Representational State Transfer (REST)

Programmierparadigma für Webanwendungen

Oliver Schmitt - Python User Group Göttingen

Grundlagen

- REST-Idee: Dissertation von Roy Fielding (2000).
- URL ist das Ergebnis einer serverseitigen Aktion.
- Eigenschaften:
  - Adressierbarkeit
  - Uniform Interface
  - Unterschiedliche Repräsentation
  - Zustandslosigkeit
  - Operationen
  - Hypermedia as the Engine of Application State (HATEOAS)
  - Code on demand (optional): Server schickt Client Funktionalität

Grundlagen

- RESTful Services „leichte“ Webservice
  - Keine klassische Normierung - Vorstellung was REST ist.
  - kein WSDL zur Schnittstellenbeschreibung
  - kein SOAP zur Nachrichtenübertragung
- HTTP Protokoll für Schnittstelle (GET, POST, …) und Nachrichten
- Details (u.a. Datentypen) müssen extern zwischen Service-Provider und -Consumer kommuniziert werden.
- REST macht keine weiteren Vorgaben
- REST ist datenorientierte Aufteilung von Services
- Klassische WS ist funktionale Aufteilung von Services

Operationen (HTTP verbs)

- GET - ordert die angegebene Ressource vom Server an.
- POST - fügt eine neue (Sub-)Ressource unterhalb der angegebenen Ressource ein
- PUT - die angegebene Ressource wird angelegt oder dazu geändert.
- PATCH - ein Teil der angegeben Ressource wird geändert, Seiteneffekte erlaubt.
- DELETE - löscht die angegebene Ressource.
- HEAD - fordert Metadaten zu einer Ressource vom Server an.
- OPTIONS - prüft, welche Methoden auf einer Ressource zur Verfügung stehen.

Request-Metadata(HTTP-Header)

Client	Server
Accept	Etag
User-Agent	Content-Type
If-Modified-Since	Last-Modified

Server Response Status

- 200 OK
- 201 Created
- 202 Accepted
- 301 Moved Permanently
- 304 Not Modified
- 400 Bad Request
- 401 Not Authorized
- 409 Conflict

