/984/erste-schritte-mit-http

Kapitel 2: Antwortkodierungen und Komprimierung

Examples

HTTP-Komprimierung

Der HTTP-Nachrichtentext kann komprimiert werden (seit HTTP / 1.1). Entweder komprimiert der Server die Anforderung und fügt einen `Content-Encoding` Header hinzu oder durch einen Proxy-Assistenten und fügt einen `Transfer-Encoding` Header hinzu.

Ein Client kann einen `Accept-Encoding` Anforderungsheader senden, um anzugeben, welche Codierungen er akzeptiert.

Die am häufigsten verwendeten Kodierungen sind:

- gzip - Deflate-Algorithmus (LZ77) mit CRC32-Prüfsumme, implementiert im Komprimierungsprogramm "gzip" (RFC1952)
- deflate - Datenformat "Zlib" (RFC1950), Deflate-Algorithmus (Hybrid LZ77 und Huffman) mit Adler32-Prüfsumme

Mehrere Komprimierungsmethoden

Es ist möglich, den Inhalt einer HTTP-Antwortnachricht mehr als einmal zu komprimieren. Die Kodierungsnamen sollten dann in der Reihenfolge, in der sie angewendet wurden, durch ein Komma getrennt werden. Wenn zum Beispiel eine Nachricht über deflate und dann gzip komprimiert wurde, sollte der Header folgendermaßen aussehen:

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

Mehrere `Content-Encoding` Header sind ebenfalls gültig, werden jedoch nicht empfohlen:

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

gzip-Komprimierung

Der Client sendet zuerst eine Anforderung mit einem `Accept-Encoding` Header, der angibt, dass er gzip unterstützt:

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

Der Server kann dann eine Antwort mit einem komprimierten Antworttext und einem `Content-Encoding` Header senden, der angibt, dass die gzip-Codierung verwendet wurde

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

Antwortkodierungen und Komprimierung online lesen:
https://riptutorial.com/de/http/topic/5046/antwortkodierungen-und-komprimierung

Kapitel 3: Authentifizierung

Parameter

Parameter	Einzelheiten
Antwortstatus	`401` wenn der Ursprungsserver eine Authentifizierung erfordert, `407` wenn ein Zwischen-Proxy eine Authentifizierung erfordert
Antwortheader	`WWW-Authenticate` durch den Ursprungsserver, `Proxy-Authenticate` durch einen Zwischenproxy
Kopfzeilen anfordern	`Authorization` zur Autorisierung gegen einen Ursprungsserver, `Proxy-Authorization` gegen einen Zwischenproxy
Authentifizierungsschema	`Basic` für die Basisauthentifizierung, aber auch andere wie `Digest` und `SPNEGO` können verwendet werden. Siehe die HTTP-Authentifizierungsschema-Registrierung .
Reich	Ein Name des geschützten Bereichs auf dem Server. Ein Server kann mehrere solcher Bereiche mit jeweils unterschiedlichen Namen und Authentifizierungsmechanismen haben.
Referenzen	Für `Basic` : Benutzername und Passwort, getrennt durch einen Doppelpunkt, base64-codiert; Beispiel: `username:password` base64-codiert ist `dXNlcm5hbWU6cGFzc3dvcmQ=`

Bemerkungen

Die Basisauthentifizierung ist in RFC2617 definiert. Es kann zur Authentifizierung beim Ursprungsserver verwendet werden, nachdem ein `401 Unauthorized` empfangen wurde, sowie bei einem Proxyserver nach einem `407 (Proxy Authentication Required)` . In den (dekodierten) Anmeldeinformationen beginnt das Kennwort nach dem ersten Doppelpunkt. Daher darf der Benutzername keinen Doppelpunkt enthalten, das Passwort jedoch.

Examples

HTTP-Basisauthentifizierung

Die HTTP-Basisauthentifizierung bietet einen einfachen Mechanismus für die Authentifizierung. Anmeldeinformationen werden im Klartext gesendet und sind daher standardmäßig unsicher. Die erfolgreiche Authentifizierung läuft wie folgt ab.

Der Client fordert eine Seite an, für die der Zugriff eingeschränkt ist:

```
GET /secret
```

Der Server antwortet mit dem Statuscode `401 Unauthorized` und fordert den Client zur Authentifizierung auf:

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

Der Client sendet den `Authorization` . Die Anmeldeinformationen lauten `username:password` base64-codiert:

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

Der Server akzeptiert die Anmeldeinformationen und antwortet mit dem Seiteninhalt:

```
HTTP/1.1 200 OK
```

Authentifizierung online lesen: https://riptutorial.com/de/http/topic/3286/authentifizierung

Kapitel 4: Herkunfts- und Zugriffskontrolle

Bemerkungen

Die urheberrechtliche Ressourcenfreigabe ermöglicht dynamische Anforderungen zwischen Domänen, wobei häufig Techniken wie AJAX verwendet werden . Während das Scripting die meiste Arbeit erledigt, muss der HTTP-Server die Anfrage mit den richtigen Kopfzeilen unterstützen.

Examples

Client: Senden einer CORS-Anforderung (Cross-Origin Resource Sharing)

Eine *Querursprungsanforderung* muss einschließlich den gesendet wird `Origin` - Header. Dies gibt an, woher die Anfrage stammt. Eine Ursprungsübergreifende Anfrage von `http://example.com` an `http://example.org` würde beispielsweise so aussehen:

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

Der Server verwendet diesen Wert, um zu bestimmen, ob die Anforderung autorisiert ist.

Server: Antworten auf eine CORS-Anfrage

Die Antwort auf eine CORS-Anforderung muss einen `Access-Control-Allow-Origin` Header enthalten, der vorschreibt, welche Ursprünge die CORS-Ressource verwenden dürfen. Dieser Header kann einen von drei Werten annehmen:

- Ein Ursprung Dies erlaubt nur Anfragen von *diesem Ursprung* .
- Das Zeichen `*` . Dies erlaubt Anfragen von *jeder Herkunft* .
- Die Zeichenfolge `null` . Dies erlaubt *keine CORS-Anfragen* .

Wenn zum Beispiel eine CORS-Anfrage vom Ursprung `http://example.com` empfangen wird, würde der Server diese Antwort `http://example.com` , wenn `example.com` ein autorisierter Ursprung ist:

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

Eine Any-Origin-Antwort würde auch diese Anfrage zulassen, dh:

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

Zulassen von Benutzeranmeldeinformationen oder Sitzungen

Wenn Benutzeranmeldeinformationen oder die Benutzersitzung mit einer CORS-Anforderung gesendet werden können, kann der Server Benutzerdaten in allen CORS-Anforderungen beibehalten. Dies ist nützlich, wenn der Server prüfen muss, ob der Benutzer angemeldet ist, bevor er Daten bereitstellt (z. B. nur eine Aktion ausführen, wenn ein Benutzer angemeldet ist - dazu müsste die CORS-Anforderung mit Anmeldeinformationen gesendet werden).

