Representational State Transfer

aus Wikipedia, der freien Enzyklopädie

Representational State Transfer (mit dem Akronym **REST**, seltener auch **ReST**) bezeichnet ein Programmierparadigma für Webanwendungen. Es gibt keine explizite Norm, daher gehen die Vorstellungen, was REST ist, auseinander. Im Grunde bezeichnet REST die Idee, dass eine URL genau einen Seiteninhalt als Ergebnis einer serverseitigen Aktion (etwa das Anzeigen einer Trefferliste nach einer Suche) darstellt, wie es der Internetstandard HTTP für statische Inhalte (Permalink) bereits vorsieht, ein Ziel, das für dynamisch erzeugte Seiten mitunter jedoch zusätzlichen Aufwand erfordert.

Inhaltsverzeichnis

Geschichte

REST stammt aus der Dissertation von Roy Fielding aus dem Jahr 2000, in der er den Erfolg des World Wide Webs auf bestimmte Eigenschaften der verwendeten Mechanismen und Protokolle (z. B. HTTP) zurückführt. Fielding war zuvor auch an der Spezifikation des Hypertext-Transfer-Protokolls (HTTP) beteiligt.

REST gilt in seiner Konsequenz als eine wichtige Richtlinie für die Nutzung von Web-Standards, in Abgrenzung zu vielen historisch gewachsenen Verfahren. Daraus folgt teils eine Rückbesinnung auf grundlegende Web-Technologien, die die Implementierung verteilter webbasierter Systeme vereinfachen soll.

Prinzipien

Es gibt fünf Eigenschaften, die ein REST-Dienst haben muss, wobei es den einzelnen Diensten überlassen ist, wie es implementiert wird.

Adressierbarkeit

Jeder Dienst, den ein REST-konformer Server zur Verfügung stellt, hat eine eindeutige Adresse, den Uniform Resource Locator (URL). Diese „Straße und Hausnummer im Netz“ stellt sicher, dass das Angebot eines Webservices auf einfache, standardisierte Art und Weise einer Vielzahl von Anwendungen (Clients) zur Verfügung steht. Eine konsistente Adressierbarkeit erleichtert es außerdem, einen Webservice als Teil eines Mashups weiterzuverwenden.

Zusätzlich zur URL, der das Service selbst adressiert, verwendet REST auch Uniform Resource Identifier (URI), um einzelne Ressourcen zu bezeichnen.

Unterschiedliche Repräsentationen

Die unter einer Adresse zugänglichen Dienste können unterschiedliche Darstellungsformen (Repräsentationen) haben. Ein REST-konformer Server kann je nachdem, was die Anwendung anfordert, verschiedene Repräsentationen ausliefern (z. B. HTML, JSON oder XML). Damit kann der Standard-Webbrowser über die gleiche URI-Adresse sowohl den Dienst, als auch die Beschreibung oder Dokumentation abrufen. Es wird nicht der interne Datenbankeintrag ausgeliefert, sondern das je nach Anfrage evtl. in eine bestimmte Kodierung oder Sprachversion übertragene Dokument.

Zustandslosigkeit

REST setzt auf ein zustandsloses Client-Server-Protokoll, wobei im Normalfall HTTP oder HTTPS eingesetzt wird. Jede REST-Nachricht enthält dabei alle Informationen, die für den Server bzw. Client notwendig sind, um die Nachricht zu verstehen. Weder der Server noch die Anwendung soll Zustandsinformationen zwischen zwei Nachrichten speichern. Eine derartig strikte Trennung der Zuständigkeiten zwischen Client und Server führt dazu, dass ein REST-konformer Webservice als zustandslos (englisch: *stateless*) bezeichnet werden kann. Jede Anfrage eines Clients an den Server ist in dem Sinne in sich geschlossen, als dass sie sämtliche Informationen über den Anwendungszustand beinhaltet, die vom Server für die Verarbeitung der Anfrage benötigt werden.

