factory_girl tutorial

http://github.com/igal/factory_girl_tutorial

Igal Koshevoy, Pragmaticraft
Business-Technology Consultant
igal@pragmaticraft.com
@igalko on Twitter & Identi.ca

What's a fixture?

"A test fixture is something used to consistently test some item, device, or piece of software."

In Ruby on Rails, this is a baked-in way to create model records with sample data for tests.

E.g., a `Kitten` model has a fixture identified as `shiva` in the file `spec/fixtures/kittens.yml`:

```
shiva:  # A fixture identified as "shiva"
  legs:  6  # An attribute "legs" set to 6
  name:  "Shiva The Destroyer
```

How are fixtures used?

```
describe Kitten do
  fixtures :kittens

  it "should have a number of legs" do
    kitten = kittens(:shiva)
    kitten.legs.should be_a_kind_of(Fixnum)
  end
end
```

What’s wrong with fixtures?

- State is **distant** from testing code
- Difficult to make **variants** of state
- **Intertwined** states
- **Slow** insertion and manipulation
- **Brittle** due to unexpected dependencies

Ways to provide sample data:

1. **Fixtures**
2. **Real models**: Setup records manually in test.
3. **Stubs**: Real models (e.g., `Kitten`) with some methods returning fake results, e.g. `stub!`
4. **Expectations**: Real models with some methods expecting to return fake results, e.g., `should_receive`
5. **Mock models**: Fake objects pretending to be real models returning fake results, e.g. `mock_model`
6. **Stub models**: Real models with some methods returning fake results and no DB, e.g. `stub_model`.
7. **Factories**: Real models produced by factory method calls, may be instantiated, saved or stubbed.

A simple, custom factory:

```
describe Kitten do
  def create_kitten(name=nil, legs=nil)
    @name ||= 0
    name ||= "kitten-#{@name += 1}"
    legs ||= 4
    return Kitten.create(:name => name, :legs => legs)
  end

  it "should have a number of legs" do
    kitten = create_kitten("Shiva The Destroyer", 6)
    kitten.legs.should be_a_kind_of(Fixnum)
  end
end
```

Wish list for factories:

- Easily make objects from sensible **defaults**
- Easily **override** the given defaults
- Easily use **sequences** to generate unique values
- Easily **derive** values from other values
- Easily define simple and complex **associations**
- Easy, concise, yet comprehensive **API**
- Supports favorite **ORMs**, e.g. ActiveRecord
- Supports favorite **databases**, e.g. PostgreSQL
- Supports favorite **data stores**, e.g. MongoDB
- Supports favorite **test frameworks**, e.g. RSpec

What's factory_girl?

- **thoughtbot**'s factory library for ActiveRecord API and supported databases, works great in **RSpec**, **Shoulda** and **Test::Unit**.
- Easily create objects from **defaults** and **override**
- Easily define **sequences** and **derived** values
- Easily set simple **associations** – but complex associations can be tricky
- Easy, concise, and fairly comprehensive **API** with callbacks

Defining a factory_girl factory:

```
# A factory named "kitten" to create Kitten records
Factory.define :kitten do |f|
  # Make records with 4 legs by default
  f.legs 4
  # Name records "kitten-1", "kitten-2", etc.
  f.sequence(:name) { |n| "kitten-#{n}" }
  # Derive an attribute using a method call
  f.description { |r| "A kitten with #{r.legs} legs" }
end
```

Using a factory_girl factory:

```
describe Kitten do
  it "should have a number of legs" do
    kitten = Factory(:kitten)
    kitten.legs.should be_a_kind_of(Fixnum)
  end

  it "should have the expected number of legs" do
    # Override the default number of legs
    kitten = Factory(:kitten, :legs => 6)
    kitten.legs.should == 6
  end
end
```

Defining belongs_to associations and inheriting:

```
Factory.define(:toy) do |f|
  f.sequence(:name) { |n| "toy-#{n}" }
end

Factory.define(:toy_with_kitten, \
    :parent => :toy) do |f|
  # Create a kitten for this toy
  f.association(:kitten)
end
```

Defining has_many associations:

```
Factory.define :kitten_with_toys, :parent => :kitten do |f|
  # Toys to create via :has_many association
  f.toys do |r|
    [
      r.association(:toy, :name => "Fragile Vase"),
      r.association(:toy, :name => "My Leg")
    ]
  end
end
```

Using & overriding associations

```
describe Toy do
  it "should know owner's name" do
    # The Kitten association is automatically created
    toy = Factory(:toy_with_kitten)
    toy.kitten.name.should_not be_blank
  end

  it "should know owner's specific name" do
    # Override the default set of toys
    kitten = Factory(:kitten_with_toys, :toys => [], \
      :name => "Shiva")
    # Override the default kitten
    toy = Factory(:toy, :kitten => kitten)
    toy.kitten_name.should == kitten.name
  end
end
```

Callbacks

```
Factory.define :kitten_with_toys_and_description_callback, \
    :parent => :kitten_with_toys do |f|
  f.after_build do |r|
    r.description = "A kitten with #{r.legs} legs" \
      + " and #{r.toys.count} toys"
  end
end

describe Kitten
  it "should have a description set by a callback" do
    kitten = Factory(:kitten_with_toys_and_description_callback)
    kitten.description.should =~ /A kitten with \d+ legs and \d+ toys/
  end
end
```

Custom factory + factory_girl

```
def create_kitten_with_description(name, &block)
  kitten = Factory(:kitten, :name => name)
  kitten.description = block.call(kitten)
  return kitten
end

it "should set description" do
  kitten = create_kitten_with_description("Shiva") do |k|
    k.name.size.to_s
  end
  kitten.description.should == "5"
end
```

Conclusion

- There are many alternatives to fixtures, each have their own pros/cons
- Factories can make writing tests/specs easier and faster in some cases
- Explore **factory_girl**, **machinist**, and **object_daddy** for your factory needs