TP
RESTful und SOAP Webservices

5. Klasse TFO Brixen

Michael Mutschlechner

Einführung

- Zum Ansprechen entfernter Ressourcen und entfernter Methoden gibt es mehrere Ansätze.
- Ein einfaches Verfahren ist die Anfrage mit Parametern an einen Web-Server, der den Inhalt zurückliefert.
- Gilt es, auf einer Client-Server-Architektur auf einem Server Operationen auszuführen, so gibt es unterschiedliche Standards wie RMI, CORBA, DCOM, RPC, …
- Diese sind nicht immer optimal
  - Es braucht einen offenen Port
    - HTML wäre optimal, da das Protokoll HTTP verbreitet ist und der Port zum HTTP-Server in der Regel frei bzw. über einen Proxy unproblematisch ist.
  - Lösungen sollten plattformübergreifend und programmiersprachenunabhängig sein, damit Client und Server auf beliebigen Betriebssystemen und in beliebigen Programmiersprachen entwickelt werden können.
    - Es ist naheliegend, ein neutrales Text-Protokoll einzusetzen, um keine Bindung an Rechnersysteme und Programmiersprachen zu erzwingen.
- Bietet ein Web-Server Dienste für Clients an, nennen wir das Web-Service, wobei wir erst einmal offen lassen, wie genau die Kommunikation zwischen Client und Server aussieht.

REST und SOAP

- REST und SOAP sind die zwei bekanntesten Web-Service-Standards
  - SOAP ist ein standardisiertes Protokoll, bei dem XML-Nachrichten übertragen werden.
    - Ist ähnlich wie RMI eine Technologie zum entfernten Methodenaufruf, bei der Argumente übergeben und eine Rückgabe eingesammelt wird.
    - Die Parameter und Rückgaben sind exakt in einer Datei beschrieben (WSDL-Datei, ebenfalls im XML-Format)
    - Es lassen sich Generatoren einsetzen, die mithilfe dieser WSDL-Datei Zugriffsklassen in allen möglichen Programmiersprachen generieren.
  - Beim REST-Prinzip wird eine Anfrage über HTTP an den Web-Server gestellt.
    - Die URL kodiert die Ressource und nur einige wenige Operationen (wie Lesen und Aktualisieren) sind möglich.
    - Im Mittelpunkt steht eine Ressource, die eindeutig adressierbar ist.
    - Die Ressource hat eine Repräsentation, die in jedem Format sein kann, also XML, Text, Bilder oder .mp3-Dateien.

REST oder SOAP?

- Beide Verfahren haben ihre Vorzüge
  - eine Variante ist nicht immer besser als die andere
  - Wenn es Ressourcen-Zugriffe gibt und es keine entfernten Aufrufe an wirkliche Objekte mit Zuständen und Verhalten gibt, ist ein RESTful Web-Service eine gute Wahl.
  - Stehen entfernte Objekte mit ihren vielfältigen Funktionen im Vordergrund, ist ein SOAP Web-Service in Ordnung.

- RESTful Web-Services

Rest

- Konzept: eine Ressource ist über einen Web-Server verfügbar und wird eindeutig über eine URI (Uniform Resource Identifier) identifiziert.
- Rest steht für: Representational State Transfer
- Der Begriff wurde im Jahr 2000 von Roy Fielding geprägt.
- Es wird das altbekannte HTTP-Protokoll verwendet.
- Da unterschieden werden muss, ob eine Ressource neu angelegt, gelesen, aktualisiert, gelöscht oder aufgelistet werden soll, werden dafür die bekannten HTTP-Methoden verwendet.

HTTP-Methode	Operation
GET	Listet Ressourcen auf oder holt eine konkrete Ressource.
PUT	Aktualisiert eine Ressource.
DELETE	Löscht eine Ressource oder eine Sammlung von Ressourcen.
POST	Semantik kann variieren, in der Regel aber geht es um das Anlegen einer neuen Ressource.

Ressourcen

- Auf den ersten Blick besteht das WWW nur aus den Komponenten, die ein menschlicher Benutzer auch wahrnehmen kann – die Webseiten und -applikationen, die beispielsweise über den Browser aufgerufen werden können.
- Es gibt aber auch „maschinelle“ Benutzer, beispielsweise andere Anwendungen, die auf Daten aus dem Internet zugreifen.
- Ein Betreiber kann somit einen bestimmten Dienst im Internet anderen Anwendungen zur Verfügung stellen – diese Dienste werden Webservices genannt.
- Ein ganz einfaches Beispiel ist etwa Twitter
  - über einen Webservice können beliebige Anwendungen (vorherige Autorisierung vorausgesetzt) im Namen eines Benutzers Tweets auslesen oder schreiben.