Dies kann serverseitig für Preflight-Anforderungen erreicht werden, indem der Header `Access-Control-Allow-Credentials` als Antwort auf die Preflight-Anfrage `OPTIONS` . Nehmen Sie den folgenden Fall einer CORS-Anforderung an, um eine Ressource zu `DELETE` :

```
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
```

Die Zeile `Access-Control-Allow-Credentials: true` gibt an, dass die folgende `DELETE` CORS-Anforderung mit Benutzeranmeldeinformationen gesendet werden kann.

Preflight-Anfragen

Eine grundlegende CORS-Anfrage kann eine von nur zwei Methoden verwenden:

- ERHALTEN
- POST

und nur wenige Kopfzeilen auswählen. POST CORS-Anforderungen können zusätzlich nur drei Inhaltstypen auswählen.

Um dieses Problem zu vermeiden, müssen Anforderungen, die andere Methoden, Header oder Inhaltstypen verwenden möchten, zuerst eine *Preflight*- Anforderung ausgeben, bei der es sich um eine `OPTIONS` Anforderung handelt, die Header für die Zugriffssteuerung enthält. Dies ist beispielsweise eine Preflight-Anforderung, die prüft, ob der Server eine `PUT` Anforderung akzeptiert, die einen `DNT` Header enthält:

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

Server: Antworten auf Preflight-Anfragen

Wenn ein Server eine Preflight-Anforderung empfängt, muss er prüfen, ob er die angeforderte Methode und die Header unterstützt, und eine Antwort zurücksenden, aus der hervorgeht, dass er die Anforderung unterstützen kann, sowie alle anderen zulässigen Daten (beispielsweise

Anmeldeinformationen).

Diese werden in Access-Allow-Headern angezeigt. Der Server sendet möglicherweise auch einen `Max-Age` Header für die Zugriffskontrolle `Max-Age` , der angibt, wie lange die Preflight-Antwort zwischengespeichert werden kann.

So könnte ein Request-Response-Zyklus für eine Preflight-Anfrage aussehen:

```
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
```

Herkunfts- und Zugriffskontrolle online lesen: https://riptutorial.com/de/http/topic/3424/herkunfts--und-zugriffskontrolle

Kapitel 5: HTTP für APIs

Bemerkungen

HTTP-APIs verwenden ein breites Spektrum an HTTP-Verben und geben in der Regel JSON- oder XML-Antworten zurück.

Examples

Erstellen Sie eine Ressource

Nicht jeder ist sich einig, was die semantisch korrekte Methode zur Ressourcenerstellung ist. Daher kann Ihre API `POST` oder `PUT` Anforderungen annehmen oder eine der beiden.

Der Server sollte mit `201 Created` antworten, wenn die Ressource erfolgreich erstellt wurde. Wählen Sie den am besten geeigneten Fehlercode, falls nicht.

Wenn Sie beispielsweise eine API zum Erstellen von Mitarbeiterdatensätzen bereitstellen, könnte die Anforderung / Antwort folgendermaßen aussehen:

```
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"
        }
    ]
}
```

Durch das Einbinden der `links` JSON-Felder in die Antwort kann der Client auf eine Ressource zugreifen, die sich auf die neue Ressource und die gesamte Anwendung bezieht, ohne deren URIs oder Methoden zuvor kennen zu müssen.

Bearbeiten Sie eine Ressource

Das Bearbeiten oder Aktualisieren einer Ressource ist ein allgemeiner Zweck für APIs. Edits kann durch Senden entweder dadurch erreicht werden , `POST` , `PUT` oder `PATCH` - Anfragen an die jeweiligen Ressource. Obwohl es dem `POST` erlaubt ist, Daten an die vorhandene Darstellung einer Ressource anzuhängen, wird empfohlen, entweder `PUT` oder `PATCH` da diese eine explizitere Semantik vermitteln.

Ihr Server sollte mit `200 OK` antworten, wenn das Update durchgeführt wurde, oder `202 Accepted` falls es noch nicht angewendet wurde. Wählen Sie den am besten geeigneten Fehlercode, falls dieser nicht abgeschlossen werden kann.

Vollständige Updates

`PUT` hat die Semantik, die aktuelle Darstellung durch die in der Anforderung enthaltene Nutzlast zu ersetzen. Wenn die Payload nicht denselben Repräsentationstyp wie die aktuelle Repräsentation der zu aktualisierenden Ressource aufweist, kann der Server entscheiden, welcher Ansatz gewählt wird. RFC7231 definiert, dass der Server entweder kann

- Konfigurieren Sie die Zielressource neu, um den neuen Medientyp wiederzugeben
- Wandeln Sie die PUT-Darstellung in ein Format um, das mit dem der Ressource übereinstimmt, bevor Sie sie als neuen Ressourcenstatus speichern
- Ablehnen die Anforderung mit einer `415 Unsupported Media Type` unterstütztem `415 Unsupported Media Type` Antwort , den anzeigt , dass die Zielressource zu einem bestimmten (set) von Medientypen beschränkt ist.

Eine Basisressource mit einer JSON- HAL- Darstellung wie ...

```
{
    "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"
        ]
    }
}
```

... erhält möglicherweise eine Aktualisierungsanfrage

```
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"
}
```

Der Server kann jetzt den Status der Ressource durch den angegebenen Anforderungshauptteil ersetzen und auch den Inhaltstyp von `application/hal+json` in `application/json` ändern oder die JSON-Nutzdaten in eine JSON-HAL-Darstellung konvertieren und dann den Inhalt der Ressource ersetzen mit dem transformierten oder lehnen Sie die Aktualisierungsanforderung aufgrund eines nicht anwendbaren Medientyps mit einer Antwort vom Typ `415 Unsupported Media Type` Medientyp ab.

