REST Error Handling

cphbusiness

REST Error Handling

cphbusiness

There are basically two things that can make a REST API fail:

1. The client makes a call the REST API is not designed to handle (call a not existing URI, provide a wrong Content-Type etc.)
2. The Backend Business Logic throws an exception (Checked or Unchecked) for example CustomerNotFoundException or a NullPointerException

http://www.hacktrix.com/checked-and-unchecked-exceptions-in-java

REST Error Handling

cphbusiness

Following the REST Architecture Rule "**Resources are decoupled from their representation"** we should in NO WAY (unless in debug mode) try to pass on the Original Exception Object as or with the error message.

The API should provide a useful error message in a known consumable format. The representation of an error should be no different than the representation of any resource, just with its own set of fields.

The API should always return sensible HTTP status codes. API errors typically break down into 2 types:

- **400** series status codes for client issues
- **500** series status codes for server issues.

If JSON is used for normal DTO's use JSON also for Error Messages.

http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

HTTP Status Codes

cphbusiness

Available via the enum:
javax.ws.rs.core.Response.Status

SUCCESSFUL
200, OK
201, Created
202, Accepted
204, No Content
205, Reset Content
206, Partial Content

REDIRECTION
301, Moved Permanently
302, Found
303, See Other
304, Not Modified
305, Use Proxy
307, Temporary Redirect

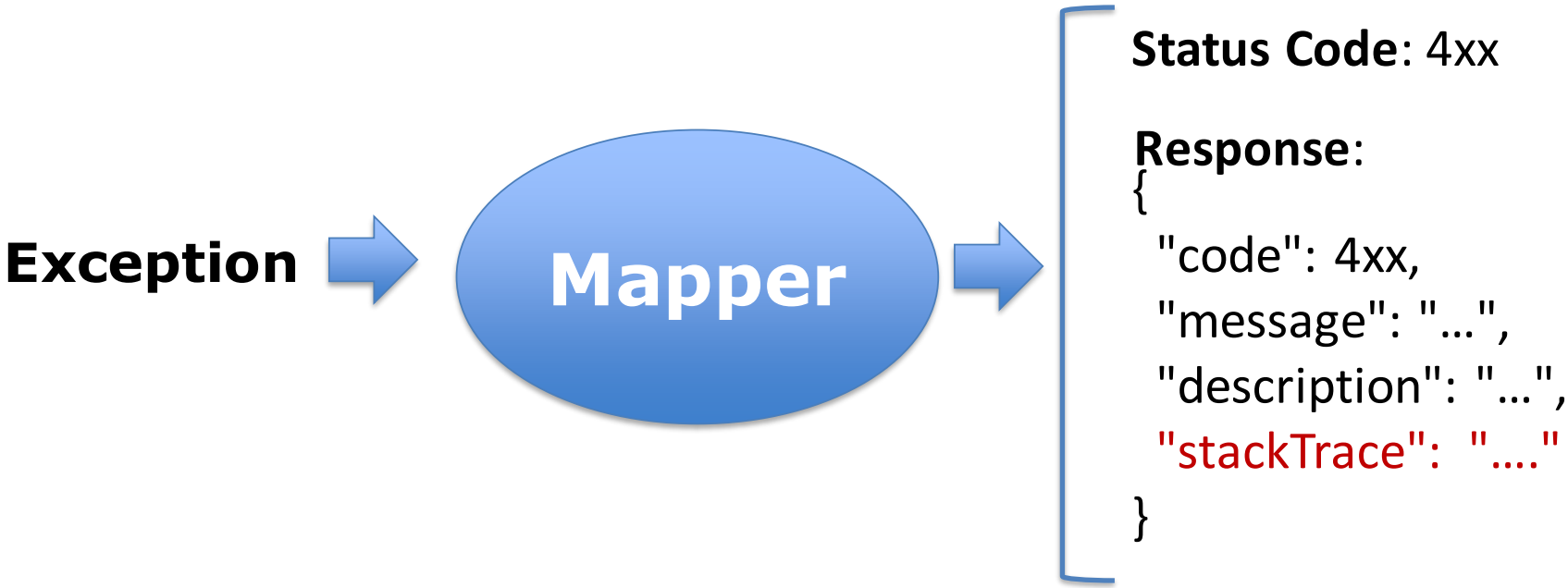
CLIENT_ERROR
400, Bad Request
401, Unauthorized
402, Payment Required
403, Forbidden
404, Not Found
405, Method Not Allowed
406, Not Acceptable
407, Proxy Authentication Required
408, Request Timeout
409, Conflict
410, Gone
411, Length Required
412, Precondition Failed
413, Request Entity Too Large
414, Request-URI Too Long
415, Unsupported Media Type
416, Requested Range Not Satisfiable
417, Expectation Failed