Ressourcen

- Im Beispiel Twitter könnte jeder Benutzer und sogar jeder einzelne Tweet als eigene Ressource betrachtet werden. Diese Ressourcen sollten folgende Anforderungen erfüllen:
  - Adressierbarkeit
    - Jede Ressource muss über eine eindeutige URI identifiziert werden können.
    - Ein Kunde mit der Kundennummer 123456 könnte also zum Beispiel über die URI http://ws.mydomain.tld/customers/123456 adressiert werden.
  - Zustandslosigkeit
    - Die Kommunikation der Teilnehmer untereinander ist zustandslos.
    - Es existieren keine Benutzersitzungen (etwa in Form von Sessions und Cookies), sondern bei jeder Anfrage werden alle notwendigen Informationen wieder neu mitgeschickt.
    - Durch die Zustandslosigkeit sind REST-Services sehr einfach skalierbar;
      - da keine Sitzungen existieren, ist es im Grunde egal, wenn mehrere Anfragen eines Clients auf verschiedene Server verteilt werden.
  - Einheitliche Schnittstelle
    - Jede Ressource muss über einen einheitlichen Satz von Standardmethoden zugegriffen werden können. Beispiele für solche Methoden sind die Standard-HTTP-Methoden GET, POST, …
  - Entkopplung von Ressourcen und Repräsentation
    - Es können verschiedene Repräsentationen einer Ressource existieren.
    - Ein Client kann somit etwa eine Ressource explizit beispielsweise im XML- oder JSON-Format anfordern.

Ein kleines Beispiel

- Fiktiver Webservice, der Zugriff auf eine Produkt-Datenbank bietet.
- Einstiegspunkt: http://ws.mydomain.tld/products
- Ein GET-Request an diese URI soll also eine Liste aller Produkte zurückliefern:

Request	Response
GET /products HTTP/1.0 Accept: application/json	HTTP/1.0 200 OK Content-Type: application/json Content-Length: 1234 [{ uri: "http://ws.mydomain.tld/products/1000", name: "Gartenstuhl", price: 24.99 }, { uri: "http://ws.mydomain.tld/products/1001", name: "Sonnenschirm", price: 49.99 }]

- Über den „Accept“-Header kann der Client mitteilen, in welchem Format er gerne eine Antwort erhalten würde.
  - Alternativ könnte man zum Beispiel auch einen Accept: text/xml-Header mitschicken
- Durch einen POST-Request an dieselbe URL könnte dann ein neuer Artikel erstellt werden:

Request	Response
POST /products HTTP/1.0 Content-Type: application/json Content-Length: 38 { name: "Sandkasten", price: 89.99 }	HTTP/1.0 201 Created Location: http://ws.mydomain.tld/products/1002

Rest: Post

- Interessant ist das Format, in dem die Daten zum Server geschickt werden.
  - Schickt man ein Formular im Browser ab (method="get"), so kodiert dieser die gesendeten Daten in URL-Codierung
    - &name=Sandkasten&price=89.99
  - In einer REST-Architektur wäre dies genauso möglich, allerdings müsste der Client dann auch einen „Content-Type: application/x-www-form-urlencoded“-Header mitschicken.
- Außerdem auffällig ist hier, dass der Server in diesem Fall nicht mit dem üblichen 200 OK-Status antwortet, sondern mit 201 Created.
- Außerdem enthält die Antwort einen Header, der die URI der neu erstellten Ressource enthält.

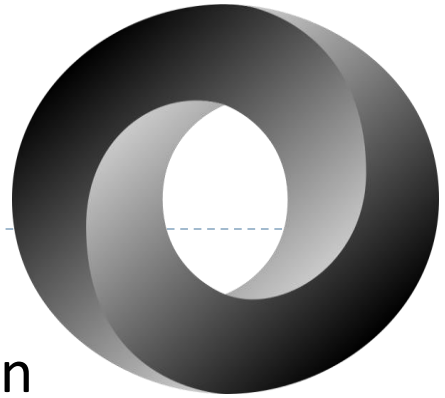
Rest: Sicherheit

- Zugriffssicherheit lässt sich am einfachsten über die ganz normale HTTP-Authentifizierung erreichen.
  - Der Client schickt bei jeder Anfrage einen Authentication-Header mit, in welchem ein Benutzername und ein Passwort codiert sind.

- Damit keiner die Kommunikation mitlesen kann, kann die komplette HTTP-Kommunikation auf Transportebene per SSL verschlüsselt werden.

- EINSCHUB: JSON UND XML

JSON



- JavaScript Object Notation
- kompaktes Datenformat in einer einfach lesbaren Textform zum Zweck des Datenaustauschs zwischen Anwendungen
- JSON ist unabhängig von der Programmiersprache.
  - Parser existieren in praktisch allen verbreiteten Sprachen.
- JSON wurde ursprünglich von Douglas Crockford spezifiziert. Aktuell wird es durch zwei konkurrierende Standards spezifiziert, RFC 7159 von Douglas Crockford und ECMA-404.
- Insbesondere bei Webanwendungen und mobilen Apps wird es in Verbindung mit JavaScript, Ajax oder WebSockets zum Transfer von Daten zwischen dem Client und dem Server häufig genutzt.