Es besteht ein Unterschied, ob Sie den Inhalt direkt ersetzen oder zuerst die Repräsentation in das definierte Repräsentationsmodell umwandeln und dann den vorhandenen Inhalt durch den transformierten ersetzen. Eine nachfolgende `GET` Anforderung gibt die folgende Antwort auf einen direkten Ersatz zurück:

```
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"
}
```

Während der Transformations- und dann Wiederherstellungsansatz wird die folgende Darstellung zurückgegeben:

```
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"
        ]
    }
}
```

Nebenwirkungen

Beachten Sie, dass `PUT` Nebenwirkungen haben darf, obwohl es als idempotenter Vorgang definiert ist! Dies ist in RFC7231 als dokumentiert

> Eine PUT-Anforderung, die auf die Zielressource angewendet wird, **kann Nebenwirkungen auf andere Ressourcen haben** . Beispielsweise kann ein Artikel einen URI zum Identifizieren der "aktuellen Version" (eine Ressource) haben, der von den URIs, die jede bestimmte Version identifizieren, getrennt ist (verschiedene Ressourcen, die an einem Punkt den gleichen Status wie die aktuelle Versionsressource hatten). Bei einer erfolgreichen PUT-Anforderung für den URI "Aktuelle Version" kann daher zusätzlich zur Änderung des Status der Zielressource eine neue Versionsressource erstellt werden. Außerdem können Verknüpfungen zwischen den zugehörigen Ressourcen hinzugefügt werden.

Das Erstellen zusätzlicher Protokolleinträge wird normalerweise nicht als Nebeneffekt betrachtet, da dies sicherlich kein Status einer Ressource im Allgemeinen ist.

Teilaktualisierungen

RFC7231 erwähnt dies bezüglich Teilaktualisierungen :

> Partial Inhaltsaktualisierungen sind möglich , indem eine separat identifiziert Ressource mit Targeting - Zustand, der einen Teil der größeren Ressource überlappt, oder durch eine andere Methode , die (beispielsweise das Verfahren in PATCH definiert für die partielle Updates speziell definiert wurde RFC5789).

Teilaktualisierungen können daher in zwei Varianten durchgeführt werden:

- Lassen Sie eine Ressource mehrere kleinere Subressourcen einbetten und aktualisieren Sie statt der gesamten Ressource nur eine entsprechende `PUT` über `PUT`
- Verwenden Sie `PATCH` und `PATCH` den Server an, was aktualisiert werden soll

Teilaktualisierung mit überlappendem Status

Wenn eine Benutzerrepräsentation aufgrund einer Verschiebung eines Benutzers an einen anderen Ort teilweise aktualisiert werden muss, anstatt den Benutzer direkt zu aktualisieren, sollte die zugehörige Ressource direkt aktualisiert werden, was eine teilweise Aktualisierung der Benutzerrepräsentation widerspiegelt.

Vor dem Umzug hatte ein Benutzer die folgende Darstellung

```
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" }
            }
        }
    }
}
```

Wenn der Benutzer an einen neuen Standort wechselt, aktualisiert er seine Standortinformationen wie folgt:

```
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"
}
```

Mit der Transformations-vor-Ersetzen-Semantik für den nicht übereinstimmenden Medientyp zwischen der vorhandenen Adressressource und der in der Anforderung (wie oben beschrieben) wird die Adressressource nun aktualisiert, was sich auf eine nachfolgende `GET` Anforderung an die Benutzerressource auswirkt Die neue Adresse für den Benutzer wird zurückgegeben.

```
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"
}
            }
        }
    }
}
```

Patchen von Teildaten

`PATCH` ist in RFC5789 definiert und ist nicht direkt Teil der HTTP-Spezifikation. Ein verbreiteter Irrglaube ist, dass das Senden nur der Felder, die teilweise aktualisiert werden sollten, innerhalb

einer `PATCH` Anforderung ausreicht . In der Spezifikation heißt es daher

> Die PATCH-Methode fordert, dass ein Satz von Änderungen, die in der Anforderungsentität beschrieben werden, auf die durch den Request-URI identifizierte Ressource angewendet wird. Die Änderungen werden in einem Format dargestellt, das als "Patch-Dokument" bezeichnet wird und durch einen Medientyp identifiziert wird.

Dies bedeutet, dass ein Client die erforderlichen Schritte zur Umwandlung der Ressource von Status A in Status B berechnen und diese Anweisungen an den Server senden muss.

Ein beliebter JSON-basierter Medientyp für das Patchen ist der JSON-Patch .

Wenn sich das Alter und die Berufsbezeichnung unseres Beispielbenutzers ändern und ein zusätzliches Feld, das das Einkommen des Benutzers darstellt, hinzugefügt werden sollte, könnte eine teilweise Aktualisierung mit `PATCH` mit JSON Patch folgendermaßen aussehen:

```
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` kann mehrere Ressourcen `PATCH` aktualisieren und erfordert, dass die Änderungen atomar angewendet werden. `PATCH` bedeutet, dass entweder alle Änderungen angewendet werden müssen oder keine, was den API-Implementierer mit Transaktionslast belastet.

Ein erfolgreiches Update kann so etwas zurückgeben

```
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
}
```

ist jedoch nicht nur auf `200 OK` Antwortcodes beschränkt.

Um Zwischenaktualisierungen zu verhindern (Änderungen zwischen dem vorherigen Abruf des Repräsentationsstatus und dem Aktualisierungsvorgang), sollten `ETag` , `If-Match` oder `If-Unmodified-Since` Header verwendet werden.

Fehlerbehandlung

Die Spezifikation auf `PATCH` empfiehlt die folgende Fehlerbehandlung:

Art	Fehlercode
Fehlerhaftes Patchdokument	`400 Bad Request`
Nicht unterstütztes Patch-Dokument	`415 Unsupported Media Type`
Nicht verarbeitbare Anfrage, dh wenn die Ressource durch Anwenden des Patches ungültig werden würde	`422 Unprocessable Entity`
Ressource nicht gefunden	`404 Not Found`
Konfliktzustand, dh ein Umbenennen (Verschieben) eines Feldes, das nicht vorhanden ist	`409 Conflict`
Widersprüchliche Änderung, dh wenn der Client einen `If-Match` oder `If-Unmodified-Since` Header verwendet, dessen Überprüfung fehlgeschlagen ist. Wenn keine Vorbedingung vorhanden war, sollte der letztere Fehlercode zurückgegeben werden	`412 Precondition Failed` oder `409 Conflict`
Gleichzeitige Änderung, dh wenn die Anforderung angewendet werden muss, bevor weitere `PATCH` Anforderungen `PATCH` werden	`409 Conflict`

Eine Ressource löschen

HTTP-APIs werden häufig verwendet, um eine vorhandene Ressource zu löschen. Dies sollte normalerweise mit `DELETE` Anforderungen erfolgen.

Wenn der Löschvorgang erfolgreich war, sollte der Server `200 OK` . ein entsprechender Fehlercode, falls nicht.

Wenn unser Mitarbeiter Charlie Smith das Unternehmen verlassen hat und wir jetzt seine Datensätze löschen möchten, könnte dies folgendermaßen aussehen:

