While we're dealing with the views, we also need to arrange for our application to send the JavaScript libraries used by Rails to the user's browser. We'll talk more about this in Chapter 23, *The Web, V2.0*, on page 523, but for now let's just add a call to javascript_include_tag to the <*head*> section of the store layout.

Download depot_l/app/views/layouts/store.rhtml

```
<html>
<head>
  <title>Pragprog Books Online Store</title>
  <%= stylesheet_link_tag "depot", :media => "all" %>
  <%= javascript_include_tag :defaults %>
</head>
```

So far, we've arranged for the browser to send an AJAX request to our application. The next step is to have the application return a response. The plan is to create the updated HTML fragment that represents the cart and to have the browser stick that HTML into the DOM[3] as a replacement for the cart that's already there. The first change is to stop the add_to_cart action redirecting to the index display. (I know, we just added that only a few pages back. Now we're taking it out again. We're agile, right?)

Download depot_l/app/controllers/store_controller.rb

```
def add_to_cart
  begin
    product = Product.find(params[:id])
  rescue ActiveRecord::RecordNotFound
    logger.error("Attempt to access invalid product #{params[:id]}")
    redirect_to_index("Invalid product")
  else
    @cart = find_cart
    @cart.add_product(product)
  end
end
```

Because of this change, when add_to_cart finishes handling the AJAX request, Rails will look for an add_to_cart template to render. We deleted the old .rhtml template back on page 129, so it looks like we'll need to add something back in. Let's do something a little bit different.

Rails supports RJS templates—the *JS* stands for JavaScript. An .rjs template is a way of getting JavaScript on the browser to do what you want, all by writing server-side Ruby code. Let's write our first: add_to_cart.rjs. It goes in the app/views/store directory, just like any other template.

Download depot_l/app/views/store/add_to_cart.rjs

```
page.replace_html("cart", :partial => "cart", :object => @cart)
```

3. The Document Object Model. This is the browser's internal representation of the structure and content of the document being displayed. By manipulating the DOM, we cause the display to change in front of the user's eyes.