Q: How does class loading work when the same class exists in different applications on the same server?

Ans:

A Java classloader typically works by looking for classes in one or more places in a fixed sequence. For instance, the classloader that loads your application when you run it from the command line looks first in the rt.jar file (and others on the bootclasspath), and then in the directories and JAR files specified by your classpath.

A webapp classloading is similar in principle, but a bit more complicated in practice. For a particular webapp, a webapp's classloader looks for classes in the following order. For example Tomcat 6 looks for classes in this order:

1. Bootstrap classes of your JVM
2. System class loader classes (described [here](http://tomcat.apache.org/tomcat-6.0-doc/class-loader-howto.html))
3. /WEB-INF/classes of the webapp
4. /WEB-INF/lib/\*.jar of the webapp
5. $CATALINA\_HOME/lib
6. $CATALINA\_HOME/lib/\*.jar

Of course, once the classloader has found the class it is looking for, it looks no further. So classes with the same name later in the order won't get loaded.

The complication is that the web container has one classloader for each webapp, and these classloaders delegate to other classloaders that manage the common classes. In practice, this means that some classes will only ever be loaded once for the entire container (e.g. 1. and 2.) and others may get loaded multiple times by different classloaders.

(When a class is loaded more than once, it results in distinct Class objects and distinct class statics. The versions of the class are different types as far as the JVM is concerned and you cannot typecast from one version to the other.)

Finally, Tomcat can be configure to allow individual webapps to be "hot loaded". This entails stopping a webapp, creating a new classloader for it, and restarting it.

**FOLLOWUP**

So ... synchronizing a static method will not protect access to a shared resource where the class has been loaded multiple times?

It depends on the details, but it probably won't. (Or to look at if another way, if a class has *actually*been loaded multiple times, then a static method of each "load" of the class will access a different set of static fields.)

If you really want a singleton application class instance to be shared by multiple webapps in the same container, it is simplest if you put the class into $CATALINA\_HOME/lib or the equivalent. But you also should ask yourself if this is good system design. Consider combining the webapps, or to using request forwarding etc instead of a shared data structure. The singleton pattern tends to be troublesome in webapps, and this flavor is even more so.

Q: What is a Java ClassLoader?

Taken from this nice [tutorial](http://www.oracle.com/technetwork/articles/javase/classloaders-140370.html) from Sun:

Motivation

Applications written in statically compiled programming languages, such as C and C++, are compiled into native, machine-specific instructions and saved as an executable file. The process of combining the code into an executable native code is called linking - the merging of separately compiled code with shared library code to create an executable application. This is different in dynamically compiled programming languages such as Java. In Java, the .class files generated by the Java compiler remain as-is until loaded into the Java Virtual Machine (JVM) -- in other words, the linking process is performed by the JVM at runtime. Classes are loaded into the JVM on an 'as needed' basis. And when a loaded class depends on another class, then that class is loaded as well.

When a Java application is launched, the first class to run (or the entry point into the application) is the one with public static void method called main(). This class usually has references to other classes, and all attempts to load the referenced classes are carried out by the class loader.

To get a feeling of this recursive class loading as well as the class loading idea in general, consider the following simple class:

public class HelloApp {

public static void main(String argv[]) {

System.out.println("Aloha! Hello and Bye");

}

}

If you run this class specifying the -verbose:class command-line option, so that it prints what classes are being loaded, you will get an output that looks as follows. Note that this is just a partial output since the list is too long to show here.

prmpt>java -verbose:class HelloApp

[Opened C:\Program Files\Java\jre1.5.0\lib\rt.jar]

[Opened C:\Program Files\Java\jre1.5.0\lib\jsse.jar]

[Opened C:\Program Files\Java\jre1.5.0\lib\jce.jar]

[Opened C:\Program Files\Java\jre1.5.0\lib\charsets.jar]

[Loaded java.lang.Object from shared objects file]

[Loaded java.io.Serializable from shared objects file]

[Loaded java.lang.Comparable from shared objects file]

[Loaded java.lang.CharSequence from shared objects file]

[Loaded java.lang.String from shared objects file]

[Loaded java.lang.reflect.GenericDeclaration from shared objects file]

[Loaded java.lang.reflect.Type from shared objects file]

[Loaded java.lang.reflect.AnnotatedElement from shared objects file]

[Loaded java.lang.Class from shared objects file]

[Loaded java.lang.Cloneable from shared objects file]

[Loaded java.lang.ClassLoader from shared objects file]

[Loaded java.lang.System from shared objects file]

[Loaded java.lang.Throwable from shared objects file]

.

.

.

[Loaded java.security.BasicPermissionCollection from shared objects file]

[Loaded java.security.Principal from shared objects file]

[Loaded java.security.cert.Certificate from shared objects file]

[Loaded HelloApp from file:/C:/classes/]

Aloha! Hello and Bye

[Loaded java.lang.Shutdown from shared objects file]

[Loaded java.lang.Shutdown$Lock from shared objects file]

As you can see, the Java runtime classes required by the application class (HelloApp) are loaded first.

Class Loaders in the Java 2 Platform

The Java programming language keeps evolving to make the life of applications developers easier everyday. This is done by providing APIs that simplify your life by allowing you to concentrate on business logic rather than implementation details of fundamental mechanisms. This is evident by the recent change of J2SE 1.5 to J2SE 5.0 in order to reflect the maturity of the Java platform.

As of JDK 1.2, a bootstrap class loader that is built into the JVM is responsible for loading the classes of the Java runtime. This class loader only loads classes that are found in the boot classpath, and since these are trusted classes, the validation process is not performed as for untrusted classes. In addition to the bootstrap class loader, the JVM has an extension class loader responsible for loading classes from standard extension APIs, and a system class loader that loads classes from a general class path as well as your application classes.

Since there is more than one class loader, they are represented in a tree whose root is the bootstrap class loader. Each class loader has a reference to its parent class loader. When a class loader is asked to load a class, it consults its parent class loader before attempting to load the item itself. The parent in turn consults its parent, and so on. So it is only after all the ancestor class loaders cannot find the class that the current class loader gets involved. In other words, a delegation model is used.

The java.lang.ClassLoader Class

The java.lang.ClassLoader is an abstract class that can be subclassed by applications that need to extend the manner in which the JVM dynamically loads classes. Constructors in java.lang.ClassLoader (and its subclasses) allow you to specify a parent when you instantiate a new class loader. If you don't explicitly specify a parent, the virtual machine's system class loader will be assigned as the default parent. In other words, the ClassLoader class uses a delegation model to search for classes and resources. Therefore, each instance of ClassLoader has an associated parent class loader, so that when requested to find a class or resources, the task is delegated to its parent class loader before attempting to find the class or resource itself. The loadClass() method of the ClassLoader performs the following tasks, in order, when called to load a class:

If a class has already been loaded, it returns it. Otherwise, it delegates the search for the new class to the parent class loader. If the parent class loader doesn't find the class, loadClass() calls the method findClass() to find and load the class. The finalClass() method searches for the class in the current class loader if the class wasn't found by the parent class loader.

There's more in the original article, which also shows you how to implement your own network class loaders, which answers your question as to why (and how). See also the [API docs](http://java.sun.com/javase/6/docs/api/java/lang/ClassLoader.html).

Java Classloader:

The Java Classloader is a part of the [Java Runtime Environment](https://en.wikipedia.org/wiki/Java_Runtime_Environment) that [dynamically loads](https://en.wikipedia.org/wiki/Dynamic_loading) [Java classes](https://en.wikipedia.org/wiki/Java_class) into the [Java Virtual Machine](https://en.wikipedia.org/wiki/Java_Virtual_Machine).

Each Java class must be loaded by a class loader.