```
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'
```

```
        }
    ]
}
```

Ressourcen auflisten

Die letzte häufige Verwendung von HTTP-APIs ist das Abrufen einer Liste der vorhandenen Ressourcen auf dem Server. Listen wie diese sollten mithilfe von `GET` Anforderungen abgerufen werden, da sie nur Daten *abrufen* .

Der Server sollte `200 OK` wenn er die Liste bereitstellen kann, oder einen entsprechenden Fehlercode, falls nicht.

Auflistung unserer Mitarbeiter könnte dann so aussehen:

```
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'
        }
    ]
}
```

HTTP für APIs online lesen: https://riptutorial.com/de/http/topic/3423/http-fur-apis

Kapitel 6: HTTP-Anfragen

Parameter

HTTP-Methode	Zweck
`OPTIONS`	Rufen Sie Informationen zu den Kommunikationsoptionen (verfügbare Methoden und Header) ab, die für den angegebenen Anforderungs-URI verfügbar sind.
`GET`	Rufen Sie die Daten ab, die durch den Anforderungs-URI identifiziert wurden, oder die Daten, die vom Skript erzeugt werden, das im Anforderungs-URI verfügbar ist.
`HEAD`	Identisch mit `GET` außer dass vom Server kein Nachrichtentext zurückgegeben wird: nur Header.
`POST`	Senden Sie einen (im Nachrichtentext angegebenen) Datenblock an den Server, um ihn der in der Anforderungs-URI angegebenen Ressource hinzuzufügen. Wird am häufigsten für die Formularverarbeitung verwendet.
`PUT`	Speichern Sie die eingeschlossenen Informationen (im Nachrichtentext) als neue oder aktualisierte Ressource unter dem angegebenen Anforderungs-URI.
`DELETE`	Löschen Sie die in der Anforderungs-URI angegebene Ressource, oder warten Sie sie zum Löschen.
`TRACE`	Im Wesentlichen ein Echo-Befehl: Ein funktionsfähiger, kompatibler HTTP-Server muss die gesamte Anforderung als Hauptteil einer 200-Antwort (OK) zurücksenden.

Bemerkungen

Die `CONNECT` Methode wird durch die Methodendefinitionsspezifikation für die Verwendung mit Proxys reserviert , die zwischen Proxying- und Tunneling-Modus wechseln können (z. B. für SSL-Tunneling).

Examples

Manuelles Senden einer minimalen HTTP-Anforderung mithilfe von Telnet

Dieses Beispiel zeigt, dass HTTP ein textbasiertes Internetkommunikationsprotokoll ist und eine grundlegende HTTP-Anforderung und die entsprechende HTTP-Antwort zeigt.

Mit Telnet können Sie wie folgt manuell eine minimale HTTP-Anforderung über die Befehlszeile senden.

1. Starten Sie eine Telnet-Sitzung mit dem Webserver `www.example.org` an Port 80:

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

   Telnet meldet, dass Sie mit dem Server verbunden sind:

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

2. Geben Sie eine Anforderungszeile zum Senden eines GET-Anforderungs-URL-Pfads `/` über HTTP 1.1 ein

   ```
   GET / HTTP/1.1
   ```

3. Geben Sie eine HTTP-Headerfeldzeile ein, um den Hostnamen-Teil der erforderlichen URL zu identifizieren, der in HTTP 1.1 erforderlich ist

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

4. Geben Sie eine leere Zeile ein, um die Anfrage abzuschließen.

   Der Webserver sendet die HTTP-Antwort, die in der Telnet-Sitzung angezeigt wird.

Die komplette Sitzung ist wie folgt. Die erste Zeile der Antwort ist die *HTTP-Statuszeile* , die den Statuscode 200 und den Statustext *OK enthält* . Dies zeigt an, dass die Anforderung erfolgreich verarbeitet wurde. Darauf folgen eine Reihe von HTTP-Header-Feldern, eine Leerzeile und die HTML-Antwort.

```
$ 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>
```

(`style` Element und Leerzeilen aus dem HTML reponse entfernt, der Kürze halber.)

Grundlegendes Anforderungsformat

In HTTP 1.1 besteht eine minimale HTTP-Anforderung aus einer Anforderungszeile und einem `Host` :

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

Die erste Zeile hat dieses Format:

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

`Method` sollte eine gültige HTTP-Methode sein. einer von [1] [2] :

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

`Request-URI` gibt entweder den URI oder den Pfad zu der Ressource an, die der Client anfordert. Es kann entweder sein:

- ein vollständig qualifizierter URI, einschließlich Schema, Host, (optionaler) Port und Pfad; oder
- ein Pfad. In diesem Fall muss der `Host` Hostheader angegeben werden

`HTTP-Version` gibt die Version des HTTP-Protokolls an, das der Client verwendet. Für HTTP 1.1-Anforderungen muss dies immer `HTTP/1.1` .

Die Anforderungszeile endet mit einem Wagenrücklauf-Zeilenvorschubpaar, normalerweise dargestellt durch `\r\n` .

Anforderungsheaderfelder

Header-Felder (normalerweise nur als "Header" bezeichnet) können einer HTTP-Anfrage hinzugefügt werden, um zusätzliche Informationen zur Anfrage bereitzustellen. Ein Header hat eine Semantik, die den Parametern ähnelt, die an eine Methode in einer beliebigen Programmiersprache übergeben werden, die solche Funktionen unterstützt.

Eine Anfrage mit `Host` , `User-Agent` und `Referer` Headern könnte folgendermaßen aussehen:

```
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
```

Eine vollständige Liste der unterstützten HTTP 1.1-Anforderungsheader finden Sie in der Spezifikation . Die häufigsten sind:

- `Host` - der Hostnamenteil der Anforderungs-URL (in HTTP / 1.1 erforderlich)
- `User-Agent` - eine Zeichenfolge, die den anfordernden Benutzeragenten darstellt;
- `Referer` - der URI, von dem aus der Client hierher `Referer` wurde; und
- `If-Modified-Since` - gibt ein Datum an, an dem der Server feststellen kann, ob sich eine Ressource geändert hat, und gibt an, dass der Client eine zwischengespeicherte Kopie verwenden kann, falls dies nicht der Fall ist.

Ein Header sollte als `Name: Value CRLF` . `Name` ist der `Name` der Kopfzeile, z. B. `User-Agent` . `Value` ist die ihm zugewiesene Daten, und die Zeile sollte mit einer CRLF enden. Die Namen der Headern unterscheiden nicht zwischen Groß- und Kleinschreibung und dürfen nur Buchstaben, Ziffern und die Zeichen `!#$%&'*+-.^_`|~` (RFC7230 Abschnitt 3.2.6 Feldwertkomponenten) verwenden.

Der Name des `Referer` Header-Feldes ist ein Tippfehler für 'Referrer', der versehentlich in RFC1945 eingeführt wurde .

Nachrichtenkörper

Einige HTTP-Anforderungen enthalten möglicherweise einen Nachrichtentext. Dies sind zusätzliche Daten, die der Server zur Verarbeitung der Anfrage verwendet. Nachrichtentexte werden am häufigsten in POST- oder PATCH- und PUT-Anforderungen verwendet, um neue Daten bereitzustellen, die der Server auf eine Ressource anwenden soll.

Anforderungen, die einen Nachrichtentext enthalten, sollten immer die Länge in Bytes mit `Content-Length` Header enthalten.

Ein Nachrichtentext ist *nach* allen Kopfzeilen und einer doppelten CRLF enthalten. Eine PUT-Beispielanforderung mit einem Body könnte folgendermaßen aussehen:

```
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` und `TRACE` Anforderungen dürfen keinen Nachrichtentext enthalten.