SERVER_ERROR
500, Internal Server Error
501, Not Implemented
502, Bad Gateway
503, Service Unavailable
504, Gateway Timeout
505, HTTP Version Not Supported

REST Error Handling and Exceptions

cphbusiness

Basically REST Error Handling boils down to (when using an OO language) → Take an existing Exception, and Map it into a sufficient HTTP response (JSON or XML).



Important: The StackTrace should **never** be included in production (reveals internal information, irrelevant for users of the API), but is extremely useful during development.

Exception Handling-1

cphbusiness

Errors can be reported to a client either by:

Creating and returning the appropriate Response object:

```
Response.status(Response.Status.LENGTH_REQUIRED).build();
```

Throwing an exception

Application code is allowed to throw any checked (i.e. java.lang.Exception) or unchecked (java.lang.RuntimeException) exceptions.

Thrown exceptions are handled by the JAX-RS runtime if you have registered an exception mapper. Exception mappers can convert an exception to an HTTP response.

If the thrown exception is not handled by a mapper, it is propagated and handled by the container (i.e., servlet) JAX-RS is running within.

https://jersey.java.net/documentation/latest/representations.html

Exception Handling-2

cphbusiness

JAX-RS provides the **WebApplicationException**.
When thrown by application code its automatically processed by JAX-RS without the need for an explicit mapper.

This exception is pre initialized with either a Response or a particular status code

```
@GET
@Path("{id}")
@Produces("application/xml")
public Customer getCustomer(@PathParam("id") int id) {
  Customer cust = findCustomer(id);
  if (cust == null) {
    throw new WebApplicationException(Response.Status.NOT_FOUND);
  }
  return cust;
}
```

https://jersey.java.net/documentation/latest/representations.html

Exception Handling-3

cphbusiness

To Convert Java Exceptions into a sufficient Error Response we can use the **ExceptionMapper** class. These objects know how to map a thrown business exception to a Response object.

```
@Provider
public class PersonNotFoundExceptionMapper implements
ExceptionMapper<PersonNotFoundException> {
    static Gson gson = new GsonBuilder().setPrettyPrinting().create();
    @Context
    ServletContext context;

    @Override
    public Response toResponse(PersonNotFoundException ex) {
       boolean isDebug = context.getInitParameter("debug").equals("true");
       ErrorMessage err = new ErrorMessage(ex,404,isDebug);
       err.setDescription("You tried to call ...");
       return Response.status(404)
                 .entity(gson.toJson(err))
                 .type(MediaType.APPLICATION_JSON).
                 build();
    }
}
```

https://jersey.java.net/documentation/latest/representations.html

Exception Handling-4

cphbusiness

ErrorMessage class used by the previous slide. Makes it simple to map an instance to JSON, for example using Gson.

```
public class ErrorMessage {

  public ErrorMessage(Throwable ex, int code,boolean debug) {
    this.code = code;
    this.message = ex.getMessage();
    this.description = ex.getMessage();
    if(debug){
      StringWriter sw = new StringWriter();
      ex.printStackTrace(new PrintWriter(sw));
      this.stackTrace = sw.toString();
    }
  }

  private int code;
  private String message;
  private String description;
  private String stackTrace;
}
```