When the JVM is started, three class loaders are used:[[3]](https://en.wikipedia.org/wiki/Java_Classloader" \l "cite_note-3)[[4]](https://en.wikipedia.org/wiki/Java_Classloader#cite_note-4)

1. Bootstrap class loader
2. Extensions class loader
3. System class loader

The bootstrap class loader loads the core Java libraries[[5]](https://en.wikipedia.org/wiki/Java_Classloader" \l "cite_note-5) located in the <JAVA\_HOME>/jre/lib directory. This class loader, which is part of the core JVM, is written in native code.

The extensions class loader loads the code in the extensions directories (<JAVA\_HOME>/jre/lib/ext,[[6]](https://en.wikipedia.org/wiki/Java_Classloader" \l "cite_note-6) or any other directory specified by the java.ext.dirs system property). It is implemented by the sun.misc.Launcher$ExtClassLoader class.

The system class loader loads code found on java.class.path, which maps to the [CLASSPATH](https://en.wikipedia.org/wiki/Classpath_(Java)) [environment variable](https://en.wikipedia.org/wiki/Environment_variable). This is implemented by the sun.misc.Launcher$AppClassLoaderclass.

# Java VisualVM - Monitoring Application Threads

Java VisualVM presents data for local and remote applications in a tab specific for that application. Application tabs are displayed in the main window to the right of the Applications window. You can have multiple application tabs open at one time. Each application tab contains sub-tabs that display different types of information about the application.

## Monitoring Thread Activity

Java VisualVM displays real-time, high-level data on thread activity in the Threads tab.  
**Note:** The information displayed in the Threads tab is based on Java Management Extensions (JMX). The Threads tab is visible if Java VisualVM can make a JMX technology-based connection (JMX connection) with the target application and retrieve JMX instrumentation from the Java Virtual Machine (JVM). If the target application is a local application running on Java Development Kit (JDK) version 1.6, the JMX connection is made automatically. If the target application is not running on JDK version 1.6, you may need to explicitly establish a JMX connection with the JVM software. For more on establishing a JMX connection, see the following document:

* [Connecting to JMX Agents Explicitly](https://docs.oracle.com/javase/8/docs/technotes/guides/visualvm/jmx_connections.html)

By default the Threads tab displays a timeline of current thread activity. You can click a thread in the timeline to view details about that thread in the Details tab.

### Timeline Tab

This tab displays a timeline with real-time thread states. Use the buttons in the Timeline toolbar to zoom in/out on the current view and to switch to the Scale to Fit mode. The drop-down list enables you to select which threads are displayed. You can choose to view all threads, live threads or finished threads. You can also select a single thread or multiple threads to display a subset of the threads. You can double-click on a thread timeline to open that thread in the Details tab.

A timeline for each thread provides a quick overview of the thread's activity.

## Taking a Thread Dump

You can use Java VisualVM to take a thread dump (stack trace) while a local application is running. Taking a thread dump does not stop the application. When you print the thread dump you get a printout of the thread stack that includes thread states for the Java threads.

When you print a thread dump in Java VisualVM, the tool prints a stack trace of the active threads of the application. Using Java VisualVM to take a thread dump can be very convenient in cases where you do not have a command-line console for the application. You can use a stack trace to help diagnose a number of issues such as deadlocks or when an application hangs.

Screenshot of thread dump (stack trace) in thread dump sub-tab.

Q: How to identify stuck thread in java? How to resolve issue?

1. By using Visual VM take Thread dump, and kill the threads which are stuck.
2. Use interrupt(), instead of thread.stop().

Q: prevent deadlocks in java?

Some quick tips out of my head

* don't use multiple threads (like Swing does, for example, by mandating that everything is done in the EDT)
* don't hold several locks at once. If you do, always acquire the locks in the same order
* don't execute foreign code while holding a lock
* use interruptible locks

Q: What is a covariant return type?

Covariant return, means that when one overrides a method, the return type of the overriding method is allowed to be a subtype of the overridden method's return type.

To clarify this with an example, a common case is Object.clone() - which is declared to return a type of Object. You could override this in your own class as follows:

public class MyFoo

{

...

// Note covariant return here, method does not just return Object

public MyFoo clone()

{

// Implementation

}

}

The benefit here is that any method which holds an explicit reference to a MyFoo object will be able to invoke clone() and know (without casting) that the return value is an instance of MyFoo. Without covariant return types, the overridden method in MyFoo would have to be declared to return Object- and so calling code would have to explicitly downcast the result of the method call (even thought both sides "know" it can only ever be an instance of MyFoo).

Note that there's nothing special about clone() and that any overridden method can have a covariant return - I used it as an example here as it's a standard method where this is often useful.

**Q: What is the difference between a process and a thread?**

Both processes and threads are independent sequences of execution. The typical difference is that threads (of the same process) run in a shared memory space, while processes run in separate memory spaces.

I'm not sure what "hardware" vs "software" threads you might be referring to. Threads are an operating environment feature, rather than a CPU feature (though the CPU typically has operations that make threads efficient).

Erlang uses the term "process" because it does not expose a shared-memory multiprogramming model. Calling them "threads" would imply that they have shared memory

**Process**  
Each process provides the resources needed to execute a program. A process has a virtual address space, executable code, open handles to system objects, a security context, a unique process identifier, environment variables, a priority class, minimum and maximum working set sizes, and at least one thread of execution. Each process is started with a single thread, often called the primary thread, but can create additional threads from any of its threads.

**Thread**  
A thread is an entity within a process that can be scheduled for execution. All threads of a process share its virtual address space and system resources. In addition, each thread maintains exception handlers, a scheduling priority, thread local storage, a unique thread identifier, and a set of structures the system will use to save the thread context until it is scheduled. The thread context includes the thread's set of machine registers, the kernel stack, a thread environment block, and a user stack in the address space of the thread's process. Threads can also have their own security context, which can be used for impersonating clients

Q: executorService.execute(**new** Task(i));

Executes the given task sometime in the future. The task may execute in a new thread or in an existing pooled thread. If the task cannot be submitted for execution, either because this executor has been shutdown or because its capacity has been reached, the task is handled by the current RejectedExecutionHandler.

Q: How does a single servlet handle multiple requests from client side?

Q: Hibernate get() vs load()

Ans: get() is eager method, it will fetch the records from the Db, where load is lazy, it will hit the DB on demand.

If object is not found in DB, get() returns null and load() will throw the ObjectNotFoundException.

load() will return proxy object.

Q: Hibernate hbm2ddl.auto possible values and what they do?

These are changes that could happen over DB:

* New tables
* new columns in old tables
* columns deleted
* data type of a column changed
* a type of a column changed it attributes
* tables have been dropped
* values of a column has changed

In each case what is the best solution?

From the [community documentation](https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html#configurations-hbmddl):

hibernate.hbm2ddl.auto Automatically validates or exports schema DDL to the database when the SessionFactory is created. With create-drop, the database schema will be dropped when the SessionFactory is closed explicitly.

e.g. validate | update | create | create-drop

So the list of possible options are,

* validate: validate the schema, makes no changes to the database.
* update: update the schema.
* create: creates the schema, destroying previous data.
* create-drop: drop the schema when the SessionFactory is closed explicitly, typically when the application is stopped.

These options seem intended to be developers tools and not to facilitate any production level databases, you may want to have a look at the following question; [Hibernate: hbm2ddl.auto=update in production?](https://stackoverflow.com/questions/221379/hibernate-hbm2ddl-autoupdate-in-production)

Q: JPA JoinColumn vs mappedBy?

What is the difference between:

@Entity

public class Company {

@OneToMany(cascade = CascadeType.ALL , fetch = FetchType.LAZY)

@JoinColumn(name = "companyIdRef", referencedColumnName = "companyId")

private List<Branch> branches;

...