HTTP-Anfragen online lesen: https://riptutorial.com/de/http/topic/2909/http-anfragen

Kapitel 7: HTTP-Antworten

Parameter

Statuscode	Grund-Phrase - Beschreibung
100	**Fortfahren** - Der Client sollte den folgenden Teil einer mehrteiligen Anforderung senden.
101	**Protokollwechsel** - Der Server ändert die Version oder den Protokolltyp, der in dieser Kommunikation verwendet wird.
200	**OK** - Der Server hat die Anfrage des Clients erhalten und abgeschlossen.
201	**Erstellt** - Der Server hat die Anfrage akzeptiert und eine neue Ressource erstellt, die unter dem URI im Header `Location` verfügbar ist.
202	**Accepted** (**Akzeptiert)** - Der Server hat die Anfrage des Clients erhalten und akzeptiert, aber die Verarbeitung wurde noch nicht gestartet oder abgeschlossen.
203	**Nicht autorisierende Informationen** - Der Server gibt Daten zurück, die möglicherweise eine Unter- oder Obermenge der auf dem ursprünglichen Server verfügbaren Informationen sind. Wird hauptsächlich von Proxies verwendet.
204	**Kein Inhalt** - wird anstelle von 200 (OK) verwendet, wenn die Antwort keinen Text enthält.
205	**Inhalt zurücksetzen** - identisch mit 204 (kein Inhalt), der Client sollte jedoch die aktive Dokumentansicht neu laden.
206	**Partial Content** - wird anstelle von 200 (OK) verwendet, wenn der Client einen `Range` Header anfordert.
300	**Multiple Choices** (**Mehrfachauswahl)** - Die angeforderte Ressource ist für mehrere URIs verfügbar. Der Client sollte die Anforderung an einen in der Liste angegebenen URI im Nachrichtentext umleiten.
301	**Dauerhaft verschoben** - Die angeforderte Ressource ist für diesen URI nicht mehr verfügbar, und der Client sollte diese und alle zukünftigen Anforderungen an den im `Location` Header angegebenen URI umleiten.
302	**Gefunden** - Die Ressource befindet sich vorübergehend unter einem anderen URI. Diese Anforderung sollte bei Bestätigung durch den Benutzer an den URI im `Location` Header weitergeleitet werden, zukünftige Anforderungen sollten jedoch nicht geändert werden.

Statuscode	Grund-Phrase - Beschreibung
303	**Siehe Sonstiges** - sehr ähnlich zu 302 (gefunden), erfordert jedoch keine Benutzereingaben, um an den bereitgestellten URI umzuleiten. Der bereitgestellte URI sollte mit einer GET-Anforderung abgerufen werden.
304	**Nicht geändert** - Der Client hat einen `If-Modified-Since` oder einen ähnlichen Header gesendet, und die Ressource wurde seitdem nicht geändert. Der Client sollte eine zwischengespeicherte Kopie der Ressource anzeigen.
305	**Proxy verwenden** - Die angeforderte Ressource muss erneut über den im Kopfzeilenfeld `Location` angegebenen Proxy angefordert werden.
307	**Temporäre Umleitung** - identisch mit 302 (Found), aber HTTP 1.0-Clients unterstützen keine 307 Antworten.
400	Fehlerhafte **Anforderung** - Der Client hat eine fehlerhafte Anforderung mit Syntaxfehlern gesendet und sollte die Anforderung ändern, um diese zu korrigieren, bevor sie wiederholt wird.
401	**Nicht autorisiert** - Die angeforderte Ressource ist ohne Authentifizierung nicht verfügbar. Der Client kann die Anforderung unter Verwendung eines `Authorization` Headers wiederholen, um Authentifizierungsdetails anzugeben.
402	**Zahlung erforderlich** - reservierter, nicht spezifizierter Statuscode für Anwendungen, für die Benutzerabonnements erforderlich sind, um Inhalte anzuzeigen.
403	**Verboten** : Der Server versteht die Anforderung, lehnt sie jedoch aufgrund vorhandener Zugriffskontrollen ab. Die Anfrage sollte nicht wiederholt werden.
404	**Nicht gefunden** - Auf diesem Server ist keine Ressource verfügbar, die dem angeforderten URI entspricht. Kann anstelle von 403 verwendet werden, um zu vermeiden, dass Details der Zugriffskontrolle offengelegt werden.
405	**Methode nicht zulässig** - Die Ressource unterstützt die Anforderungsmethode (HTTP-Verb) nicht. Im Header " `Allow` akzeptable Anforderungsmethoden aufgeführt.
406	**Nicht akzeptabel** - Die Ressource weist Eigenschaften auf, die gegen die in der Anforderung gesendeten Accept-Header verstoßen.
407	**Proxy-Authentifizierung erforderlich** - Ähnlich wie 401 (Unauthorized), gibt jedoch an, dass sich der Client zuerst beim Zwischen-Proxy authentifizieren muss.
408	**Request Timeout** (**Anforderungszeitlimit)** : Der Server hat eine andere Anforderung vom Client erwartet, jedoch wurde keine innerhalb eines akzeptablen Zeitraums bereitgestellt.