Data

Customer	
Id	Name
123456	Charles Brown
345663	Martin Miller

Request

```
http://company.com/customers/123456
```

Ressourcen Sammlung

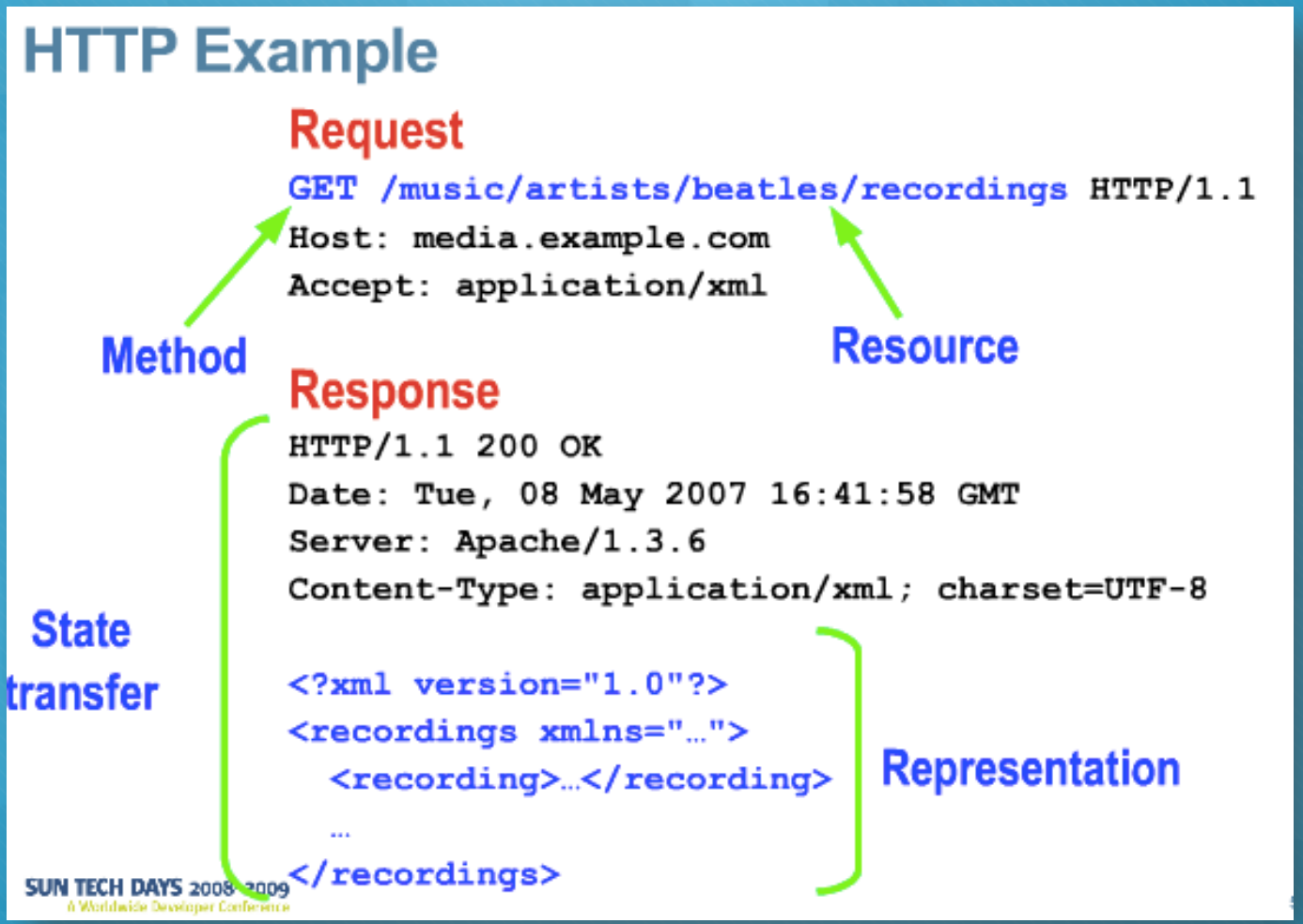
Primärschlüssel

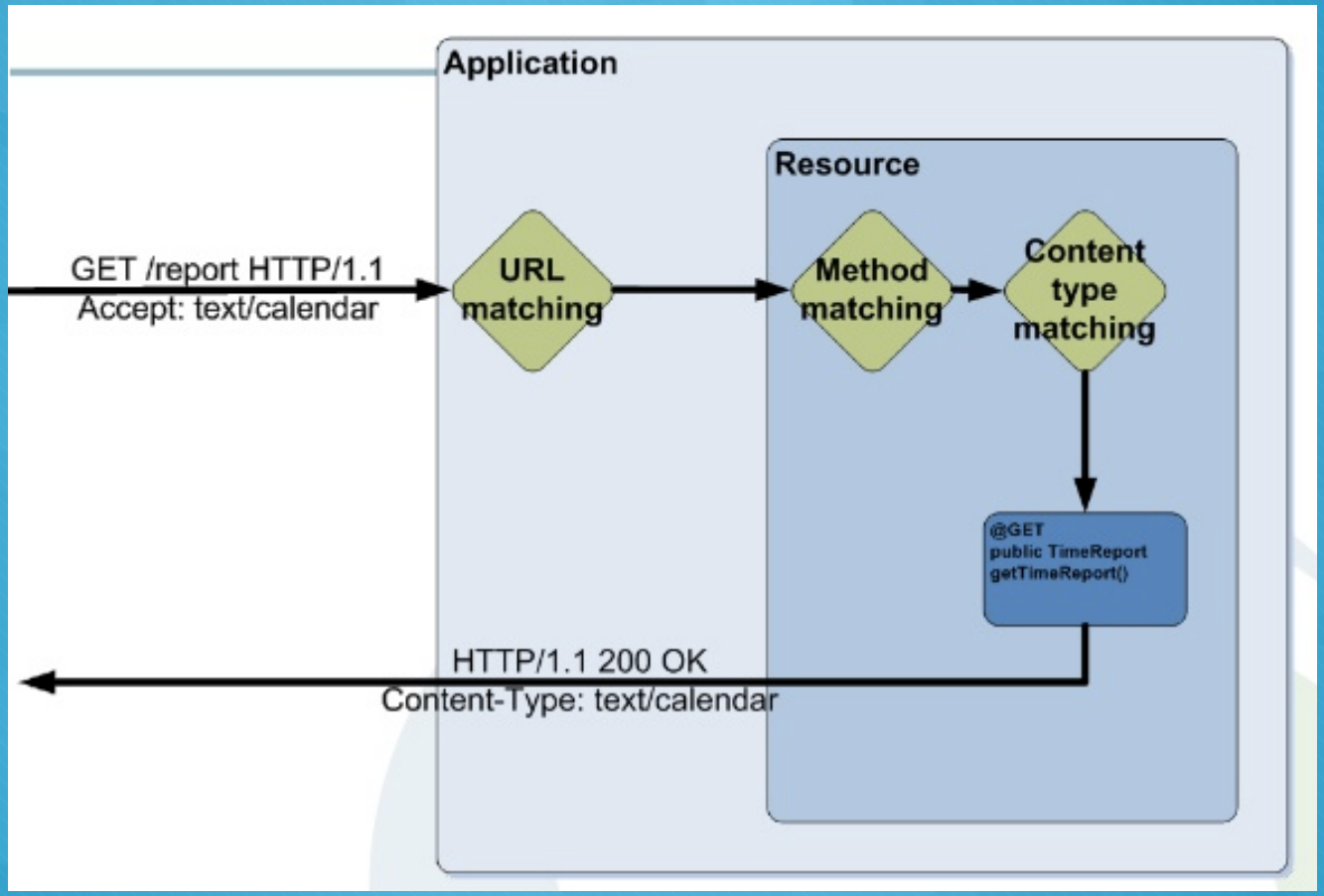
Response

```
{
  "id": "123456",
  "name": "Charles Brown"
}
```

```
...
<id>123456</id>
<name>Charles Brown</name>
...
```

HTTP Example




Application
Resource
GET /report HTTP/1.1
Accept: text/calendar
URL matching
Method matching
Content type matching
@GET
public TimeReport
getTimeReport()
HTTP/1.1 200 OK
Content-Type: text/calendar

Annotation mit Bottle

```
from bottle import route, run, template

@route('/hello/:name')
def index(name='World'):
    return template('<b>Hello {{name}}</b>!', name=name)

@route('/events/:id', method='GET')
def get_event(id):
    return dict(name = 'Event ' + str(id))
run(host='localhost', port=8082)
```