}

and

@Entity

public class Company {

@OneToMany(cascade = CascadeType.ALL , fetch = FetchType.LAZY, mappedBy = "companyIdRef")

private List<Branch> branches;

...

}

Ans:

**@JoinColumn could be used on both sides of the relationship.** The question was about using @JoinColumn on the @OneToMany side (rare case). And the point here is in physical information duplication (column name) along with not optimized SQL query that will produce some additional UPDATE statements.

According to [documentation](http://docs.jboss.org/hibernate/annotations/3.5/reference/en/html_single/):

Since **many to one are** (almost) always the **owner side** of a bidirectional relationship in the JPA spec, the one to many association is annotated by @OneToMany(mappedBy=...)

@Entity

public class Troop {

@OneToMany(mappedBy="troop")

public Set<Soldier> getSoldiers() {

...

}

@Entity

public class Soldier {

@ManyToOne

@JoinColumn(name="troop\_fk")

public Troop getTroop() {

...

}

Troop has a bidirectional one to many relationship with Soldier through the troop property. You don't have to (must not) define any physical mapping in the mappedBy side.

To map a bidirectional one to many, with the **one-to-many side as the owning side**, you have to remove the mappedBy element and set the many to one @JoinColumn as insertable and updatable to false. This solution is not optimized and will produce some additional UPDATE statements.

@Entity

public class Troop {

@OneToMany

@JoinColumn(name="troop\_fk") //we need to duplicate the physical information

public Set<Soldier> getSoldiers() {

...

}

@Entity

public class Soldier {

@ManyToOne

@JoinColumn(name="troop\_fk", insertable=false, updatable=false)

public Troop getTroop() {

...

}

Ans2:

The annotation @JoinColumn indicates that this entity is the owner of the relationship (that is: the corresponding table has a column with a foreign key to the referenced table), whereas the attribute mappedBy indicates that the entity in this side is the inverse of the relationship, and the owner resides in the "other" entity. This also means that you can access the other table from the class which you've annotated with "mappedBy" (fully bidirectional relationship).

In particular, for the code in the question the correct annotations would look like this:

@Entity

public class Company {

@OneToMany(fetch = FetchType.LAZY, mappedBy = "company")

private List<Branch> branches;

}

@Entity

public class Branch {

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "companyId")

private Company company;

}

The [JPA 2.0 Specification](http://www.jcp.org/en/jsr/detail?id=317) states that:

* The entity class must have a no-arg constructor. It may have other constructors as well. The no-arg constructor must be public or protected.
* The entity class must a be top-level class. An enum or interface must not be designated as an entity.
* The entity class must not be final. No methods or persistent instance variables of the entity class may be final.
* **If an entity instance is to be passed by value as a detached object** (e.g., through a remote interface), the entity class must implement the Serializable interface.
* Both abstract and concrete classes can be entities. Entities may extend non-entity classes as well as entity classes, and non-entity classes may extend entity classes.

The specification contains no requirements about the implementation of equals and hashCode methods for entities, only for primary key classes and map keys as far as I know.

Q: Only using **@JsonIgnore** during serialization, but not deserialization

Exactly how to do this depends on the version of Jackson that you're using. This changed around version 1.9, before that, you could do this by adding @JsonIgnore to the getter.

Which you've tried:

Add @JsonIgnore on the getter method only

Do this, and also add a specific @JsonProperty annotation for your JSON "password" field name to the setter method for the password on your object.

More recent versions of Jackson have added READ\_ONLY and WRITE\_ONLY annotation arguments for JsonProperty. So you could also do something like:

@JsonProperty(access = Access.WRITE\_ONLY)

private String password;

Q: How to forward a REST request to another resource?

@MatrixParam

@headerParam

@CookieParam

@Context uriInfo uriInfo

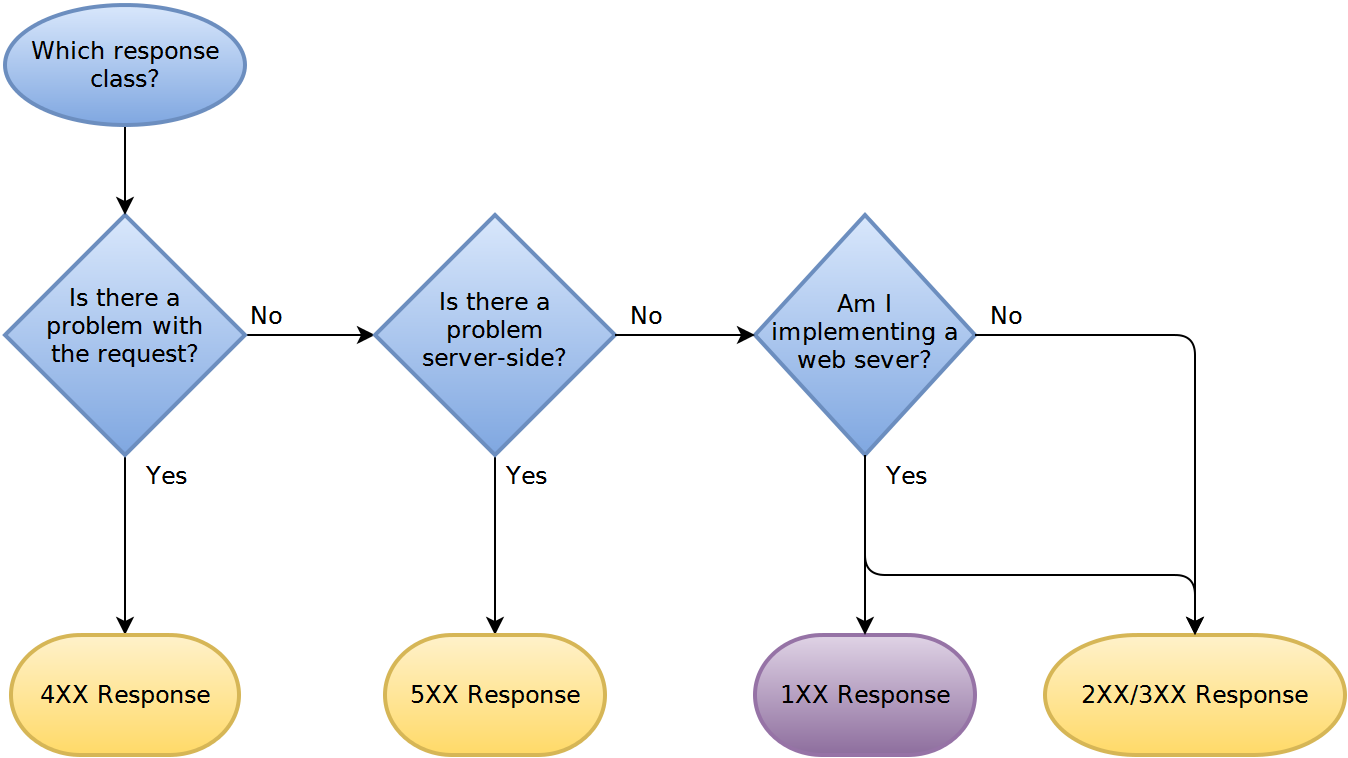
@Contect HttpHeaders header;