Statuscode	Grund-Phrase - Beschreibung
409	**Konflikt** - Die Anforderung konnte nicht abgeschlossen werden, da sie mit dem aktuellen Status der Ressource in Konflikt stand.
410	**Gegangen** - ähnlich wie 404 (nicht gefunden), weist jedoch auf eine dauerhafte Entfernung hin. Es ist keine Weiterleitungsadresse verfügbar.
411	**Erforderliche Länge** : Der Client hat keinen gültigen `Content-Length` Header angegeben und muss dies tun, bevor der Server diese Anforderung akzeptiert.
412	**Voraussetzung fehlgeschlagen** - Die Ressource ist nicht mit allen Bedingungen verfügbar, die in den vom Client gesendeten bedingten Headern angegeben sind.
413	**Request Entity Too Large** - Der Server kann einen Nachrichtentext in der vom Client gesendeten Länge derzeit nicht verarbeiten.
414	**Request-URI zu lang** - Der Server lehnt die Anfrage ab, da der Request-URI länger ist, als der Server interpretieren möchte.
415	**Nicht** unterstützter Medientyp - Der Server unterstützt den vom Client angegebenen MIME-Typ oder Medientyp nicht und kann diese Anforderung nicht bearbeiten.
416	**Angeforderter Bereich nicht zufriedenstellend** : Der Client hat einen **Bytebereich** angefordert, der Server kann der Spezifikation jedoch keinen Inhalt bereitstellen.
417	**Erwartung fehlgeschlagen** - Der Client hat Einschränkungen im `Expect` Header angegeben, die der Server nicht erfüllen kann.
500	**Internal Server Error** (**Interner Serverfehler)** : Der Server hat eine unerwartete Bedingung oder einen Fehler erfüllt, der die Ausführung dieser Anforderung verhindert.
501	**Nicht implementiert** - Der Server unterstützt nicht die zum Abschließen der Anforderung erforderliche Funktionalität. Wird normalerweise verwendet, um eine Anforderungsmethode anzugeben, die für *keine* Ressource unterstützt wird.
502	**Bad Gateway** - Der Server ist ein Proxy und hat beim Verarbeiten dieser Anforderung eine ungültige Antwort vom Upstream-Server erhalten.
503	**Dienst nicht verfügbar** - Der Server ist stark **ausgelastet** oder wird **gewartet** und kann diese Anforderung derzeit nicht bedienen.
504	**Gateway-Timeout** - Der Server ist ein Proxy und hat nicht rechtzeitig eine Antwort vom Upstream-Server erhalten.

Statuscode	Grund-Phrase - Beschreibung
505	**HTTP-Version nicht unterstützt** - Der Server unterstützt nicht die Version des HTTP-Protokolls, mit der der Client eine Anforderung gestellt hat.

Examples

Grundlegendes Antwortformat

Wenn ein HTTP-Server eine wohlgeformte HTTP-Anforderung erhält, muss er die in der Anforderung enthaltenen Informationen verarbeiten und eine Antwort an den Client zurückgeben. Eine einfache HTTP 1.1-Antwort kann wie eine der folgenden aussehen, normalerweise gefolgt von einer Reihe von Headerfeldern und möglicherweise einem Antworttext:

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

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

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

Eine einfache HTTP 1.1-Antwort hat folgendes Format:

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

Wie in einer Anfrage gibt die `HTTP-Version` die Version des verwendeten HTTP-Protokolls an. Für HTTP 1.1 muss dies immer die Zeichenfolge `HTTP/1.1` .

`Status-Code` ist ein dreistelliger Code, der den Status der Anfrage des Kunden angibt. Die erste Ziffer dieses Codes ist die *Statusklasse* , die den Statuscode in eine von fünf Antwortkategorien [1] einordnet :

- `1xx` **Informational** - Der Server hat die Anfrage erhalten und die Verarbeitung wird fortgesetzt
- `2xx` **Erfolg** - Der Server hat die Anfrage akzeptiert und verarbeitet
- `3xx` **Umleitung** - Der Client muss weitere Schritte `3xx` , um die Anforderung abzuschließen
- `4xx` **Client-Fehler** - Der Client hat eine Anfrage gesendet, die fehlerhaft war oder nicht erfüllt werden konnte
- `5xx` **Server-Fehler** - Die Anforderung war gültig, kann jedoch derzeit vom Server nicht erfüllt werden

`Reason-Phrase` ist eine kurze Beschreibung des Statuscodes. Beispielsweise hat der Code `200` eine Grundphrase von `OK` ; Code `404` hat eine Phrase von `Not Found` . Eine vollständige Liste der Begründungsausdrücke finden Sie in den folgenden Parametern oder in der HTTP-Spezifikation .

Die Zeile endet mit einem Wagenrücklauf-Zeilenvorschubpaar, normalerweise dargestellt durch `\r\n` .

Zusätzliche Header

Wie eine HTTP-Anfrage kann eine HTTP-Antwort zusätzliche Header enthalten, um die von ihr bereitgestellte Antwort zu ändern oder zu erweitern.

Eine vollständige Liste der verfügbaren Header ist in §6.2 der Spezifikation definiert . Die am häufigsten verwendeten Header sind:

- `Server` , der wie ein `User-Agent` Anforderungsheader für den Server fungiert;
- `Location` , der in den Statusantworten 201 und 3xx verwendet wird, um einen URI anzugeben, an den weitergeleitet werden soll; und
- `ETag` , eine eindeutige Kennung für diese Version der zurückgegebenen Ressource, damit Clients die Antwort im Cache speichern können.

Antwortheader stehen hinter der Statuszeile und werden wie bei Anforderungsheatern als solche gebildet:

```
Name: Value CRLF
```

`Name` gibt den Headernamen an, z. B. `ETag` oder `Location` , und `Value` gibt den Wert an, den der Server für diesen Header festlegt. Die Zeile endet mit einer CRLF.

Eine Antwort mit Kopfzeilen könnte folgendermaßen aussehen:

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

Nachrichtenkörper

Wie bei Anforderungskörpern können HTTP-Antworten einen Nachrichtentext enthalten. Dadurch werden zusätzliche Daten bereitgestellt, die der Client verarbeitet. 200 OK-Antworten auf eine wohlgeformte GET-Anforderung sollten immer einen Nachrichtentext enthalten, der die angeforderten Daten enthält. (Wenn dies nicht der Fall ist, ist 204 No Content eine angemessenere Antwort).

Ein Nachrichtentext ist nach allen Kopfzeilen und einer doppelten CRLF enthalten. Bei Anfragen sollte die Länge in Byte mit dem `Content-Length` Header angegeben werden. Eine erfolgreiche Antwort auf eine GET-Anfrage könnte daher wie folgt aussehen:

```
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.
```

HTTP-Antworten online lesen: https://riptutorial.com/de/http/topic/3077/http-antworten

Kapitel 8: HTTP-Antworten zwischenspeichern

Bemerkungen

Die Antworten werden für jede URL und jede HTTP-Methode separat zwischengespeichert.

HTTP-Caching ist in RFC 7234 definiert.