JSON: Datenstruktur

- Daten können beliebig verschachtelt werden
  - beispielsweise ist ein Array von Objekten möglich
- Als Zeichenkodierung benutzt JSON standardmäßig UTF-8

JSON: Datentypen

- Nullwert
  - wird durch das Schlüsselwort null dargestellt.
- Boolescher Wert
  - true oder false dargestellt
  - Sind keine Zeichenketten, werden daher, wie null, nicht in Anführungszeichen gesetzt
- Zahl
  - Folge der Ziffern 0–9
  - kann durch ein negatives Vorzeichen - eingeleitet und einen Dezimalpunkt . unterbrochen sein.
  - Die Zahl kann durch die Angabe eines Exponenten e oder E ergänzt werden, dem ein Vorzeichen + oder - und eine Folge der Ziffern 0–9 folgt
- Zeichenkette
  - beginnt und endet mit doppelten geraden Anführungszeichen ("). Sie kann Unicode-Zeichen und Escape-Sequenzen enthalten.
- Array
  - beginnt mit [und endet mit]
  - enthält eine durch Kommata geteilte, geordnete Liste von Werten gleichen oder verschiedenen Typs
  - leere Arrays sind zulässig
- Objekt
  - beginnt mit { und endet mit }.
  - enthält eine durch Kommata geteilte, ungeordnete Liste von Eigenschaften.
  - Objekte ohne Eigenschaften („leere Objekte“) sind zulässig
  - Eigenschaft
    - besteht aus einem Schlüssel und einem Wert, getrennt durch einen Doppelpunkt (Schlüssel:Wert)
    - Jeder Schlüssel darf in einem Objekt nur einmal enthalten sein
    - der Schlüssel ist eine Zeichenkette
    - der Wert ist ein Objekt, ein Array, eine Zeichenkette, eine Zahl oder einer der Ausdrücke true, false oder null.

JSON: Beispiel


```
{
  "Herausgeber": "Xema",
  "Nummer": "1234-5678-9012-3456",
  "Deckung": 2e+6,
  "Waehrung": "EURO",
  "Inhaber":
  {
    "Name": "Mustermann",
    "Vorname": "Max",
    "maennlich": true,
    "Hobbys": [ "Reiten", "Golfen", "Lesen" ],
    "Alter": 42,
    "Kinder": [],
    "Partner": null
  }
}
```


JSON: Beispiel: Facebook


```
{
  "data": [
    {
      "id": "X999_Y999",
      "from": {
        "name": "Tom Brady", "id": "X12"
      },
      "message": "Looking forward to 2010!",
      "actions": [
        {
          "name": "Comment",
          "link": "http://www.facebook.com/X999/posts/Y999"
        },
        {
          "name": "Like",
          "link": "http://www.facebook.com/X999/posts/Y999"
        }
      ],
      "type": "status",
      "created_time": "2010-08-02T21:27:44+0000",
      "updated_time": "2010-08-02T21:27:44+0000"
    },
    {
      "id": "X998_Y998",
      "from": {
        "name": "Peyton Manning", "id": "X18"
      },
      "message": "Where's my contract?",
      "actions": [
        {
          "name": "Comment",
          "link": "http://www.facebook.com/X998/posts/Y998"
        },
        {
          "name": "Like",
          "link": "http://www.facebook.com/X998/posts/Y998"
        }
      ],
      "type": "status",
      "created_time": "2010-08-02T21:27:44+0000",
      "updated_time": "2010-08-02T21:27:44+0000"
    }
  ]
}
```


XML

- eXtensible Markup Language
  (erweiterbare Auszeichnungssprache)
- 1998 in Version 1.0 von der W3C als Standard verabschiedet
- Grundidee:
  - Trennung von Inhalt und Struktur

- inhaltliche Bausteine:
  - Elemente
  - Attribute
- formale Bausteine:
  - syntaktische Festlegung auf die Notation
    der inhaltlichen Bausteine
- genau ein Wurzelelement
- Hierarchischer Aufbau (Baum)

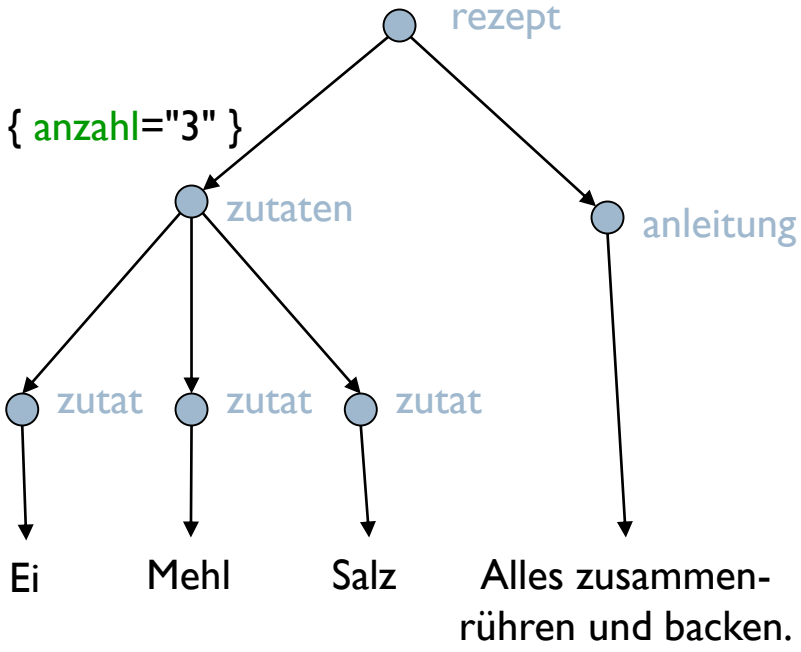
XML ist

- einfach
  - zu lesen durch Menschen
  - zu verarbeiten durch Maschinen
  - zu generieren
- erweiterbar
  - XML ist nur generische Syntax
  - Zusatz-Standards nutzen diese Syntax
- standardisiert
  - weithin akzeptiertes Format
  - Standard-Tools als Basiskomponenten für Applikationen