Zustandslosigkeit in der hier beschriebenen Form wirkt sich begünstigend auf die Skalierbarkeit eines Webservices aus. Beispielsweise können eingehende Anfragen im Zuge der Lastverteilung unkompliziert auf beliebige Maschinen verteilt werden: Da jeder Request in sich geschlossen ist und Anwendungsinformationen somit ausschließlich auf der Seite des Clients vorgehalten werden, ist auf der Seite des Servers keine Sitzungsverwaltung erforderlich. In der Praxis nutzen jedoch viele HTTP-basierte Anwendungen Cookies und andere Techniken, um Zustandsinformationen zu behalten.

Operationen

REST-konforme Dienste müssen einige Operationen vorsehen, die auf alle Dienste (*Ressourcen* genannt) angewendet werden können. HTTP zum Beispiel hat eine klar definierte Menge von Operationen, darunter `GET`, `POST`, `PUT` und `DELETE`.

HTTP schreibt vor, dass `GET` „sicher“ (englisch *safe*) sein muss, was bedeutet, dass diese Methode nur Informationen beschafft und keine sonstigen Effekte verursacht. Die Methoden `GET`, `HEAD`, `PUT` und `DELETE` müssen laut HTTP-Spezifikation idempotent sein, was in diesem Zusammenhang bedeutet, dass das mehrfache Absenden der gleichen Anforderung sich nicht anders auswirkt als ein einzelner Aufruf.[1]

Verwendung von Hypermedia

Sowohl für Anwendungsinformationen als auch für Zustandsveränderungen werden Hypermedia benutzt: Repräsentationen in einem REST-System sind typischerweise im HTML- oder XML-Format, welche sowohl Informationen als auch Links zu anderen Ressourcen enthalten. Deshalb ist es oftmals möglich, von einer Ressource zu einer anderen zu navigieren, indem man einfach Verknüpfungen folgt, ohne dass dafür Registrierungsdatenbanken oder ähnliche Infrastrukturen erforderlich sind. Diese Verknüpfung von Ressourcen innerhalb einer REST-Architektur wird auch als *Verbindungshaftigkeit* bezeichnet.

Umsetzung

REST vereinheitlicht die Schnittstelle zwischen Systemen auf eine überschaubare und – bezüglich des zu erwartenden Verhaltens – standardisierte Menge von Aktionen („Verben“). Welche Aktionen dies sind, ist in REST nicht festgelegt, aber alle Aktionen sind allgemein definiert.

Für die Umsetzung des REST-Architekturstils werden meist Internet-Technologien verwendet, als Anwendungsschicht-Protokoll dient hauptsächlich HTTP.

Der Client kann folgende Befehle absetzen

Befehl (HTTP-Verb)	Beschreibung	Anmerkungen
GET	fordert die angegebene Ressource vom Server an. GET weist keine Nebeneffekte auf. Der Zustand am Server wird nicht verändert, weshalb GET als *sicher* bezeichnet wird.	[n 1]
POST	fügt eine neue (Sub-)Ressource unterhalb der angegebenen Ressource ein. Da die neue Ressource noch keine URI besitzt, adressiert den URI die übergeordnete Ressource. Als Ergebnis wird der neue Ressourcenlink dem Client zurückgegeben. POST kann im weiteren Sinne auch dazu verwendet werden Operationen abzubilden, die von keiner anderen Methode abgedeckt werden.	[n 2]
PUT	die angegebene Ressource wird angelegt. Wenn die Ressource bereits existiert wird sie geändert.	[n 3]
PATCH	ein Teil der angegeben Ressource wird geändert. Hierbei sind Nebeneffekte erlaubt.	[n 4]
DELETE	löscht die angegebene Ressource. Wenn der Client versucht eine Ressource zu löschen die nicht existiert, bzw. bereits gelöscht wurde, erhält der Client – sofern die REST-Schnittstelle korrekt implementiert wurde – keine Fehlermeldung (*siehe auch:* HTTP-Statuscodes). Abhängig von der Implementierung wird eine Ressource meist – entgegen der HTTP-Spezifikation – nicht physisch gelöscht, sondern nur als gelöscht gekennzeichnet und somit versteckt und deaktiviert.	[n 3]
HEAD	fordert Metadaten zu einer Ressource vom Server an.	[n 4][n 1]
OPTIONS	prüft, welche Methoden auf einer Ressource zur Verfügung stehen.	[n 4][n 1]
CONNECT	Dient dazu die Anfrage durch einen TCP-Tunnel zu leiten. Wird meist eingesetzt um eine HTTPS-Verbindung über einen HTTP-Proxy herzustellen.	[n 4][n 1][n 5]
TRACE	Gibt die Anfrage zurück, wie sie der Zielserver erhält. Dient etwa dazu um Änderungen der Anfrage durch Proxyserver zu ermitteln.	[n 4][n 1][n 5]