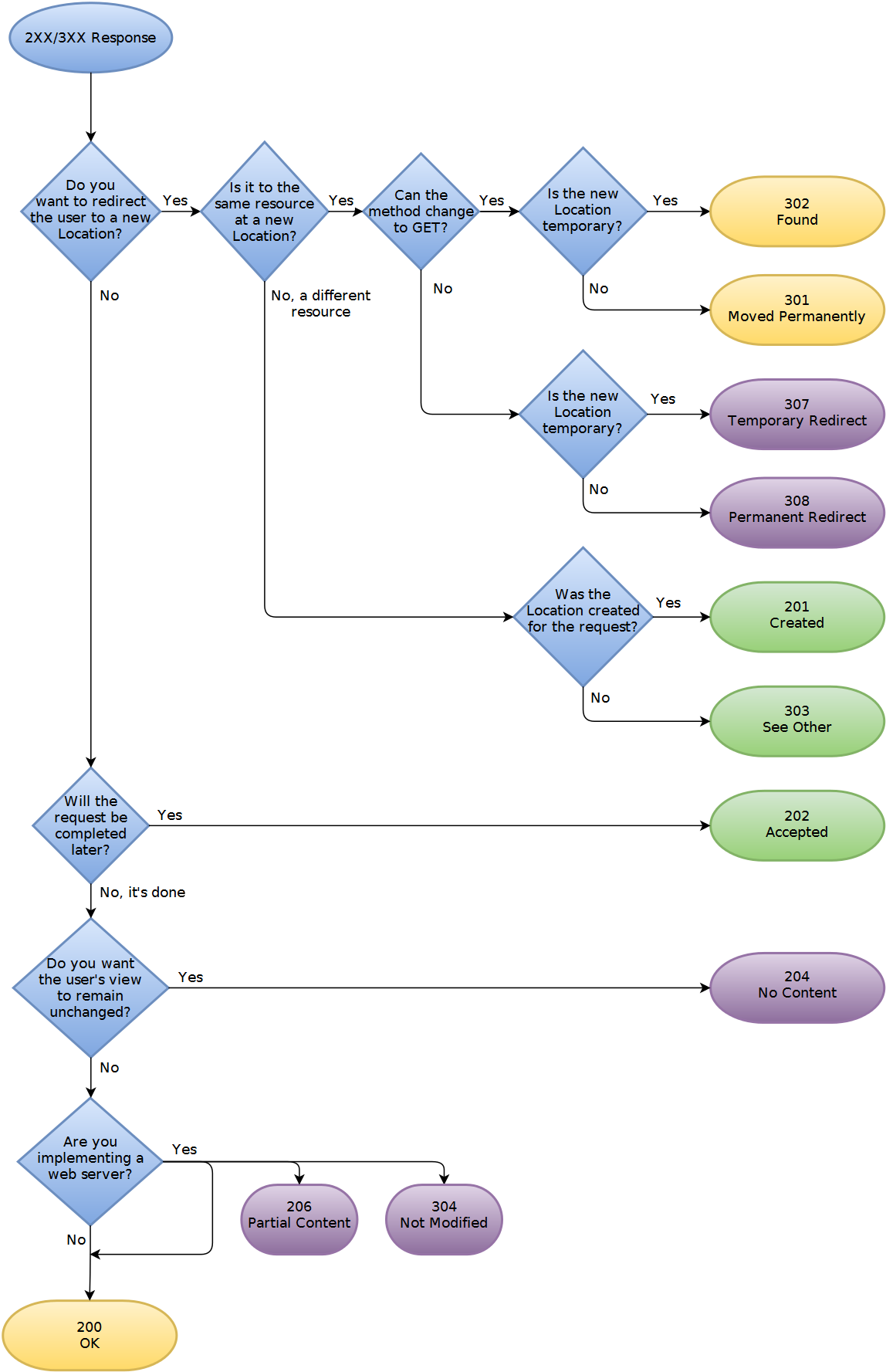
@Contect HttpHeaders header;

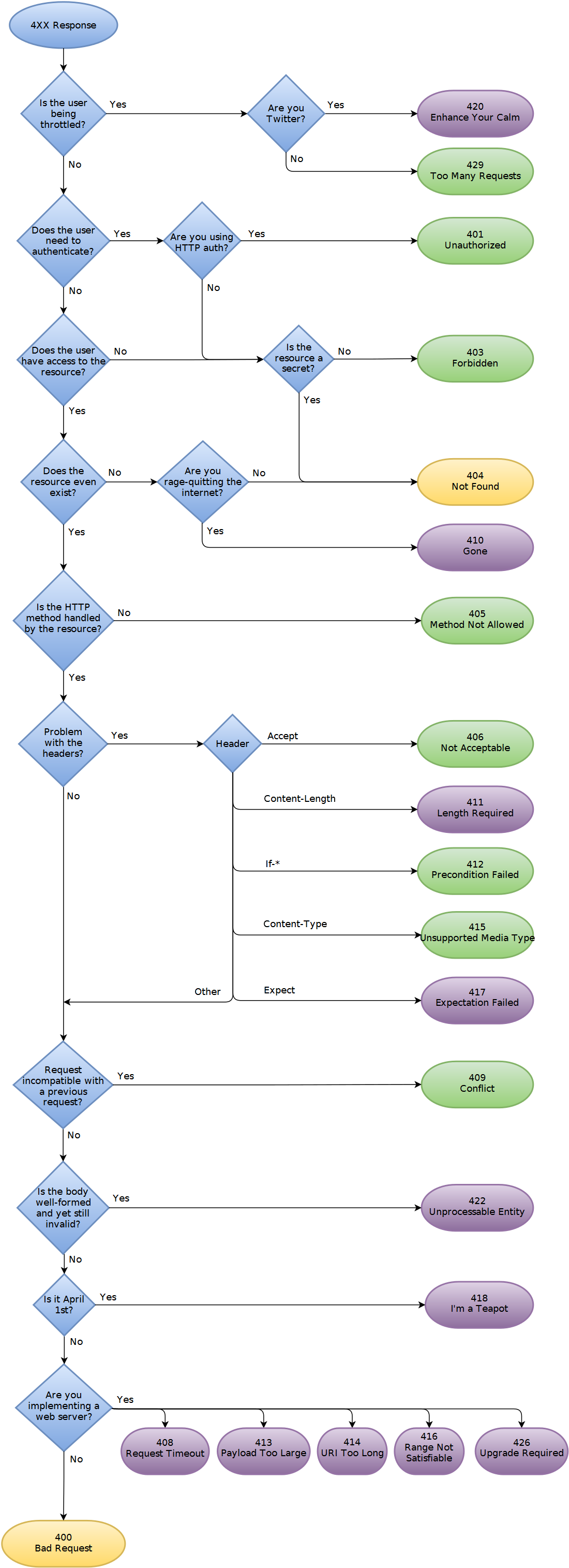
@BeanParam-> CustomBean🡪it should have properties with @MatrixParam and @headerParam