XML: Beispiel

```
<?xml version="1.0"?>

<rezept>
  <zutaten anzahl="3">
    <zutat>Ei</zutat>
    <zutat>Mehl</zutat>
   <zutat>Salz</zutat>
  </zutaten>
  <anleitung>
    Alles zusammen-
    rühren und backen.
  </anleitung>
</rezept>
```



Textuelle Darstellung

Graphische Darstellung

XML: Begriffe

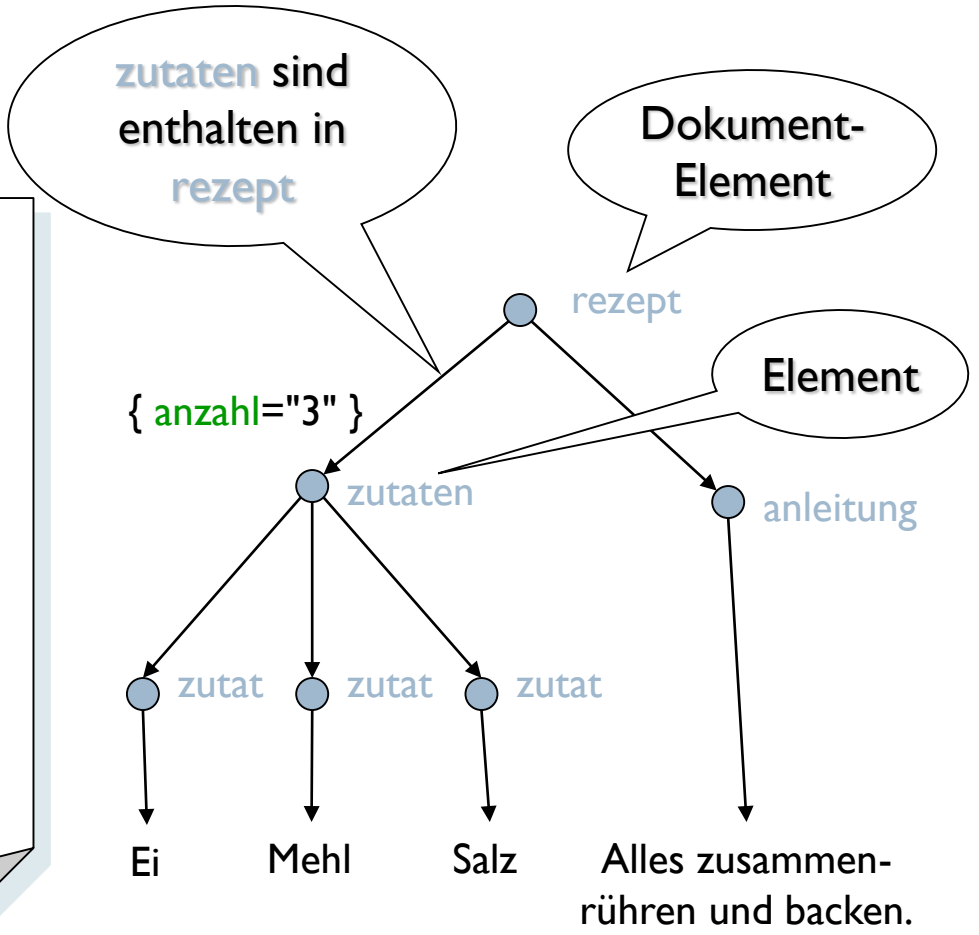
- Element
  - Wichtigste Struktureinheit
  - Der Name eines XML-Elements kann weitgehend frei gewählt werden
  - Elemente können weitere Elemente, Text- und andere Knoten – ggfs. auch vermischt – enthalten.
  - sind die Träger der Information in einem XML-Dokument, unabhängig davon, ob es sich um Text, Bilder usw. handelt
- Wohlgeformtheit
  - Ein XML-Dokument ist wohlgeformt, wenn es der XML-Syntax folgt
    - Das Dokument besitzt genau ein Wurzelelement. Als Wurzelelement wird dabei das jeweils äußerste Element bezeichnet, z. B. <html> in XHTML.
    - Alle Elemente mit Inhalt besitzen einen Beginn- und einen End-Auszeichner (-Tag) (z. B. <eintrag>Eintrag 1</eintrag>). Elemente ohne Inhalt können auch in sich geschlossen sein, wenn sie aus nur einem Auszeichner bestehen, der mit /> abschließt (z. B. <eintrag />).
    - Die Beginn- und End-Auszeichner sind ebenentreu-paarig verschachtelt. Das bedeutet, dass alle Elemente geschlossen werden müssen, bevor die End-Auszeichner des entsprechenden Elternelements oder die Beginn-Auszeichner eines Geschwisterelements erscheinen.
    - Ein Element darf nicht mehrere Attribute mit demselben Namen besitzen.
    - Attributeigenschaften müssen in Anführungszeichen stehen.
    - Die Beginn- und End-Auszeichner beachten die Groß- und Kleinschreibung (z. B. <eintrag></Eintrag> ist nicht gültig).
- Gültigkeit
  - Soll XML für den Datenaustausch verwendet werden, ist es von Vorteil, wenn das Format mittels einer Grammatik (z. B. einer Dokumenttypdefinition oder eines XML-Schemas) definiert ist. Der Standard definiert ein XML-Dokument als gültig (oder englisch valid), wenn es wohlgeformt ist, den Verweis auf eine Grammatik enthält und das durch die Grammatik beschriebene Format einhält.