1. nullipotent. Ein Aufruf dieser Methoden führt zu keinen Nebeneffekten.
2. nicht idempotent. Ein erneuter Aufruf erstellt für jeden Aufruf mit derselben URI ein neues Objekt, anstatt dasselbe Objekt zurückzugeben.
3. idempotent. Der erste Aufruf dieser Methoden mit einer bestimmten URI führt zu Nebeneffekten. Ein erneuter Aufruf mit derselben URI führt zu keinen weiteren Nebeneffekten.
4. optional. Wird nicht für CRUD-Operationen benötigt.
5. Eine Implementierung dieser Methoden wirkt sich ggf. auf die Sicherheit der Anwendung aus.

Im Gegensatz zu vielen RPC-Architekturen kodiert REST keine Methodeninformation in den URI, da der URI Ort und Namen der Ressource angibt, nicht aber die Funktionalität, die die Ressource anbietet. Es wird versucht, die Funktionalität hauptsächlich über die HTTP-Verben abzubilden.

Sicherheit

In der Theorie verlangt das Paradigma, dass *alle* Informationen, die eine Anwendung braucht, um den Seitenzustand wieder herzustellen, in der Anfrage enthalten sind. Dabei identifiziert der URI die Ressource während im HTTP-Header Informationen wie Zugriffsart (GET, PUT), Rückgabeformat oder Authentifizierung enthalten sein können.

In seiner Reinform lässt sich REST für Webseiten und Webservices verwenden, die keine Authentifizierung erfordern oder diese auf anderem Wege (beispielsweise durch Prüfung der IP-Adresse oder über HTTP-Authentifizierung) erreichen, wodurch bereits viele Anwendungsfälle abgedeckt werden können. Alternativ kann beispielsweise auch HMAC, OAuth oder OpenID verwendet werden.

Da REST – im Gegensatz zu SOAP mit WS-Security – selbst keine Verschlüsselungsmethoden definiert, wird bei sicherheitskritischen Nachrichten ein verschlüsseltes Transportprotokoll wie HTTPS verwendet.

HATEOAS

Hypermedia as the Engine of Application State, kurz **HATEOAS**, ist ein Entwurfsprinzip von REST-Architekturen. Bei HATEOAS navigiert der Client einer REST-Schnittstelle ausschließlich über URLs, welche vom Server bereitgestellt werden.

Abhängig von der gewählten Repräsentation geschieht die Bereitstellung der URIs über Hypermedia, also z. B.

- in Form von „href“- und „src“-Attributen bei HTML-Dokumenten bzw. HTML-Snippets, oder
- in für die jeweilige Schnittstelle definierten und dokumentierten JSON- bzw. XML-Attributen bzw. -Elementen.

Abstrakt betrachtet stellen HATEOAS-konforme REST-Services einen endlichen Automaten dar, dessen Zustandsveränderungen durch die Navigation mittels der bereitgestellten URIs erfolgt.

Durch HATEOAS ist eine lose Bindung gewährleistet und die Schnittstelle kann verändert werden. Im Gegensatz dazu kommuniziert ein SOAP-basiertes Webservice über ein fixiertes Interface. Für eine Änderung des Services muss hier eine neue Schnittstelle bereitgestellt werden und in der Schnittstellenbeschreibung (ein WSDL-Dokument) definiert werden.