Glossar

- **fresh** - Status einer zwischengespeicherten Antwort, die noch nicht abgelaufen ist. In der Regel kann eine neue Antwort Anforderungen *erfüllen* , ohne dass der Server kontaktiert werden muss, von dem die Antwort stammt.
- **stale** - Zustand einer zwischengespeicherten Antwort, deren Ablaufdatum überschritten ist. Normalerweise können veraltete Antworten nicht verwendet werden, *um* eine Anforderung zu *erfüllen,* ohne beim Server zu prüfen, ob sie noch gültig ist.
- **Eine befriedigte** zwischengespeicherte Antwort *erfüllt* eine Anforderung, wenn alle Bedingungen in der Anforderung mit der zwischengespeicherten Antwort übereinstimmen, z. B. dieselbe HTTP-Methode und URL, die Antwort neu ist oder die Anforderung veraltete Antworten zulässt, Anforderungsheader stimmen mit den im Antwortkopf " `Vary` Header überein .
- **Revalidierung** - Prüfen , ob eine zwischengespeicherte Antwort frisch ist. Dies erfolgt normalerweise mit einer *bedingten Anforderung,* die `If-Modified-Since` oder `If-None-Match` und den Antwortstatus `304` .
- **privater Cache** - Cache für einen einzelnen Benutzer, z. B. in einem Webbrowser. Private Caches können personalisierte Antworten speichern.
- **Öffentlicher Cache** - Cache, der von vielen Benutzern gemeinsam genutzt wird, z. B. in einem Proxyserver. Ein solcher Cache kann dieselbe Antwort an mehrere Benutzer senden.

Examples

Cache-Antwort für alle für 1 Jahr

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

`public` bedeutet, dass die Antwort für alle Benutzer gleich ist (sie enthält keine personalisierten Informationen). `max-age` ist in wenigen Sekunden. 31536000 = 60 * 60 * 24 * 365.

Dies wird für statische Assets empfohlen, die sich niemals ändern sollen.

Eine personalisierte Antwort für 1 Minute zwischenspeichern

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

`private` gibt an, dass die Antwort nur für Benutzer zwischengespeichert werden kann, die die Ressource angefordert haben, und nicht erneut verwendet werden kann, wenn andere Benutzer dieselbe Ressource anfordern. Dies ist angemessen für Antworten, die von Cookies abhängen.

Stoppen Sie die Verwendung von zwischengespeicherten Ressourcen, ohne vorher den Server zu überprüfen

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

Der Client verhält sich so, als wäre die Antwort nicht zwischengespeichert worden. Dies ist angemessen für Ressourcen, die jederzeit unvorhersehbar geändert werden können und die Benutzer immer in der neuesten Version sehen müssen.

Antworten mit `no-cache` werden langsamer (hohe Latenzzeit), da der Server bei jeder Verwendung erneut kontaktiert werden muss.

Doch um Bandbreite zu sparen, *können* immer noch die Kunden solche Antworten speichern. Antworten `no-cache` werden nicht verwendet, um Anforderungen zu erfüllen, ohne den Server jedes Mal zu kontaktieren, um zu prüfen, ob die zwischengespeicherte Antwort erneut verwendet werden kann.

Beantworten Sie die Antworten nicht, um überhaupt gespeichert zu werden

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

Weist Clients an, die Antwort auf keine Weise zu zwischenspeichern und so schnell wie möglich zu vergessen.

Diese Direktive wurde ursprünglich für sensible Daten entwickelt (heutzutage sollte stattdessen HTTPS verwendet werden), sie kann jedoch verwendet werden, um die Verschmutzung von Caches zu vermeiden, deren Antworten nicht wiederverwendet werden können.

Dies ist nur in bestimmten Fällen sinnvoll, in denen die Antwortdaten immer unterschiedlich sind, z. B. ein API-Endpunkt, der eine große Zufallszahl zurückgibt. Andernfalls können `no-cache` und Revalidierung verwendet werden, um ein Verhalten einer "nicht cachbaren" Antwort zu haben, während dennoch etwas Bandbreite eingespart werden kann.

Veraltete, redundante und nicht standardmäßige Header

- `Expires` - gibt das Datum an, an dem die Ressource `Expires` . Es setzt voraus, dass Server und Clients genaue Uhren haben und Zeitzonen richtig unterstützen. `Cache-control: max-age` hat Vorrang vor `Expires` und ist im Allgemeinen zuverlässiger.
- `post-check` und `pre-check` - Richtlinien sind nicht-Standard - Internet - Explorer - Erweiterungen , die Verwendung von veralteten Antworten ermöglichen. Die

Standardalternative ist " `stale-while-revalidate` .

- `Pragma: no-cache` - 1999 veraltet. Stattdessen sollte die `Cache-control` verwendet werden.

Zwischengespeicherte Ressourcen ändern

Die einfachste Methode, den Cache zu umgehen, besteht darin, die URL zu ändern. Dies wird als bewährte Methode verwendet, wenn die URL eine Version oder eine Prüfsumme der Ressource enthält, z

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

Diese beiden URLs werden separat zwischengespeichert. Selbst wenn `…?version=1` für *immer* zwischengespeichert wurde, konnte eine neue Kopie sofort als `…?version=2` abgerufen werden.

Bitte verwenden Sie keine zufälligen URLs, um Caches zu umgehen. Verwenden Sie `Cache-control: no-cache` oder `Cache-control: no-store` . Wenn Antworten mit zufälligen URLs ohne die Direktive `no-store` gesendet werden, werden sie unnötig in Caches gespeichert und nützlichere Antworten aus dem Cache herausgeschoben, wodurch die Leistung des gesamten Caches beeinträchtigt wird.

HTTP-Antworten zwischenspeichern online lesen: https://riptutorial.com/de/http/topic/3296/http-antworten-zwischenspeichern

Kapitel 9: HTTP-Statuscodes

Einführung

Statuscodes sind in HTTP ein maschinenlesbarer Mechanismus, der das Ergebnis einer zuvor ausgegebenen Anforderung angibt. Aus RFC 7231, sek. 6 : "Das Statuscode-Element ist ein dreistelliger Ganzzahlcode, der das Ergebnis des Versuchs anzeigt, die Anforderung zu verstehen und zu erfüllen."

Durch die formale Grammatik können Codes zwischen `000` und `999` . Nur der Bereich von `100` bis `599` hat jedoch Bedeutung zugewiesen.

Bemerkungen

HTTP / 1.1 definiert eine Anzahl numerischer HTTP-Statuscodes , die in der Statuszeile (erste Zeile einer HTTP-Antwort) angezeigt werden, um zusammenzufassen, was der Client mit der Antwort tun soll.

Die erste Ziffer eines Statuscodes definiert die Antwortklasse:

- `1xx` informativ
- `2xx` erfolgreich
- `3xx` Request umgeleitet - weitere Aktion erforderlich, z. B. eine neue Anfrage
- `4xx` Client-Fehler - Wiederholen Sie dieselbe Anfrage nicht
- `5xx` Serverfehler - versuchen Sie es vielleicht noch einmal

In der Praxis ist es nicht immer einfach, den am besten geeigneten Statuscode auszuwählen.