XML: Elemente

- Elemente haben einen Namen.
- Namen sollten 'sinnvoll' sein, um die spätere Interpretation durch den Menschen zu gewährleisten.
- Elemente haben öffnenden und schließenden Tag: <rezept> ... </rezept>
- Elemente sind hierarchisch verschachtelt: <rezept><zutaten> ... </zutaten></rezept>
- Spezielles Element: Dokument-Element

```
<?xml version="1.0"?>

<rezept>
  <zutaten anzahl="3">
    <zutat>Ei</zutat>
    <zutat>Mehl</zutat>
   <zutat>Salz</zutat>
  </zutaten>
  <anleitung>
    Alles zusammen-
    rühren und backen.
  </anleitung>
</rezept>
```



XML: Kommentare

- Kommentare beginnen mit der Zeichenfolge <!-- und enden mit -->. Dazwischen dürfen sich beliebige Zeichen befinden, allerdings keine aufeinanderfolgenden Bindestriche.
  - <!-- Dies ist ein Kommentar --> <!-- Kommentare dürfen selbstverständlich über mehrere Zeilen gehen -->

XML: Beispiel

```
<?xml version="1.0"?>
<PurchaseOrder PurchaseOrderNumber="99503" OrderDate="1999-10-20">
  <Address Type="Shipping">
    <Name>Ellen Adams</Name>
    <Street>123 Maple Street</Street>
    <City>Mill Valley</City>
    <State>CA</State>
    <Zip>10999</Zip>
    <Country>USA</Country>
  </Address>
  <Address Type="Billing">
    <Name>Tai Yee</Name>
    <Street>8 Oak Avenue</Street>
    <City>Old Town</City>
    <State>PA</State>
    <Zip>95819</Zip>
    <Country>USA</Country>
  </Address>
  <DeliveryNotes>Please leave packages in shed by driveway.</DeliveryNotes>
  <Items>
    <Item PartNumber="872-AA">
      <ProductName>Lawnmower</ProductName>
      <Quantity>1</Quantity>
      <USPrice>148.95</USPrice>
      <Comment>Confirm this is electric</Comment>
    </Item>
    <Item PartNumber="926-AA">
      <ProductName>Baby Monitor</ProductName>
      <Quantity>2</Quantity>
      <USPrice>39.98</USPrice>
      <ShipDate>1999-05-21</ShipDate>
    </Item>
  </Items>
</PurchaseOrder>
```

- EINSCHUB ENDE

Rest in der Praxis: JAX-RS

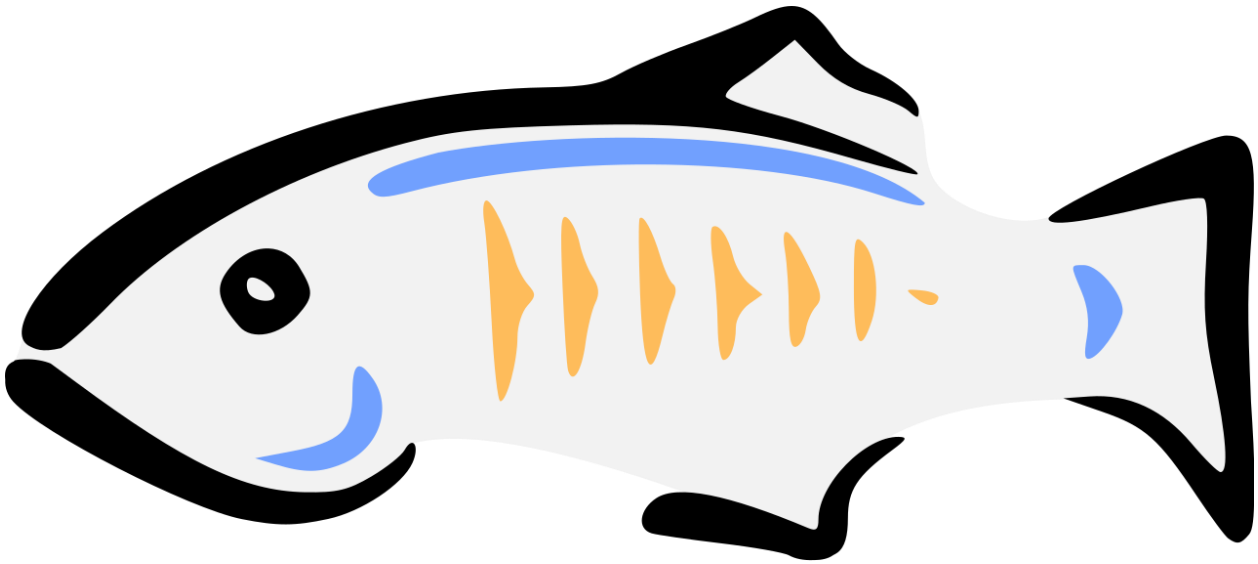
- Für Java gibt es mit JAX-RS einen Standard zum Deklarieren von REST-basierten Web-Services.
- JAX-RS = Java API for RESTful Web Services
  - **Spezifikation** einer Programmierschnittstelle (API) der Programmiersprache Java, die die Verwendung von REST im Rahmen von Webservices ermöglicht und vereinheitlicht.
  - JAX-RS benutzt Annotationen
- Jeder Applikationsserver ab Java EE 6 (Enterprise Edition) enthält standardmäßig eine JAX-RS-Implementierung.
- Da wir JAX-RS ohne einen Applikationsserver ausprobieren möchten, ist eine JAX-RS-Implementierung nötig.
  - Es ist naheliegend, die JAX-RS-Referenzimplementierung Jersey (http://jersey.dev.java.net/) zu nutzen, die auch intern von Applikationsservern verwendet wird.
    - Mit Jersey lässt sich entweder ein Servlet-Endpunkt definieren, sodass der RESTful Web-Service in einem Servlet-Container wie Tomcat läuft, oder der zusammen ab Java SE 6 eingebaute Mini-HTTP-Server nutzen.