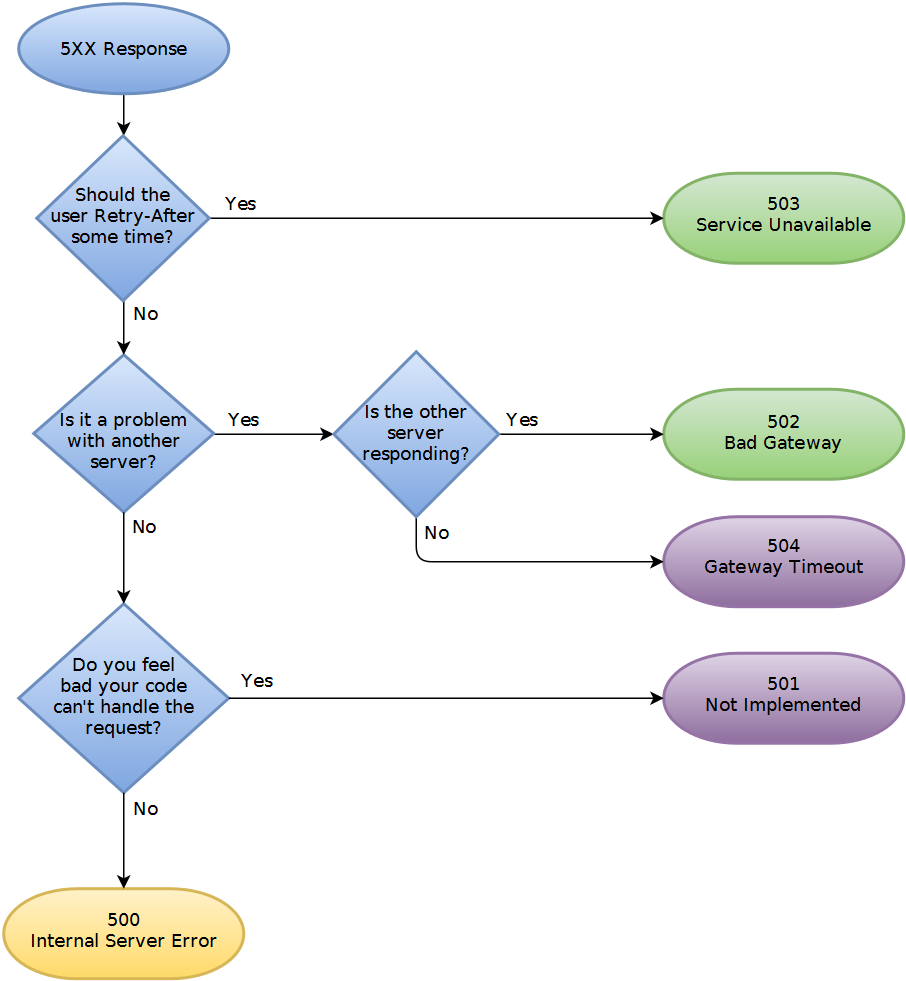
@path annotation is optional for Subresources

@XMLTransient









Q: How do I upload a file with metadata using a REST web service?

You basically have three choices:

1. Base64 encode the file, at the expense of increasing the data size by around 33%.
2. Send the file first in a multipart/form-data POST, and return an ID to the client. The client then sends the metadata with the ID, and the server re-associates the file and the metadata.
3. Send the metadata first, and return an ID to the client. The client then sends the file with the ID, and the server re-associates the file and the metadata.

I agree with Greg that a two phase approach is a reasonable solution, however I would do it the other way around. I would do:

POST http://server/data/media

body:

{

"Name": "Test",

"Latitude": 12.59817,

"Longitude": 52.12873

}

To create the metadata entry and return a response like:

201 Created

Location: http://server/data/media/21323

{

"Name": "Test",

"Latitude": 12.59817,

"Longitude": 52.12873,

"ContentUrl": "http://server/data/media/21323/content"

}

The client can then use this ContentUrl and do a PUT with the file data.

The nice thing about this approach is when your server starts get weighed down with immense volumes of data, the url that you return can just point to some other server with more space/capacity. Or you could implement some kind of round robin approach if bandwidth is an issue.

Q: Understanding REST: Verbs, error codes, and authentication

Question:

I am looking for a way to wrap APIs around default functions in my PHP-based web applications, databases and CMSs.

I have looked around and found several "skeleton" frameworks. In addition to the answers in my question, there is [Tonic](http://tonic.sourceforge.net/), a REST framework I like because it is very lightweight.

I like REST the best for its simplicity, and would like to create an API architecture based on it. I'm trying to get my head around the basic principles and have not fully understood it yet. Therefore, a number of questions.

**1. Am I understanding it right?**

Say I have a resource "users". I could set up a number of URIs like so:

/api/users when called with GET, lists users

/api/users when called with POST, creates user record

/api/users/1 when called with GET, shows user record

when called with PUT, updates user record

when called with DELETE, deletes user record

is this a correct representation of a RESTful architecture so far?

**2. I need more verbs**

Create, Update and Delete may be enough in theory, but in practice I will have the need for a lot more verbs. I realize these are things that could be embedded in an update request, but they are specific actions that can have specific return codes and I wouldn't want to throw them all into one action.

Some that come to mind in the user example are:

activate\_login

deactivate\_login

change\_password

add\_credit

how would I express actions such as those in a RESTful URL architecture?

My instinct would be to do a GET call to a URL like

/api/users/1/activate\_login

and expect a status code back.

That deviates from the idea of using HTTP verbs, though. What do you think?

**3. How to return error messages and codes**

A great part of REST's beauty stems from its use of standard HTTP methods. On an error, I emit a header with a 3xx,4xx or 5xx error status code. For a detailed error description, I can use the body (right?). So far so good. But what would be the way to transmit a **proprietary error code** that is more detailed in describing what went wrong (e.g. "failed to connect to database", or "database login wrong")? If I put it into the body along with the message, I have to parse it out afterwards. Is there a standard header for this kind of thing?

**4. How to do authentication**

* What would a API key based authentication following REST principles look like?
* Are there strong points against using sessions when authenticating a REST client, other than that it's a blatant violation of the REST principle? :) (only half kidding here, session based authentication would play well with my existing infrastructure.)

Ans:

I noticed this question a couple of days late, but I feel that I can add some insight. I hope this can be helpful towards your RESTful venture.

**Point 1: Am I understanding it right?**

You understood right. That is a correct representation of a RESTful architecture. You may find the following matrix from [Wikipedia](http://en.wikipedia.org/wiki/Representational_State_Transfer#Applied_to_web_services) very helpful in defining your nouns and verbs:

When dealing with a **Collection** URI like: **http://example.com/resources/**

* **GET**: List the members of the collection, complete with their member URIs for further navigation. For example, list all the cars for sale.
* **PUT**: Meaning defined as "replace the entire collection with another collection".
* **POST**: Create a new entry in the collection where the ID is assigned automatically by the collection. The ID created is usually included as part of the data returned by this operation.
* **DELETE**: Meaning defined as "delete the entire collection".

When dealing with a **Member** URI like: **http://example.com/resources/7HOU57Y**

* **GET**: Retrieve a representation of the addressed member of the collection expressed in an appropriate MIME type.
* **PUT**: Update the addressed member of the collection or create it with the specified ID.
* **POST**: Treats the addressed member as a collection in its own right and creates a new subordinate of it.
* **DELETE**: Delete the addressed member of the collection.

**Point 2: I need more verbs**

In general, when you think you need more verbs, it may actually mean that your resources need to be re-identified. Remember that in REST you are always acting on a resource, or on a collection of resources. What you choose as the resource is quite important for your API definition.

**Activate/Deactivate Login**: If you are creating a new session, then you may want to consider "the session" as the resource. To create a new session, use POST to http://example.com/sessions/with the credentials in the body. To expire it use PUT or a DELETE (maybe depending on whether you intend to keep a session history) to http://example.com/sessions/SESSION\_ID.

**Change Password:** This time the resource is "the user". You would need a PUT to http://example.com/users/USER\_ID with the old and new passwords in the body. You are acting on "the user" resource, and a change password is simply an update request. It's quite similar to the UPDATE statement in a relational database.

My instinct would be to do a GET call to a URL like /api/users/1/activate\_login

This goes against a very core REST principle: The correct usage of HTTP verbs. Any GET request should never leave any side effect.

For example, a GET request should never create a session on the database, return a cookie with a new Session ID, or leave any residue on the server. The GET verb is like the SELECT statement in a database engine. Remember that the response to any request with the GET verb should be cache-able when requested with the same parameters, just like when you request a static web page.

**Point 3: How to return error messages and codes**

Consider the 4xx or 5xx HTTP status codes as error categories. You can elaborate the error in the body.

**Failed to Connect to Database:** / **Incorrect Database Login**: In general you should use a 500 error for these types of errors. This is a server-side error. The client did nothing wrong. 500 errors are normally considered "retryable". i.e. the client can retry the same exact request, and expect it to succeed once the server's troubles are resolved. Specify the details in the body, so that the client will be able to provide some context to us humans.

