Richie Zirbes
Midterm Exam (the correct one)
Mar 6, 2012
Due March 10

Architecture

1) What is the difference between an HTTP PUT request and a POST request?

The defined HTTP PUT request is for creating a resource [replace]
The defined HTTP POST request is for modifying existing resources [update]
The main difference is that the PUT request is idempotent, meaning that applying the same request multiple times does not cause duplication or other side effects, while the POST request may have side effects for multiple attempts.
However, this is the HTTP definition only, and the *actual* uses of these requests depends on the implementer. In many applications, HTTP is mainly only used for it's GET and POST requests.

You see a link in an interface whose markup is as follows:
```
<a href="images/new?request_type=PUT" method="POST">create a new image</a>
```
2) Is the target URL relative or absolute?

This target is relative.

3) What is the difference between an absolute and relative URL?

An absolute URL references a resource fully, so there is no dispute on where the resource is located.
A relative URL references a resource in relation to another resource (like the current directory).

4) If you clicked on this link, what kind of request would your browser generate? (which HTTP method?) Assume no JavaScript modifies the behavior of the link.

The anchor 'a' tag has no defined 'method' attribute. The action of clicking an anchor tag is to generate a GET request of the specified hypertext reference.

5) Is there a querystring? What is it?

The querystring is everything after the '?' character: 'request_type=PUT'

6) What is lacking from the link declaration that would otherwise enhance accessibility?

In software terms, accessing the anchor 'link' element can be done through the Document Object Model. This is made easier if the element is given a unique id, otherwise access would be through indirect methods such as parsing through the object hierarchy, making an application much more rigid/brittle.

7) What are the roles of the database and Web browser in most Web applications? (One sentence for each.)

Databases are used in general for organized storage of data. However, compared to sifting through text/byte files, databases are optimized for insertion and retrieval of data (and provide many other desirable operations). The types of data used in web applications may be user-oriented data such as preferences or identification hashes, or more application-based data such as locations of files and content.

Web Browsers are the platform for web applications. They interpret the given resources (HTML, images, CSS, etc.) and act as the interface to users (on the client side).

Given the following HTTP response header:
```
HTTP/1.1 200 Ok
```

```
Date: Wed, 09 Mar 2011 16:43:33 GMT
Server: Apache
Connection: Keep-Alive
Keep-Alive: timeout=2, max=100
Etag: "110e412f-7df-49e0f6a106500"
Vary: Accept-Encoding
```

8) Would an HTTP response that begins like that usually contain a body? Why or why not?

HTTP responses in the 200s are confirmations of previous requests. A 200 OK response may or may not contain a body depending on the client request. However, a body is likely to be present, as clients typically request information and pages (GET, POST).

Ruby

9) Write a Ruby class definition that meets the following criteria:

- *class is called Troll*
- *class has publicly accessible attributes ugliness, smelliness, and strength*
- *upon instantiation, an object of this class has a member variable, a String, called grunt, whose initial value is "UNGAH" (that's pronounce "oon-guh").*
- *class has an instance method called speak() that prints the value of the instance variable grunt 42 times*
- *class has an instance method called reverse() that prints the value of the instance variable grunt backwards*
- *class has a static/class method called propagate(), which returns a Troll instance whose grunt attribute is "eegah"*

```
Troll.rb:
#!/usr/bin/ruby
class Troll
      # attributes:
      attr_accessor :ugliness, :smelliness, :strength
      # class methods:
      def self.propagate
            troll = Troll.new "eegah"
      end
      # instance methods:
      def initialize(gr=nil)# constructor
            if gr
                  @grunt = gr
            else
                  @grunt = "UNGAH"
            end
      end
      def speak # speak 40 times
            (1..40).each do |i|
                  puts @grunt
            end
      end
      def reverse # reverse of grunt
            puts @grunt.reverse
      end
end
```

10) Imagine a Troll instance fred, which, when the following method is called: fred.respond_to?("fight") returns true. What is missing from your class definition in order for this example to be accurate?

A definition for the 'fight' method must be declared in order for fred.respond_to?("fight") to return true.

```
      def fight
            # do something
      end
```

11) Does the respond_to?() method illustrate object-oriented polymorphism? If so, in what manner?

All classes in the ruby hierarchy are derived from the 'Object' base class. As such, all methods available to an Object are available to sub-classes (who can then override or remove them if desired). This allows a consistent interface to objects, and default commonly used methods.

12) According to Ruby conventions, what kind of value would you expect to receive from a method that ends in a question mark (?) ?

Methods with a '?' are conventionally used to return a boolean value, or more specifically an object of the 'TrueClass' or 'FalseClass'.

13) According to Ruby conventions, what is the difference between pairs of methods like do_this and do_this! (notice the bang)?