Jersey

- Jersey RESTful Web Services framework
  - Ist ein Open-Source Framework, um RESTful Web Services in Java zu programmieren.

Variante 1: Glassfish

- GlassFish ist ein Open-Source-Anwendungsserver-Projekt für Java EE, das von Sun Microsystems gestartet wurde und seit 2010 von der Oracle Corporation gesponsert wird. GlassFish ist freie Software.



Rest mit Jersey und Glassfish: ein Beispiel

```
public class CinemaEventHandler {
        public CinemaEventHandler() {
                super();
        }

        public String postMovieEvent(String str) {
                System.out.println("received event:" + str);
                return "event received " + str;
        }

        public String getMovieEvent(Request request) {
                return "nothing to report from getMovieEvent";
        }
}
```

Annotationen

```
@Path("movieevent")
public class CinemaEventHandler {
        public CinemaEventHandler() {
                super();
        }

        @POST
        @Consumes("application/json")
        @Produces("text/plain")
        public String postMovieEvent(@Context Request request, String str) {
                System.out.println("received event:" + str);
                return "event received " + str;
        }

        @GET
        @Produces("text/plain")
                public String getMovieEvent(@Context Request request) {
                return "nothing to report from getMovieEvent";
        }
}
```

Startklasse

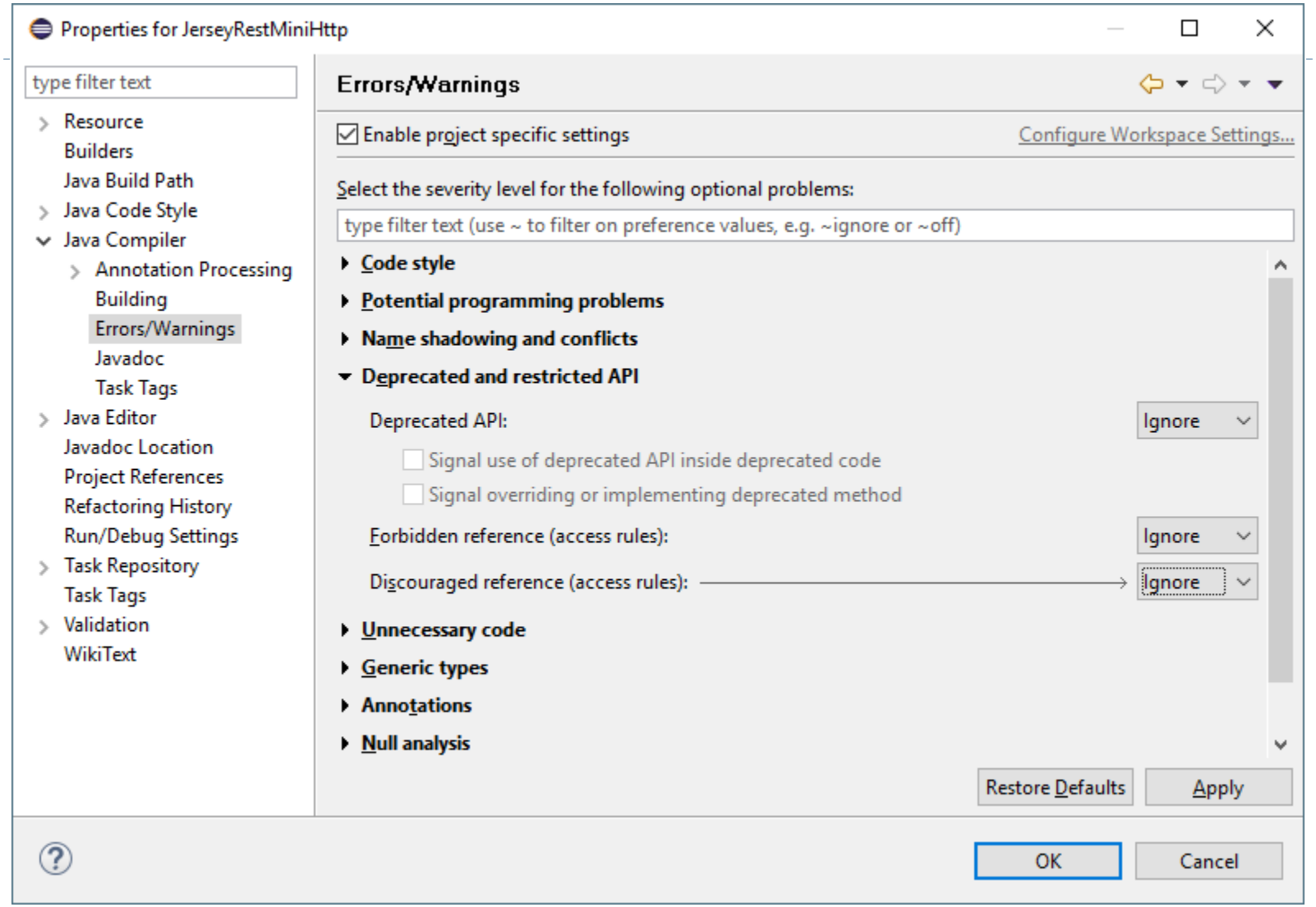
```
import com.sun.net.httpserver.HttpServer;
import java.net.URI;
import javax.ws.rs.core.UriBuilder;
import org.glassfish.jersey.jdkhttp.JdkHttpServerFactory;
import org.glassfish.jersey.server.ResourceConfig;

public class CinemaEventHandlerRestStartup {
    private final static int port = 9998;
    private final static String host="http://localhost/";

    public static void main(String[] args) {
          URI baseUri = UriBuilder.fromUri(host).port(port).build();
          ResourceConfig config = new ResourceConfig(CinemaEventHandler.class);
          HttpServer server = JdkHttpServerFactory.createHttpServer(baseUri, config);
    }
}
```

Fallstricke

- Es müssen die Jersey JAR-Dateien in den Buildpath aufgenommen werden.
  - https://jersey.java.net/download.html
- Es muss die JAR-Datei für den Webserver in den Buildpath aufgenommen werden.