The other category of errors would be the 4xx family, which in general indicate that the client did something wrong. In particular, this category of errors normally indicate to the client that there is no need to retry the request as it is, because it will continue to fail permanently. i.e. the client needs to change something before retrying this request. For example, "Resource not found" (HTTP 404) or "Malformed Request" (HTTP 400) errors would fall in this category.

**Point 4: How to do authentication**

As pointed out in point 1, instead of authenticating a user, you may want to think about creating a session. You will be returned a new "Session ID", along with the appropriate HTTP status code (200: Access Granted or 403: Access Denied).

You will then be asking your RESTful server: "Can you GET me the resource for this Session ID?".

There is no authenticated mode - REST is stateless: You create a session, you ask the server to give you resources using this Session ID as a parameter, and on logout you drop or expire the session.

Q: REST Content-Type: Should it be based on extension or Accept header?

Should the representation(html, xml, json) returned by a RESTful web service be determined by the url or by the Accept HTTP header?

Ans:

Both are valid. Quote from [xml.com](http://www.xml.com/pub/a/2004/08/11/rest.html):

A resource may have more than one representation. There are four frequently used ways of delivering the correct resource representation to consumers:

1. Server-driven negotiation. The service provider determines the right representation from prior knowledge of its clients or uses the information provided in HTTP headers like Accept, Accept-Charset, Accept-Encoding, Accept-Language, and User-Agent. The drawback of this approach is that the server may not have the best knowledge about what a client really wants.
2. Client-driven negotiation. A client initiates a request to a server. The server returns a list of available of representations. The client then selects the representation it wants and sends a second request to the server. The drawback is that a client needs to send two requests.
3. Proxy-driven negotiation. A client initiates a request to a server through a proxy. The proxy passes the request to the server and obtains a list of representations. The proxy selects one representation according to preferences set by the client and returns the representation back to the client.
4. URI-specified representation. A client specifies the representation it wants in the URI query string.

Q: SOAP vs REST (differences)

Question: I have read articles about the differences between SOAP and REST as a web service communication protocol, but I think that the biggest advantages for REST over SOAP are:

1. REST is more dynamic, no need for creating and updating UDDI.
2. REST is not restricted to XML format. REST web services can send plain text, JSON, and also XML.

But SOAP is more standardized (Ex; security).

So, am I correct in these points?

Ans1:

Unfortunately, there are a lot of misinformation and misconceptions around REST. Not only your question and the [answer by @cmd](https://stackoverflow.com/a/19884368/282110) reflect those, but most of the questions and answers related to the subject on Stack Overflow.

SOAP and REST can't be compared directly, since the first is a protocol (or at least tries to be) and the second is an architectural style. This is probably one of the sources of confusion around it, since people tend to call REST any HTTP API that isn't SOAP.

Pushing things a little and trying to establish a comparison, the main difference between SOAP and REST is the degree of coupling between client and server implementations. A SOAP client works like a custom desktop application, tightly coupled to the server. There's a rigid contract between client and server, and everything is expected to break if either side changes anything. You need constant updates following any change, but it's easier to ascertain if the contract is being followed.

A REST client is more like a browser. It's a generic client that knows how to use a protocol and standardized methods, and an application has to fit inside that. You don't violate the protocol standards by creating extra methods, you leverage on the standard methods and create the actions with them on your media type. If done right, there's less coupling, and changes can be dealt with more gracefully. A client is supposed to enter a REST service with zero knowledge of the API, except for the entry point and the media type. In SOAP, the client needs previous knowledge on everything it will be using, or it won't even begin the interaction. Additionally, a REST client can be extended by code-on-demand supplied by the server itself, the classical example being JavaScript code used to drive the interaction with another service on the client-side.

I think these are the crucial points to understand what REST is about, and how it differs from SOAP:

* REST is protocol independent. It's not coupled to HTTP. Pretty much like you can follow an ftp link on a website, a REST application can use any protocol for which there is a standardized URI scheme.
* REST is not a mapping of CRUD to HTTP methods. Read [this](https://stackoverflow.com/questions/19843480/s3-rest-api-and-post-method/19844272#19844272) answer for a detailed explanation on that.
* REST is as standardized as the parts you're using. Security and authentication in HTTP are standardized, so that's what you use when doing REST over HTTP.
* REST is not REST without [hypermedia](https://stackoverflow.com/a/29586455/1202421) and [HATEOAS](http://en.wikipedia.org/wiki/HATEOAS). This means that a client only knows the entry point URI and the resources are supposed to return links the client should follow. Those fancy documentation generators that give URI patterns for everything you can do in a REST API miss the point completely. They are not only documenting something that's supposed to be following the standard, but when you do that, you're coupling the client to one particular moment in the evolution of the API, and any changes on the API have to be documented and applied, or it will break.
* REST is the architectural style of the web itself. When you enter Stack Overflow, you know what a User, a Question and an Answer are, you know the media types, and the website provides you with the links to them. A REST API has to do the same. If we designed the web the way people think REST should be done, instead of having a home page with links to Questions and Answers, we'd have a static documentation explaining that in order to view a question, you have to take the URI stackoverflow.com/questions/<id>, replace id with the Question.id and paste that on your browser. That's nonsense, but that's what many people think REST is.

This last point can't be emphasized enough. If your clients are building URIs from templates in documentation and not getting links in the resource representations, that's not REST. Roy Fielding, the author of REST, made it clear on this blog post: [REST APIs must be hypertext-driven](http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven).

With the above in mind, you'll realize that while REST might not be restricted to XML, to do it correctly with any other format you'll have to design and standardize some format for your links. Hyperlinks are standard in XML, but not in JSON. There are draft standards for JSON, like [HAL](http://stateless.co/hal_specification.html).

Finally, REST isn't for everyone, and a proof of that is how most people solve their problems very well with the HTTP APIs they mistakenly called REST and never venture beyond that. REST is hard to do sometimes, especially in the beginning, but it pays over time with easier evolution on the server side, and client's resilience to changes. If you need something done quickly and easily, don't bother about getting REST right. It's probably not what you're looking for. If you need something that will have to stay online for years or even decades, then REST is for you.

Q: What exactly is RESTful programming?

Ans1:

An **architectural style** called [**REST (Representational State Transfer)**](https://en.m.wikipedia.org/wiki/Representational_state_transfer) advocates that web applications should use HTTP as it was **originally envisioned**. Lookups should use [GET](https://en.m.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) requests. [PUT, POST, and DELETE requests](https://en.m.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Request_methods) should be used for **mutation, creation, and deletion respectively**.

REST proponents tend to favor URLs, such as

http://myserver.com/catalog/item/1729

but the REST architecture does not require these "pretty URLs". A GET request with a parameter

http://myserver.com/catalog?item=1729

is every bit as RESTful.

Keep in mind that GET requests should never be used for updating information. For example, a GET request for adding an item to a cart

http://myserver.com/addToCart?cart=314159&item=1729

would not be appropriate. GET requests should be [idempotent](https://en.m.wikipedia.org/wiki/Idempotence). That is, issuing a request twice should be no different from issuing it once. That's what makes the requests cacheable. An "add to cart" request is not idempotent—issuing it twice adds two copies of the item to the cart. A POST request is clearly appropriate in this context. Thus, even a **RESTful web application** needs its share of POST requests.

This is taken from the excellent book Core JavaServer faces book by David M. Geary.

Ans2:

REST is the underlying architectural principle of the web. The amazing thing about the web is the fact that clients (browsers) and servers can interact in complex ways without the client knowing anything beforehand about the server and the resources it hosts. The key constraint is that the server and client must both agree on the media used, which in the case of the web is HTML.

An API that adheres to the principles of REST does not require the client to know anything about the structure of the API. Rather, the server needs to provide whatever information the client needs to interact with the service. An HTML form is an example of this: The server specifies the location of the resource and the required fields. **The browser doesn't know in advance where to submit the information, and it doesn't know in advance what information to submit. Both forms of information are entirely supplied by the server.** (This principle is called [HATEOAS: Hypermedia As The Engine Of Application State](https://en.wikipedia.org/wiki/HATEOAS).)