Examples

500 Interner Serverfehler

Ein **interner HTTP 500-Serverfehler** ist eine allgemeine Nachricht, die besagt, dass der Server auf etwas Unerwartetes gestoßen ist. Anwendungen (oder der übergeordnete Webserver) sollten eine 500 verwenden, wenn bei der Verarbeitung der Anforderung ein Fehler aufgetreten ist - dh eine Ausnahme wird ausgelöst oder ein Zustand der Ressource verhindert, dass der Prozess abgeschlossen wird.

Beispiel Statuszeile:

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

404 Nicht gefunden

HTTP 404 nicht gefunden bedeutet, dass der Server den Pfad nicht mit der vom Client

angeforderten URI finden konnte.

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

Meistens wurde die angeforderte Datei gelöscht, aber manchmal kann es sich um eine fehlerhafte Konfiguration des Dokumentstamms oder fehlende Berechtigungen handeln (obwohl fehlende Berechtigungen häufiger HTTP 403 Forbidden auslösen).

Beispielsweise schreibt Microsoft IIS 404.0 (0 ist der Unterstatus) in seine Protokolldateien, wenn die angeforderte Datei gelöscht wurde. Wenn jedoch die eingehende Anforderung durch Anforderungsfilterregeln blockiert wird, schreibt sie 404.5-404.19 in Protokolldateien, nach denen die Anforderung die Anforderung blockiert. Eine detailliertere Fehlercode-Referenz finden Sie beim Microsoft Support .

Zugriff auf geschützte Dateien verweigern

Verwenden Sie 403 Verboten, wenn ein Client eine Ressource angefordert hat, auf die aufgrund vorhandener Zugriffskontrollen nicht zugegriffen werden kann. Wenn Ihre App beispielsweise über eine `/admin` Route verfügt, auf die nur Benutzer mit Administratorrechten zugreifen dürfen, können Sie 403 verwenden, wenn ein normaler Benutzer die Seite anfordert.

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

```
HTTP/1.1 403 Forbidden
```

Erfolgreiche Anfrage

Senden Sie eine HTTP-Antwort mit Statuscode `200` , um eine erfolgreiche Anforderung anzuzeigen. Die HTTP-Antwortstatuszeile lautet dann:

```
HTTP/1.1 200 OK
```

Der Statustext `OK` ist nur informativ. Der Antworttext (Message Payload) sollte eine Darstellung der angeforderten Ressource enthalten. Wenn keine Repräsentation vorhanden ist, sollte kein Inhalt verwendet werden.

Antworten auf eine bedingte Anforderung für zwischengespeicherten Inhalt

Senden Sie als Antwort auf eine Clientanforderung, die die Kopfzeilen `If-Modified-Since` und `If-None-Match` enthält, einen Antwortstatus " **304 Not Modified", der** vom Server gesendet wurde.

Wenn beispielsweise eine Clientanforderung für eine Webseite den Header `If-Modified-Since: Fri, 22 Jul 2016 14:34:40 GMT` und die Seite seitdem nicht geändert wurde, antworten Sie mit der Statuszeile `HTTP/1.1 304 Not Modified`

Top 10 HTTP-Statuscode

2xx Erfolg

- **200 OK** - Standardantwort für erfolgreiche HTTP-Anforderungen.
- **201 Created** - Die Anforderung wurde erfüllt, wodurch eine neue Ressource erstellt wurde.
- **204 No Content** - Der Server hat die Anforderung erfolgreich verarbeitet und gibt keinen Inhalt zurück.

3xx Umleitung

- **304 Not Modified** (**Nicht geändert)** - Gibt an, dass die Ressource seit der durch die Anforderungsheader `If-Modified-Since` oder `If-None-Match` angegebenen Version nicht geändert wurde.

4xx Clientfehler

- **400 Falsche Anforderung** - Der Server kann oder kann die Anfrage aufgrund eines offensichtlichen Clientfehlers (z. B. fehlerhafte Anforderungssyntax, zu große Größe, ungültiges Anforderungsnachrichten-Framing oder falsches Anforderungsrouting) nicht verarbeiten.
- **401 Unauthorized** - *Ähnlich wie 403 Forbidden* , jedoch besonders für die Verwendung, wenn eine Authentifizierung erforderlich ist und fehlgeschlagen ist oder noch nicht bereitgestellt wurde. Die Antwort muss ein `WWW-Authenticate` Headerfeld enthalten, das eine für die angeforderte Ressource geltende Herausforderung enthält.
- **403 Verboten** - Die Anfrage war eine gültige Anfrage, der Server weigert sich jedoch, darauf zu antworten. Der Benutzer ist möglicherweise angemeldet, verfügt jedoch nicht über die erforderlichen Berechtigungen für die Ressource.
- **404 nicht gefunden** - Die angeforderte Ressource wurde nicht gefunden, ist jedoch möglicherweise in der Zukunft verfügbar. Nachträgliche Anfragen des Auftraggebers sind zulässig.
- **409 Konflikt** - Zeigt an, dass die Anforderung aufgrund eines Konflikts in der Anforderung nicht verarbeitet werden konnte, beispielsweise aufgrund eines Bearbeitungskonflikts zwischen mehreren gleichzeitigen Aktualisierungen.

5xx Serverfehler

- **500 Internal Server Error** (**Interner Serverfehler)** - Eine generische Fehlermeldung, die angezeigt wird, wenn eine unerwartete Bedingung aufgetreten ist und keine speziellere Nachricht geeignet ist.

HTTP-Statuscodes online lesen: https://riptutorial.com/de/http/topic/2577/http-statuscodes

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit HTTP	Community, DaSourcerer, Kornel, Peter Hilton
2	Antwortkodierungen und Komprimierung	Jeff Bencteux, Peter Hilton
3	Authentifizierung	DaSourcerer, Peter Hilton, Stefan Kögl
4	Herkunfts- und Zugriffskontrolle	ArtOfCode
5	HTTP für APIs	ArtOfCode, mnoronha, Peter Hilton, Roman Vottner
6	HTTP-Anfragen	artem, ArtOfCode, Jeff Bencteux, Peter Hilton
7	HTTP-Antworten	ArtOfCode, Jeff Bencteux, Peter Hilton
8	HTTP-Antworten zwischenspeichern	DaSourcerer, Kornel
9	HTTP-Statuscodes	ArtOfCode, DaSourcerer, Deltik, Kornel, Lex Li, mnoronha, Peter Hilton, Rptk99, Sender, Xevaquor