  - jersey-container-jdk-http-2.26-b03
  - https://mvnrepository.com/artifact/org.glassfish.jersey.containers/jersey-container-jdk-http
- Fehlermeldung:
  - Access restriction: The type 'HttpServer' is not API (restriction on required library 'B:\Program Files\Java\jdk1.8.0_121\jre\lib\rt.jar')
  - Der HttpServer funktioniert nur mit der Oracle Java JDK, nicht mit anderen Java Implementierungen wie z.B. OpenJDK
  - Daher muss dies im Projekt bestätigt werden / Fehlermeldung ausschalten:
    - Nächste Folie


Properties for JerseyRestMiniHttp
type filter text
Resource
Builders
Java Build Path
Java Code Style
Java Compiler
Annotation Processing
Building
Errors/Warnings
Javadoc
Task Tags
Java Editor
Javadoc Location
Project References
Refactoring History
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Errors/Warnings
Enable project specific settings
Configure Workspace Settings...
Select the severity level for the following optional problems:
type filter text (use ~ to filter on preference values, e.g. ~ignore or ~off)
Code style
Potential programming problems
Name shadowing and conflicts
Deprecated and restricted API
Deprecated API:
Ignore
Signal use of deprecated API inside deprecated code
Signal overriding or implementing deprecated method
Forbidden reference (access rules):
Ignore
Discouraged reference (access rules):
Ignore
Unnecessary code
Generic types
Annotations
Null analysis
Restore Defaults
Apply
OK
Cancel

Beispiele

- Nach dem Aufruf im Webserver von http://localhost:9998/movieevent
  - nothing to report from getMovieEvent

- Oder mit einem Rest-Client
  - Standalone/Browser Plugin

Annotations

- JAX-RS definiert einige Annotationen, die zentrale Konfigurationen bei RESTful Web-Services vornehmen, etwa Pfadangaben, Ausgabeformate oder Parameter.
  - @Path: Die Pfadangabe, die auf den Basispfad gesetzt wird.
  - @GET: Die HTTP-Methode. Hier gibt es auch die Annotationen für die anderen HTTP-Methoden POST, PUT, ...
  - @Produces: kann zwar grundsätzlich auch entfallen, aber besser ist es, deutlich einen MIME-Typ zu setzen. Es gibt String-Konstanten für die wichtigsten MIME-Typen, wie MediaType.TEXT_XML oder MediaType.TEXT_HTML, und auch Strings wie »application/pdf« können direkt gesetzt werden.
  - @Consumes
  - …
    - https://jersey.java.net/documentation/latest/jaxrs-resources.html