**So, how does this apply to HTTP, and how can it be implemented in practice?** HTTP is oriented around verbs and resources. The two verbs in mainstream usage are GET and POST, which I think everyone will recognize. However, the HTTP standard defines several others such as PUT and DELETE. These verbs are then applied to resources, according to the instructions provided by the server.

For example, Let's imagine that we have a user database that is managed by a web service. Our service uses a custom hypermedia based on JSON, for which we assign the mimetype application/json+userdb (There might also be an application/xml+userdb and application/whatever+userdb - many media types may be supported). The client and the server have both been programmed to understand this format, but they don't know anything about each other. As [Roy Fielding](http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven) points out:

A REST API should spend almost all of its descriptive effort in defining the media type(s) used for representing resources and driving application state, or in defining extended relation names and/or hypertext-enabled mark-up for existing standard media types.

A request for the base resource / might return something like this:

**Request**

GET /

Accept: application/json+userdb

**Response**

200 OK

Content-Type: application/json+userdb

{

"version": "1.0",

"links": [

{

"href": "/user",

"rel": "list",

"method": "GET"

},

{

"href": "/user",

"rel": "create",

"method": "POST"

}

]

}

We know from the description of our media that we can find information about related resources from sections called "links". This is called Hypermedia controls. In this case, we can tell from such a section that we can find a user list by making another request for /user:

**Request**

GET /user

Accept: application/json+userdb

**Response**

200 OK

Content-Type: application/json+userdb

{

"users": [

{

"id": 1,

"name": "Emil",

"country: "Sweden",

"links": [

{

"href": "/user/1",

"rel": "self",

"method": "GET"

},

{

"href": "/user/1",

"rel": "edit",

"method": "PUT"

},

{

"href": "/user/1",

"rel": "delete",

"method": "DELETE"

}

]

},

{

"id": 2,

"name": "Adam",

"country: "Scotland",

"links": [

{

"href": "/user/2",

"rel": "self",

"method": "GET"

},

{

"href": "/user/2",

"rel": "edit",

"method": "PUT"

},

{

"href": "/user/2",

"rel": "delete",

"method": "DELETE"

}

]

}

],

"links": [

{

"href": "/user",

"rel": "create",

"method": "POST"

}

]

}

We can tell a lot from this response. For instance, we now know we can create a new user by POSTing to /user:

**Request**

POST /user

Accept: application/json+userdb

Content-Type: application/json+userdb

{

"name": "Karl",

"country": "Austria"

}

**Response**

201 Created

Content-Type: application/json+userdb

{

"user": {

"id": 3,

"name": "Karl",

"country": "Austria",

"links": [

{

"href": "/user/3",

"rel": "self",

"method": "GET"

},

{

"href": "/user/3",

"rel": "edit",

"method": "PUT"

},

{

"href": "/user/3",

"rel": "delete",

"method": "DELETE"

}

]

},

"links": {

"href": "/user",

"rel": "list",

"method": "GET"

}

}

We also know that we can change existing data:

**Request**

PUT /user/1

Accept: application/json+userdb

Content-Type: application/json+userdb

{

"name": "Emil",

"country": "Bhutan"

}

**Response**

200 OK

Content-Type: application/json+userdb

{

"user": {

"id": 1,

"name": "Emil",

"country": "Bhutan",

"links": [

{

"href": "/user/1",

"rel": "self",

"method": "GET"

},

{

"href": "/user/1",

"rel": "edit",

"method": "PUT"

},

{

"href": "/user/1",

"rel": "delete",

"method": "DELETE"

}

]

},

"links": {

"href": "/user",

"rel": "list",

"method": "GET"

}

}

Notice that we are using different HTTP verbs (GET, PUT, POST, DELETE etc.) to manipulate these resources, and that the only knowledge we presume on the clients part is our media definition.

Further reading:

* The many much better answers on this very page.
* [How I explained REST to my wife](http://www.looah.com/source/view/2284)~~.~~
* [How I explained REST to my wife](http://web.archive.org/web/20130116005443/http:/tomayko.com/writings/rest-to-my-wife).
* [Martin Fowler's thoughts](http://martinfowler.com/articles/richardsonMaturityModel.html)
* [Paypal's API has hypermedia controls](https://developer.paypal.com/docs/api/)

(This answer has been the subject of a fair amount of criticism for missing the point. For the most part, that has been a fair critique. What I originally described was more in line with how REST was usually implemented a few years ago when I first wrote this, rather than its true meaning. I've revised the answer to better represent the real meaning.)

Q: What do “branch”, “tag” and “trunk” mean in Subversion repositories?

Ans1:

Not sure I agree with Nick re tag being similar to a branch. A tag is just a marker

* [**Trunk**](http://svnbook.red-bean.com/en/1.8/svn.tour.importing.html#svn.tour.importing.layout) would be the main body of development, originating from the start of the project until the present.
* [**Branch**](http://svnbook.red-bean.com/en/1.8/svn.branchmerge.whatis.html) will be a copy of code derived from a certain point in the trunk that is used for applying major changes to the code while preserving the integrity of the code in the trunk. If the major changes work according to plan, they are usually merged back into the trunk.
* [**Tag**](http://svnbook.red-bean.com/en/1.8/svn.branchmerge.tags.html) will be a point in time on the trunk or a branch that you wish to preserve. The two main reasons for preservation would be that either this is a major release of the software, whether alpha, beta, RC or RTM, or this is the most stable point of the software before major revisions on the trunk were applied.

In open source projects, major branches that are not accepted into the trunk by the project stakeholders can become the bases for *forks* -- e.g., totally separate projects that share a common origin with other source code.

Ans2:

First of all, as @AndrewFinnell and @KenLiu point out, in SVN the directory names themselves mean nothing -- "trunk, branches and tags" are simply a common convention that is used by most repositories. Not all projects use all of the directories (it's reasonably common not to use "tags" at all), and in fact, nothing is stopping you from calling them anything you'd like, though breaking convention is often confusing.

I'll describe probably the most common usage scenario of branches and tags, and give an example scenario of how they are used.

* **Trunk**: The main development area. This is where your next major release of the code lives, and generally has all the newest features.
* **Branches**: Every time you release a major version, it gets a branch created. This allows you to do bug fixes and make a new release without having to release the newest - possibly unfinished or untested - features.
* **Tags**: Every time you release a version (final release, release candidates (RC), and betas) you make a tag for it. This gives you a point-in-time copy of the code as it was at that state, allowing you to go back and reproduce any bugs if necessary in a past version, or re-release a past version exactly as it was. Branches and tags in SVN are lightweight - on the server, it does not make a full copy of the files, just a marker saying "these files were copied at this revision" that only takes up a few bytes. With this in mind, you should never be concerned about creating a tag for any released code. As I said earlier, tags are often omitted and instead, a changelog or other document clarifies the revision number when a release is made.

For example, let's say you start a new project. You start working in "trunk", on what will eventually be released as version 1.0.

* **trunk/ - development version, soon to be 1.0**
* branches/ - empty

Once 1.0.0 is finished, you branch trunk into a new "1.0" branch, and create a "1.0.0" tag. Now work on what will eventually be 1.1 continues in trunk.

* trunk/ - development version, **soon to be 1.1**
* **branches/1.0 - 1.0.0 release version**
* **tags/1.0.0 - 1.0.0 release version**

You come across some bugs in the code, and fix them in trunk, and then merge the fixes over to the 1.0 branch. You can also do the opposite, and fix the bugs in the 1.0 branch and then merge them back to trunk, but commonly projects stick with merging one-way only to lessen the chance of missing something. Sometimes a bug can only be fixed in 1.0 because it is obsolete in 1.1. It doesn't really matter: you only want to make sure that you don't release 1.1 with the same bugs that have been fixed in 1.0.