REST Maturity Model

Das **REST Maturity Model** (*REST-Reifegradmodell*), kurz **RMM**, ist ein von Leonard Richardson entwickelter Maßstab, der angibt wie stark ein Service auf REST basiert.

REST Maturity Model[2]

Level	Eigenschaften
0	▪ verwendet XML-RPC oder SOAP ▪ der Service wird über eine einzelne URI adressiert ▪ verwendet eine einzelne HTTP-Methode (POST)
1	▪ verwendet verschiedene URIs und Ressourcen ▪ verwendet eine einzelne HTTP-Methode (POST)
2	▪ verwendet verschiedene URIs und Ressourcen ▪ verwendet mehrere HTTP-Verben
3	▪ basiert auf HATEOAS; verwendet Hypermedia für Navigation ▪ verwendet verschiedene URIs und Ressourcen ▪ verwendet mehrere HTTP-Verben

Abgrenzung zu anderen Kommunikationsmechanismen

Bei REST handelt es sich um ein Entwurfsmuster, welches mit verschiedenen Mechanismen implementiert werden kann. Eine Neuerung ist dabei die Verwendung von möglichst vielen HTTP-Verben in Verbindung mit der auszuführenden Aktion. Im Gegensatz dazu wurde zum Beispiel SOAP im Wesentlichen mit den Verben GET und PUT verwendet. Der Unterschied liegt also mehr in der breiteren Verwendung von HTTP als Protokoll und URI als Identifizierungsmechanismus für konkrete Objekte. In der Vergangenheit wurde im Gegensatz dazu mit SOAP eher ein RPC-basierter Aufbau der Schnittstellen durchgeführt. Durch die starke Vorgabe des Entwurfsmusters bei REST sind solche Dienste per Definition strukturierter gebaut und ermöglicht die Verwendung von Clean URLs.

Siehe auch

- Architektur (Informatik)
- Java API for RESTful Web Services (JAX-RS)
- Webservice – „klassische“ SOAP-basierte Webservices
- Web Application Description Language – Beschreibungssprache für REST-basierte Dienste
- XML-RPC - der Vorläufer von SOAP
- OData

Literatur

- Leonard Richardson, Sam Ruby: *Web Services mit REST*. O’Reilly Verlag, 1. Oktober 2007, ISBN 978-3897217270.
- Stefan Tilkov: *REST und HTTP*. Einsatz der Architektur des Web für Integrationsszenarien. dpunkt Verlag, 27. Juli 2009, ISBN 978-3898645836.
- Jamie Kurtz: *ASP.NET MVC 4 and the Web API: Building a REST Service from Start to Finish*. Apress, 30. Januar 2013, ISBN 978-1430249771.

Weblinks

- Roy Fielding: *Architectural Styles and the Design of Network-based Software Architectures.* (http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm) Abgerufen am 6. April 2013 (englisch, Dissertation von Roy Fielding, in der REST beschrieben wird).
- Roy T. Fielding: *REST APIs must be hypertext-driven.* (http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven) 20. September 2008, abgerufen am 7. April 2013 (englisch, Empfehlungen zum Entwerfen von REST-Schnittstellen mit HATEOAS).
- David Megginson: *The quick pitch.* (http://quoderat.megginson.com/2007/02/15/rest-the-quick-pitch) In: *Quoderat*. 15. Februar 2007, abgerufen am 6. April 2013 (englisch, Pragmatische Empfehlungen für die Anwendung von REST).
- Thomas Bayer: *REST Web Services.* (http://www.oio.de/public/xml/rest-webservices.htm) Orientation in Objects (http://www.oio.de/), November 2002, abgerufen am 6. April 2013 (Einführung in RESTful Web Services).
- Alex Rodriquez: *RESTful Web services: The basics.* (http://www.ibm.com/developerworks/webservices/library/ws-restful/) In: *developerWorks.* IBM, 6. November 2008, abgerufen am 6. April 2013 (englisch, Basisprinzipien von REST).
- Stefan Tilkov: *REST – Der bessere Web Service?* (http://it-republik.de/jaxenter/artikel/REST---Der-bessere-Web-Service-2158.html) In: *jaxenter.* IT-Republik, Februar 2009, abgerufen am 6. April 2013 (deutsch, Grundlagen der REST-Architektur).
- Gregor Roth: *RESTful HTTP in practice.* (http://www.infoq.com/articles/designing-restful-http-apps-roth) InfoQ, 18. August 2009, abgerufen am 6. April 2013 (REST in der Praxis).
- Mathieu Fenniak: *The Web API Checklist — 43 Things To Think About When Designing, Testing, and Releasing your API.* (http://mathieu.fenniak.net/the-api-checklist/) 15. April 2013, abgerufen am 16. April 2013 (englisch, Checkliste für die Entwicklung von REST-APIs).

