08: REST constraints (i.e. design rules) interview Q&A

Posted on June 17, 2015 by Arulkumaran Kumaraswamipillai

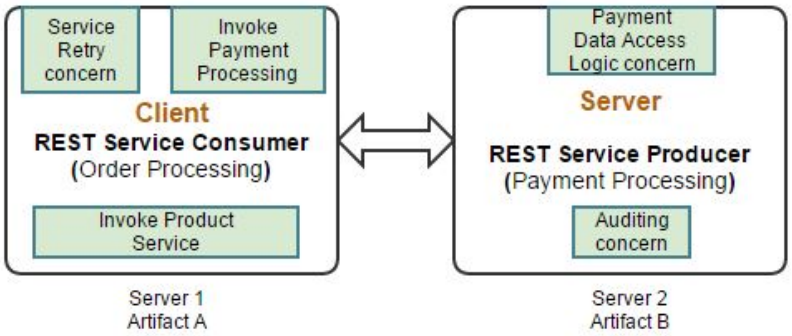
Q1. What are the 6 REST constraints?

A1. REST constraints are design rules that are applied to establish the distinct characteristics of the REST architectural style. These constraints don't dictate what kind of technology to use, and they only define how data is transferred between components and what benefits we reap by following these constraints.

- Client-Server
- Stateless
- Cacheable
- Uniform Interface
- Layered System
- Code On Demand (Optional)

Q2. What is the key focus of “Client-Server” constraint?
A2. Separation of concerns and loose coupling.

The purpose of this division is to separate architecture and responsibilities in both environments. Thus, the client (consumer of the service) does not care about tasks like database logic, cache management, logging, auditing, etc and the service provider is not concerned about tasks such as service retry, user experience, etc. Fundamentally this is a distributed architecture, thereby supporting the independent deployment of client-side & server-side artifacts and evolution of the client-side logic and server-side logic.



Note: A client can be browser or another application written in Java or other languages.

So, implementing a concern wrong side of wire or having a concern half implemented on client side and the rest on the server side can tightly couple client and server. For example, client having its own payment logic in addition to the logic on the server side. This means, if the server logic changes then the client logic may need to change as well.

Q3. Why should the REST call be stateless?
A3. Stateless services are more **scalable** and **reliable** as the state is not maintained on the server. The call can go to any server on the cluster as state is not maintained on the server. This means each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the

16 Technical Key Areas

80+ step by step Java Tutorials

server. The session state is therefore kept entirely on the client. This improves **Visibility** since a monitoring system does not have to look beyond a single request. **Reliability** is also improved due to easier recoverability from any partial failures.

Q. What are the tradeoffs of being stateless?
A. Reduced network performace due to larger payload. This is where the constraint of "**Cacheable**" becomes handy. The "second fetch" doesn't have to be made at all if the data is already sitting in your local cache. If the data can be designed in such a way to take advantage of this, you can reduce total network traffic.

REST services use both server-side caching and client-side caching. Server side caching requires tools like Squid from squid-cache-org, DynaCache from IBM, etc. Client-side caching is achieved by using cache directives in the response header. "Expires", "Cache-Control: max-age=n", etc. The "**expiration model**" of caching is good for static resources like images. For dynamic data, you should use a "**validation model**" of caching.

Q. What is a "Validation Model" for caching involving both client and server side?
A. I this model a unique identifier called an **ETag** is used. When you make a subsequent request to the same resource, you should send this **ETag**. The server uses this identifier to check if the resource you requested has changed. If the resource has changed, it sends you the latest copy.If not changed, it sends a response code of "304 Not Modified". Validation model requires development effort on both client and server side. The validation model uses **ETag** and Last-**Modified** HTTP Headers.

Q What are the tradeoffs of caching?
A Stale data may reduce reliability.

Q4. What do you understand by the term "Uniform Interface" with respect to REST?
A4. This basically means a well defined and easily

Tutorial - Diagnosis (2
Akka Tutorial (9)
Core Java Tutorials (2
Hadoop & Spark Tuto
JEE Tutorials (19)
Scala Tutorials (1)
Spring & HIbernate Tu
Tools Tutorials (19)
Other Tutorials (45)

understandbale contract for communication between clients and server. A uniform interface has four sub-constraints

1. **Identification of resources:** Each resource must have a specific and cohesive URI to be made available. E.g. http://localhost:8080/mayapp/user/1234. REST URI Conventions
2. **Manipulation of resources through representations:** Representation of the resource need to provide enough information to manipulate the resource. For example, Book info with links to provide comments and ratings, list of records with delete button, etc.
3. **Self descriptive messages:** Request or Response can contain data and metadata. The metadata in the request and response can be: HTTP response code, Host, Content-Type, etc. The "Content-Type" is very important as clients and servers rely on this header's value to tell them how to process the sequence of bytes in a message's body
4. **Hypermedia as the engine of application state (**HATEOAS**)**: This returns all the necessary information in response so client knows how to navigate and have access to all application resources. How does it provide state transition, scalability, and loose coupling?

Q5. What does the layered system and code on demand mean with respect to RES constraints?
A5. Your application should consist of layers, and these layers should be easy to change, both to add more layers and to remove them. Components can act as client on one side and as servers on the other side. You can have intermediary services to reduce complxity. The intermediary proxies can provide services like data translation, performance enhancement, load balancing, shared caching, service retries, security, etc.

Code on demand is an optional constraint. It allows a client to download and execute code from a server. For example, JavaScripts, Applets, Flash, etc can be downloaded from the server and executed on the client side. This may have some

tradeoffs like reducing the visibility and some security ramnifications. It is optional because some clients may not support execution of a code.

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

766 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003,

and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ 05: RESTFul Web Service URI conventions with Spring MVC examples

03: ♥ JAX-RS how to create a RESTFul Web Service in Java? ›

Posted in member-paid, WebService

Tags: Architect FAQs

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer

 ↑