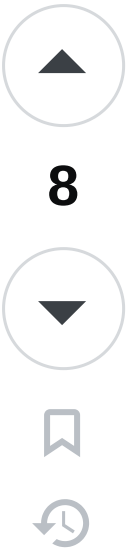
Isn't resource-oriented really object-oriented?

Asked 16 years, 2 months ago Modified 16 years, 2 months ago

Viewed 5k times



When you think about it, doesn't the REST paradigm of being resource-oriented boil down to being object-oriented (with constrained functionality, leveraging HTTP as much as possible)?

I'm not necessarily saying it's a bad thing, but rather that if they are ~~essentially the same~~ very similar then it becomes much easier to understand REST and the implications that such an architecture entails.

Update: Here are more specific details:

1. REST resources are equivalent to public classes. Private classes/resources are simply not exposed.
2. Resource state is equivalent to class public methods or fields. Private methods/fields/state is simply not exposed (this doesn't mean it's not there).
3. While it is certainly true that REST does not retain client-specific state across requests, it **does** retain resource state across all clients. Resources **have** state, the same way classes have state.

4. REST resources are are globally uniquely identified by a URI in the same way that server objects are globally uniquely identified by their database address, table name and primary key. Granted there isn't (yet) a URI to represent this, but you can easily construct one.

rest oop

Share

Improve this question

Follow

edited Oct 8, 2008 at 17:27

community wiki
5 revs, 3 users 100%
Gili

8 Answers

Sorted by: Highest score (default)

24

REST is similar to OO in that they both model the world as entities that accept messages (i.e., methods) but beyond that they're different.

Object orientation emphasizes encapsulation of state and **opacity**, using as many different methods necessary to operate on the state. REST is about transfer of (representation of) state and **transparency**. The number of methods used in REST is constrained and uniform across *all* resources. The closest to that in OOP is the

`ToString()` method which is very roughly equivalent to an HTTP GET.

Object orientation is **stateful**--you refer to an object and can call methods on it while maintaining state within a session where the object is still in scope. REST is **stateless**--everything you want to do with a resource is specified in a single message and all you ever need to know regarding that message is sent back in a single response.

In object-orientation, **there is no concept of universal object identity**--objects either get identity from their memory address at any particular moment, a framework-specific UUID, or from a database key. In REST **all resources are identified with a URI** and don't need to be instantiated or disposed--they always exist in the cloud unless the server responds with a *404 Not Found* or *410 Gone*, in whch case you know there's no resource with that URI.

REST has guarantees of **safety** (e.g., a GET message won't change state) and **idempotence** (e.g., a PUT request sent multiple times has same effect as just one time). Although some guidelines for particular object-oriented technologies have something to say about how certain constructs affect state, there really isn't anything about object orientation that says anything about safety and idempotence.

 edited Oct 3, 2008 at 7:41

Follow

answered Sep 30, 2008 at 14:13

Mark Cidade

99.8k ● 33 ● 229 ● 237

The only thing I would add to your excellent explanation is that REST is naturally a distributed architecture whereas object-oriented is not. – Darrel Miller Sep 30, 2008 at 15:07

marxidad, I've updated my question based on your answer. Please let me know what you think. – Gili Oct 2, 2008 at 5:16

I agree with those particular simularities between REST and OO but I think that saying that "they are essentially the same" is a bit of stretch. Understanding object orientation alone is hardly enough to realize what a REST-based architecture implies. – Mark Cidade Oct 3, 2008 at 7:35

I disagree with most of your answer. REST/HTTP has opacity as well: You can only see or change what the server wants you to. HTTP objects have obvious state and object identity (these are really only two sides to the same thing): A URI uniquely identifies a given resource. The only thing that's stateless is "the connection/session", and that's only true for "REST the ideal", not typically in practice. Many OOP have also "const" methods which is like GET. – Jo So Jun 13, 2017 at 18:25

You seem to concentrate on the "connection" which would be the message *passing* in OOP. If you think about the resources managed through HTTP / a RESTful API instead, it's easy to see it's very object-oriented. – Jo So Jun 13, 2017 at 18:27

2

I think there's a difference between saying a concept can be expressed in terms of objects and saying the concept is the *same* as object orientation.

OO offers a way to describe REST concepts. That doesn't mean REST itself implements OO.

Share Improve this answer

Follow

answered Sep 30, 2008 at 14:33

catfood

4,331 ● 5 ● 32 ● 57

1

You are right. Dan Connolly wrote an article about it in 1997. The Fielding thesis also talks about it.

Share Improve this answer

Follow

answered Sep 30, 2008 at 12:49

community wiki

JeroenH

> Though very short, this should be the accepted answer.
> – Jo So Jun 13, 2017 at 18:29

1

Objects bundle state and function together. Resource-orientation is about explicitly modeling state(data), limiting function to predefined verbs with universal semantics (In the case of HTTP, GET/PUT/POST/DELETE), and leaving the rest of the processing to the client.

There is no equivalent for these concepts in the object-orientation world.

Share Improve this answer

Follow

answered Oct 6, 2008 at 23:29

Alexandros Marinos

1,396 ● 2 ● 15 ● 25

0

Only if your objects are DTOs (Data Transfer Objects) - since you can't really have behavior other than persistence.

Share Improve this answer

Follow

answered Sep 30, 2008 at 12:47

Mark Brackett

85.6k ● 17 ● 111 ● 155

0

Yes, your parallel to object-orientation is correct.

The thing is, most webservices (REST, RESTful, SOAP,..) can pass information in the form of objects, so that isn't what makes it different. SOAP tends to lead to fewer services with more methods. REST tends to lead to more services (1 per resource type) with a few calls each.

Share Improve this answer

Follow

answered Sep 30, 2008 at 12:49

Steve g

2,489 ● 17 ● 17

0

Yes, REST is about transfer of objects. But it isn't the whole object; just the object's current state. The implicit assumption is that the class definitions on both sides of the REST are potentially similar; otherwise the object state has been coerced into some new object.

REST only cares about 4 events in the life on an object, create (POST), retrieve (GET), update (PUT) and delete. They're significant events, but there's only these four.

An object can participate in lots of other events with lots of other objects. All the rest of this behavior is completely outside the REST approach.

There's a close relationship -- REST moves Objects -- but saying they're the same reduces your objects to passive collections of bits with no methods.

Share Improve this answer

Follow

answered Sep 30, 2008 at 13:00

S.Lott

391k ● 82 ● 517 ● 788

0

REST is not just about objects, its also about properties :: a post request to /users/john/phone_number with a new phone number is not adding a new object, its setting a property of the user object 'john'

This is not even the whole state of the object, but only a change to a small part of the state.

It's certainly not a 1:1 match.

Share Improve this answer

Follow

answered Sep 30, 2008 at 14:08

garrow

3,489 1 23 24

Your example would be a PUT. But you can also POST to /users/john, with an arbitrary command (OOP method call) in the body. – Jo So Jun 13, 2017 at 18:30