Java

- *JSR 311.* (http://jsr311.java.net/) In: *java.net (http://java.net/).* Oracle, abgerufen am 6. April 2013 (englisch, JAX-RS: Java API für RESTful Web Services).
- *Jersey.* (http://jersey.java.net/) In: *java.net (http://java.net/).* Oracle, abgerufen am 6. April 2013 (englisch, REST-Framework, JAX-RS Referenz-Implementierung).
- *Restlet Framework.* (http://www.restlet.org/) Restlet, abgerufen am 6. April 2013 (englisch, REST-Framework).
- *RESTEasy.* (http://jboss.org/resteasy/) Jboss, abgerufen am 6. April 2013 (englisch, REST-Framework, JAX-RS Implementierung).
- *restfuse: An open-source JUnit extension to test HTTP/REST APIs.* (http://restfuse.com) In: *EclipseSource Developer.* Abgerufen am 6. April 2013 (Framework zum Testen von REST Services).
- *REST-assurred.* (http://code.google.com/p/rest-assured/) In: *Google Code.* Abgerufen am 6. April 2013 (Framework zum Testen von REST Services).

.NET

- *Web API.* (http://www.asp.net/web-api) In: *asp.net (http://asp.net/).* Microsoft, abgerufen am 6. April 2013 (englisch, ASP.NET MVC API für .NET-basierte REST-Services).
- Jamie Kurtz: *ASP.NET Web API REST Security Basics.* (http://jamiekurtz.com/2013/01/14/asp-net-web-api-security-basics/) 14. Januar 2013, abgerufen am 7. April 2013 (englisch).

Einzelnachweise

1. Fielding, et al.: *9 Method Definitions.* (http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html) In: *Hypertext Transfer Protocol -- HTTP/1.1*. 2004, abgerufen am 1. Juli 2010 (englisch).
2. Martin Fowler: *Richardson Maturity Model.* (http://martinfowler.com/articles/richardsonMaturityModel.html) 18. März 2010, abgerufen am 7. April 2013 (englisch, Erklärung des REST Maturity Models (RMM)).

Normdaten (Sachbegriff): GND: 7592728-7

Von „http://de.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=121965884“

Kategorien: Softwarearchitektur | Webservice

- Diese Seite wurde zuletzt am 27. August 2013 um 17:30 Uhr geändert.
- Abrufstatistik

Der Text ist unter der Lizenz „Creative Commons Attribution/Share Alike“ verfügbar; Informationen zum Lizenzstatus eingebundener Mediendateien (etwa Bilder oder Videos) können im Regelfall durch Anklicken dieser abgerufen werden. Möglicherweise unterliegen die Inhalte jeweils zusätzlichen Bedingungen. Durch die Nutzung dieser Website erklären Sie sich mit den Nutzungsbedingungen und der Datenschutzrichtlinie einverstanden.
Wikipedia® ist eine eingetragene Marke der Wikimedia Foundation Inc.