* trunk/ - development version, soon to be 1.1
* branches/1.0 - **upcoming 1.0.1 release**
* tags/1.0.0 - 1.0.0 release version

Once you find enough bugs (or maybe one critical bug), you decide to do a 1.0.1 release. So you make a tag "1.0.1" from the 1.0 branch, and release the code. At this point, trunk will contain what will be 1.1, and the "1.0" branch contains 1.0.1 code. The next time you release an update to 1.0, it would be 1.0.2.

* trunk/ - development version, soon to be 1.1
* branches/1.0 - **upcoming 1.0.2 release**
* tags/1.0.0 - 1.0.0 release version
* **tags/1.0.1 - 1.0.1 release version**

Eventually you are almost ready to release 1.1, but you want to do a beta first. In this case, you likely do a "1.1" branch, and a "1.1beta1" tag. Now, work on what will be 1.2 (or 2.0 maybe) continues in trunk, but work on 1.1 continues in the "1.1" branch.

* trunk/ - development version, **soon to be 1.2**
* branches/1.0 - upcoming 1.0.2 release
* **branches/1.1 - upcoming 1.1.0 release**
* tags/1.0.0 - 1.0.0 release version
* tags/1.0.1 - 1.0.1 release version
* **tags/1.1beta1 - 1.1 beta 1 release version**

Once you release 1.1 final, you do a "1.1" tag from the "1.1" branch.

You can also continue to maintain 1.0 if you'd like, porting bug fixes between all three branches (1.0, 1.1, and trunk). The important takeaway is that for every main version of the software you are maintaining, you have a branch that contains the latest version of code for that version.

Another use of branches is for features. This is where you branch trunk (or one of your release branches) and work on a new feature in isolation. Once the feature is completed, you merge it back in and remove the branch.

* trunk/ - development version, soon to be 1.2
* branches/1.1 - upcoming 1.1.0 release
* **branches/ui-rewrite - experimental feature branch**

The idea of this is when you're working on something disruptive (that would hold up or interfere with other people from doing their work), something experimental (that may not even make it in), or possibly just something that takes a long time (and you're afraid if it holding up a 1.2 release when you're ready to branch 1.2 from trunk), you can do it in isolation in branch. Generally you keep it up to date with trunk by merging changes into it all the time, which makes it easier to re-integrate (merge back to trunk) when you're finished.

Also note, the versioning scheme I used here is just one of many. Some teams would do bug fix/maintenance releases as 1.1, 1.2, etc., and major changes as 1.x, 2.x, etc. The usage here is the same, but you may name the branch "1" or "1.x" instead of "1.0" or "1.0.x". (Aside, [semantic versioning](http://semver.org/) is a good guide on how to do version numbers).

# Q: [How to understand the “synchronous” and “asynchronouns” messaging in JMS?](https://stackoverflow.com/questions/22088873/how-to-understand-the-synchronous-and-asynchronouns-messaging-in-jms)

After reading some document of JMS, I totally puzzled by the phrase synchronous and asynchronouns.

See this page: <http://docs.oracle.com/cd/E19798-01/821-1841/bncdq/index.html>

**Synchronous**

You use the receive method to consume a message synchronously. You can use this method at any time after you call the start method:

connection.start();

Message m = consumer.receive();

connection.start();

Message m = consumer.receive(1000); // time out after a second

To consume a message asynchronously, you use a message listener, described in the next section.

**Asynchronous**

JMS Message Listeners A message listener is an object that acts as an asynchronous event handler for messages. This object implements the MessageListener interface, which contains one method, onMessage. In the onMessage method, you define the actions to be taken when a message arrives.

You register the message listener with a specific MessageConsumer by using the setMessageListener method. For example, if you define a class named Listener that implements the MessageListener interface, you can register the message listener as follows:

Listener myListener = new Listener();

consumer.setMessageListener(myListener);

I have two questions:

1. As what I understood, the nature of JMS is asynchronous. Producer publishes messages to the queue/topic, it doesn't need to wait consumer. This is asynchronous behaviour. How can it be "synchronous"?
2. If the "mesageListener" is asynchronous, but in my test with spring-jms, I found it always running in a thread. That means, if I write Thread.sleep(2000) in onMessage, it have to be wait 2 seconds before processing next message. Is it "asynchronous"?

Ans:

If you understand it better like this, consumer.receive() uses a pull model: you read from a queue and are blocked waiting for this message until it comes, or some timeout has elapsed.

Using a listener uses a push model: you register a listener and, when a message comes in, the listener is called, in a separate thread.

Everything is done in a thread in Java, and the listener call is no exception. Whether the listener message handling prevents the processing of other messages in the queue depends on how many threads are dedicated to message processing. If you configure Spring to use a pool of 5 threads to process messages asynchronously, then 5 listeners will be able to process messages in parallel.

Q: How are “mvn clean package” and “mvn clean install” different?

|  |  |
| --- | --- |
| test | run tests using a suitable unit testing framework.  These tests should not require the code be packaged or deployed. |

|  |  |
| --- | --- |
| package | take the compiled code and package it in its distributable format, such as a JAR. |
| install | install the package into the local repository, for use as a dependency in other projects locally. |
| deploy | done in an integration or release environment,  copies the final package to the remote repository for sharing with other developers and projects. |

Well, both will clean. That means they'll remove the target folder. The real question is what's the difference between package and install?

package will compile your code and also package it. For example, if your pom says the project is a jar, it will create a jar for you when you package it and put it somewhere in the target directory (by default).

install will compile and package, but it will also put the package in your local repository. This will make it so other projects can refer to it and grab it from your local repository.

#### A Build Lifecycle is Made Up of Phases

Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

For example, the default lifecycle comprises of the following phases (for a complete list of the lifecycle phases, refer to the [Lifecycle Reference](http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference)):

* validate - validate the project is correct and all necessary information is available
* compile - compile the source code of the project
* test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
* package - take the compiled code and package it in its distributable format, such as a JAR.
* verify - run any checks on results of integration tests to ensure quality criteria are met
* install - install the package into the local repository, for use as a dependency in other projects locally
* deploy - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

These lifecycle phases (plus the other lifecycle phases not shown here) are executed sequentially to complete the default lifecycle. Given the lifecycle phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the integration tests, install the verified package to the local repository, then deploy the installed package to a remote repository.

Q: Advantage of Spring Boot?

Ans:

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run". We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.

## Features

* Create stand-alone Spring applications
* Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
* Provide opinionated 'starter' POMs to simplify your Maven configuration
* Automatically configure Spring whenever possible
* Provide production-ready features such as metrics, health checks and externalized configuration
* Absolutely **no code generation** and **no requirement for XML** configuration

Q: What is the difference between window.onload and document.ready()

What are the differences between JavaScript's [window.onload](https://developer.mozilla.org/en/docs/Web/API/GlobalEventHandlers/onload) and jQuery's [$(document).ready()](https://api.jquery.com/ready/)method?

The ready event occurs after the HTML document has been loaded, while the onload event occurs later, when all content (e.g. images) also has been loaded.

The onload event is a standard event in the DOM, while the ready event is specific to jQuery. The purpose of the ready event is that it should occur as early as possible after the document has loaded, so that code that adds functionality to the elements in the page doesn't have to wait for all content to load.

$(document).ready(function() {

// Executes when the HTML document is loaded and the DOM is ready

alert("Document is ready");

});

// .load() method deprecated from jQuery 1.8 onward

$(window).on("load", function() {

// Executes when complete page is fully loaded, including

// all frames, objects and images

alert("Window is loaded");

});

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>