Rest-Parameter

- Beispiel für Pfadangaben in einem RESTful Service:
  - *http://www.tutego.de/blog/javainsel/*category*/java-7/*page*/2/*
  - Als Schlüssel-Werte-Paar lassen sich festhalten: category=java-7 und page=2.
  - Der Server wird die URL auseinanderpflücken und genau die Blog-Einträge liefern, die zur Kategorie »java-7« gehören und sich auf der zweiten Seite befinden.
- Beispiel:
  - Auf einem Endpunkt */rest/message/* kann es unterschiedliche URLs geben, die Operationen wie »finde alle« oder »finde alle mit der Einschränkung X« abbilden.
    - */rest/message/*: alle Nachrichten aller Nutzer
    - */rest/message/user/chris*: alle Nachrichten von Benutzer »chris«
    - */rest/message/user/chris/search/kitesurfing*: alle Nachrichten von Benutzer »chris« mit dem Betreff »kitesurfing«

Rest-Parameter

- Die JAX-RS-API erlaubt es, dass Parameter leicht eingefangen werden können.
- Im Beispiel entsthen für die drei möglichen URLs entstehen drei überladene Methoden:

```
@GET @Produces( MediaType.TEXT_PLAIN )
public String message() ...

@GET @Produces( MediaType.TEXT_PLAIN )
@Path("user/{user}")
public String message( @PathParam("user") String user )
{
    return String.format( "Benutzer = %s", user );
}

@GET
@Produces( MediaType.TEXT_PLAIN )
@Path("user/{user}/search/{search}")
public String message( @PathParam("user") String user,
@PathParam("search") String search )
{
    return String.format( "Benutzer = %s, Suchstring = %s", user, search );
}
```

Rest-Parameter

- Die bekannte @Path-Annotation enthält nicht einfach nur einen statischen Pfad, sondern beliebig viele Platzhalter in geschweiften Klammern.
- Der Name des Platzhalters taucht in der Methode wieder auf, nämlich dann, wenn er mit @PathParam an einen Parameter gebunden wird.
- Jersey parst für uns die URL und füllt die Parametervariablen passend auf bzw. ruft die richtige Methode auf.

URL-Endung	Aufgerufene Methode
/rest/message/	`message()`
/rest/message/user/chris	`message( String user )`
/rest/message/user/chris/search/kitesurfing	`message( String user, String search )`

Variante 2: Tomcat

- Apache Tomcat ist ein Open-Source-Webserver und Webcontainer, der die Spezifikation für Java Servlets und JavaServer Pages (JSP) implementiert und es damit erlaubt, in Java geschriebene Web-Anwendungen auszuführen.

- Dafür ist die „Eclipse IDE for Java EE Developers” notwendig

Rest mit Jersey und Tomcat: ein Beispiel

- Eclipse EE starten
- Ein Dynamic Web Project erstellen
  - Als Target Runtime Apache Tomcat wählen (eventuell vorher installieren)
  - Im übernächsten Schritt „Generate web.xml deployment descriptor“ wählen
  - Ein Paket erstellen

Rest mit Jersey und Tomcat: ein Beispiel

- Eclipse EE starten
- Ein Dynamic Web Project erstellen
  - Als Target Runtime Apache Tomcat wählen (eventuell vorher installieren)
  - Im übernächsten Schritt „Generate web.xml deployment descriptor“ wählen
  - Ein Paket erstellen
- Die Jersey JAR-Dateien in den den Ordner WEB-INF/lib/ kopieren
  - https://jersey.java.net/download.html
- Neue Klasse mit Annotations erstellen

Rest mit Jersey und Tomcat: ein Beispiel

```
@Path("/hello")
public class Hello {

  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String sayPlainTextHello() {
    return "Hello Jersey";
  }

  @GET
  @Produces(MediaType.TEXT_XML)
  public String sayXMLHello() {
    return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
  }

  @GET
  @Produces(MediaType.TEXT_HTML)
  public String sayHtmlHello() {
    return "<html> " + "<title>" + "Hello Jersey" + "</title>"
        + "<body><h1>" + "Hello Jersey" + "</body></h1>" + "</html> ";
  }
}
```

Rest mit Jersey und Tomcat: ein Beispiel

- Nun muss noch die Datei web.xml angepasst werden:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
  <display-name>JerseyRestTomcat</display-name>
  <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-
class>
    <init-param>
        <param-name>jersey.config.server.provider.packages</param-name>
        <param-value>Hello</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Hier das Paket mit den Klassen angeben

Rest mit Jersey und Tomcat: ein Beispiel

- Der Server wird mit „Run As“-> „Run on Server“ gestartet.
- Wird im Browser http://localhost:8080/JerseyRestTomcat/rest/hello aufgerufen, erscheint:

  Hello Jersey

- HTML-Code wird geliefert.
- curl http://localhost:8080/JerseyRestTomcat/rest/hello liefert nur “Hello Jersey”

- REST CLIENT

Quellen

- http://openbook.rheinwerk-verlag.de/java7/1507_13_001.html#dodtp82d1ec9d-ccf4-456f-8af9-ebd4bb3c87b4