Paul Susmarski

3/7/2012

1. They’re very similar.  POST and PUT affect changes on the server and allow for creation and updates.  Put can handle explicitly named URL objects, whereas post will handle names server side.  POST allows for multiple modifications to a single URL simultaneously.  
   A PUT request idempotent, where you can do it as many times as you like and the result will always be the same.
2. Relative
3. An absolute URL contains the full domain name and protocol and the final file you are trying to reach on the server. ex: <http://google.com/foo/bar.txt> Whereas relative URLs are paths to the file/resource that is relative to the current domain in which they reside, continuing with my example, the relative path for bar.txt from [google.com](http://google.com/) would be /foo/bar.txt.
4. GET request, tricky, there is no method attribute for <a> tag
5. There is a query and it is "request\_type=PUT"
6. This is missing the “title” attribute so that the user could get a clearer message of what is going to happen when they clicked the link.  This also increases accessibility for all users by making it clear what happens.
7. The database acts as the provider of information for the application, allowing data to be persisted and maintained across sessions.  The web browser acts as a parser converting data and code to displayable information, and allows for interaction with the application for a user.
8. Yes, we would generally see a body in this type of HTTP response.  A 200 OK response must include a message body, however the length of that body may be 0.

class Troll

attr\_accesssor :ugliness, :smelliness, :strength

def initialize(words = “UNGAH”)

@grunt = words

end

def speak

42.times do

puts @grunt

end

end

def reverse

puts @grunt.reverse

end

self.propogate

Troll.new(“eegah”)

end

end

1. A method called  “fight”, since “respond\_to” is asking whether or not the class has a method called fight.  This method might take another troll as a parameter and compare their strengths and display a winner based on the higher score. Most generally we are missing a function definition like this:  
   def fight  
   end
2. “respond\_to” illustrates object-oriented polymorphism by the fact that all classes inherit the Object class.  So no matter what the class you create is (trains or trolls) you can always call the method “respond\_to” on that class.  There are numerous functions for the Object class in Ruby, which includes the “respond\_to” method.
3. A boolean value (true/false)
4. The non bang method will return a new value that has been modified, whereas the bang method call will modify the caller itself.  Example:  
   s = “blah”  
   s.capitalize  
   puts s #s will still be blah  
   s.capitalize! #s will now be “Blah”
5. Ruby has a dynamic type system, referred to as “Duck Typing”, which means that an object’s current methods and properties determines its current type, this is instead of its inheritance or interfaces that may be included in that class.  The idea being from James Whitcomb Riley when he said: “When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.”
6. An array of strings.
7. Yes, you can use either {} or do/end to designate blocks of code.
8. No, a Ruby function will always return the value of the last expression that is evaluated within that function.
9. Callbacks: before\_save, before\_validation\_on\_update, after\_update, before\_update

|  |  |  |
| --- | --- | --- |
| HTTP method | controller action | CRUD operation |
| GET | index | read |
| GET | new | read |
| POST | create | create |
| GET | edit | read |
| PUT | update | update |
| DELETE | delete | delete |

1. Rail simulates PUT and DELETE requests because most browsers don’t support such requests.  They generally only support POST and GET requests.
2. The development environment is for your local developing purposes, whereas the production environment is for the real-world and what the client/users will eventually see.
3. So we should use the singular name when there is only going to be one instance of that model (perhaps even when there is no model), or when there is going to be a single controller.  This should be the case when you plan to only have one record of the thing on hand.  Really it’s about being consistent.
4. A helper method is a method defined in the the associated \*\*\*\_helper.rb file, and the methods are used in the view.  It is used to help remove logic and code that may be repeated in the views consistently, therefore enforcing DRY. Also they are used to remove any logic that may appear in your view to help make the view purely declarative.
5. You’ll need to make a join table between flowers and bees that would be called bees\_flowers with a bee\_id and flower\_id column.
6. No, this is not okay, as we can use eager loading to significantly reduce the amount of SQL queries.  This would improve performance.  To do this you do something like this in your controller:  
   @flower = Flower.find(params[:id], :include => :bees  
   However, don’t fix what’s not broken ;).