With two methods of the same name, differing only in the presence of the exclamation (!) character, the conventional difference is in the separate behaviors:
no exclamation: a modified object based on the caller object is returned, while the called object itself remains unchanged
exclamation: the object itself is modified (and returned)

Another, also conventional, difference is that the exclamation version of the method will propagate errors if they come up, while the non-exclamation version may just handle the error internally or ignore it.

14) Briefly explain Ruby's type system. What is it (by name)? What does it mean?

Ruby is a dynamically typed language. This means that symbols can be used to instantiated objects of one type then later used to reference an object of a different type.

15) What type of Ruby object does the following expression yield? %w(master rails and then try another framework you'll never go back)

'*%w(master rails and then try another framework you'll never go back)*' returns an Array object with String elements, as separated by the space ( ) character.

16) Given an array of strings called @happy_places, would these two snippets of code do the same thing?
```
@happy_places.each do |happy_place|
      puts happy_place
end
```
and
```
  @happy_places.each {|hp| puts hp}
```

Yes. These code snippets do the same thing, printing out the elements of the '@happy_places' array.

17) Given a function that needs to return a value to its caller, does the function need an explicit return statement? If so, explain why. If not, then what can you always expect a Ruby function to return?

If the 'return' word is explicitly used, then the subsequent expression will be returned. However, the return statement is not required, and in its absence the result of the most recent previous expression is returned (A method with no content returns nil).

Rails

18) Name four ActiveRecord callbacks that you can bind methods to.

Callbacks are a way to combine default Rails behavior along with custom code - a way to insert actions into critical points in an object's life-cycle. Although there are many, some examples are:
before_validation, before_destroy, after_create, after_save

19) The Rails convention maps HTTP methods to certain controller methods, and those methods usually involve specific CRUD operations on models. Given the following CRUD database methods:
create, read, update, and delete
and the following HTTP methods:
GET, PUT, POST, DELETE
and the following controller actions:
index, new, create, edit, update, destroy

Complete the following table.

HTTP Method	Controller Action	CRUD Operation
GET	*index/show*	Read
GET	*new*	Read
POST	*create*	Create
GET	*edit*	Update
PUT	*update*	Update
DELETE	*destroy*	Delete

20) Rails "simulates" PUT and DELETE requests. Why?

PUT and DELETE are old-school HTTP request methods that support for can no longer be depended on. The reason for 'simulation' is because as HTTP originally planned, it is good, clear, coding practice. This is done through internal variables representing the different states (PUT, DELETE, etc.)

21) What is the difference between the two Rails environments 'production' and 'development' ?

Nothing. Except - the production and development environments have different purposes and thus are programmed using different configurations. The development environment is used for ongoing programming and coding such that feedback and errors are displayed in highly visible manners which a developer appreciates as this information is used to discover and fix problems as soon as possible. The production environment is intended for real-world use of the application, implying more elegance in error handling and higher performance.

22) Usually, Rails controllers incorporate plural nouns, such as ProtestsController and RevolutionsController. In what case should a controller have a singular name like GeocodingController?

A Singular name is used for a controller when it is only ever used on non-specific objects. It may make database references, but makes no use of concepts such as ids and as such a pluralization of it would never make sense.

23) What is a Rails "helper method" and when should they be defined and used by you, the developer?

Helper methods are used in DRY coding, and removing logic from views. They should be used anywhere in place of code that is duplicated or similar to other code.

Assume you have a Flower AR class that has_and_belongs_to_many :bees, and a Bee class that has_and_belongs_to_many :flowers.
24) What must exist in the database schema in order for AR to infer the proper foreign key / relationship?

The 'has_and_belongs_to_many' association is a many-to-many relationship. Rails thus expects the database to contain a join table of the name 'bees_flowers' with columns 'bee_id' and 'flower_id'.

Assume that a Bee :belongs_to a Hive and a Hive has_many :bees. Also assume a GET request is sent to the FlowersController#show action, which contains a finder method call @flower = Flower.find(params[:id]). Assume the view app/views/flowers/show.html.haml displays the name of the Flower and each Flower's bee's name and Hive name like so:

```
- @flower.bees.each do |b|
      %h1= b.name
      %p= b.hive.name
```

If you were tailing the log of your application during the rendering of the response, you would notice tons of database queries.
25) Are all of those queries ok? If so, explain why. If not, explain how you would reduce the number of database queries (without hand-rolling your own SQL query).

When more and more queries are requested, the database can become the bottleneck, and performance suffers. When possible, queries should be consolidated into fewer or single queries to recover from this problem. In this example, a way around this is to load the secondary queries (for each bee) all at once, and perhaps also along with the original query (for the single flower).

```
- @flower = Flower.find(@flower.id,:include => :bees)
- @flower.bees.each do |b|
      %h1= b.name
      %p= b.hive.name
```