```
lass Movie < ActiveRecord::Base
 has_many :reviews
nd
lass Review < ActiveRecord::Base
 belongs_to :movie
nd
```

"The foreign key *belongs to* me"

reviews table gets a *foreign key* (FK) field that has primary key of Movie a review is for

Dereference movie.reviews == perform database join *(lazily)* to find reviews where movie_id == movie.id

Dereference review.movie == look up the *one* movie whose PK id == review.movie_id

add a one-to-many association:

Add has_many to *owning* model and belongs_to to *owned* model

Create migration to add foreign key to *owned side* that references *owning side*

Apply migration

rake db:test:prepare to regenerate test database schema

moviegoer: has_many :reviews
has_many :movies, :through => :reviews

movie: has_many :reviews
has_many :moviegoers, :through => :reviews

reviews: belongs_to :moviegoer
belongs_to :movie

in config/routes.rb:

```
resources :movies
```

becomes

```
resources :movies do
  resources :reviews
end
```

Nested Route: access reviews by going "through" a movie

Associations are part of *application architecture*

- provides high-level, reusable association constructs that manipulate RDBMS foreign keys
- Mix-ins allow Associations mechanisms to work with any ActiveRecord subclass

Proxy methods provide Enumerable-like behaviors

- A many-fold association quacks like an Enumerable
- Proxy methods are an example of a *design pattern*

hat if we delete a movie with reviews?

movie_id field of those reviews then refers to *nonexistent* primary key

another reason primary keys are *never* recycled

arious possibilities depending on app...

delete those reviews?
has_many :reviews, :dependent => :destroy

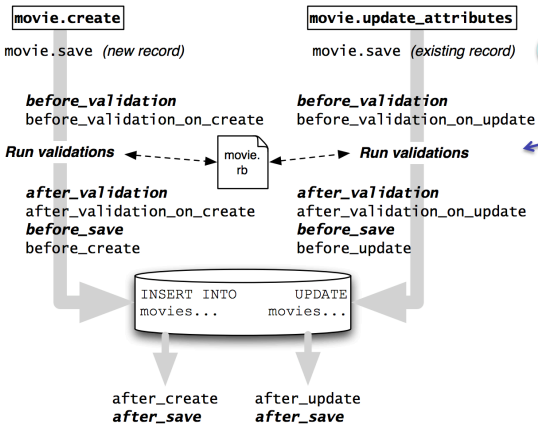
make reviews "orphaned"? (no owner)
has_many :reviews, :dependent => :nullify

an also use *lifecycle callbacks* to do other nas (eg. merging)

	Validation	Filter
Advice (DRYness)	Check invariants on model	Check conditions for allowing controller action to run
Pointcut	AR model lifecycle hooks	Before and/or after a public controller method
Can change execution flow?	No	Yes
Can define *advice* in arbitrary function?	Yes; shortcuts provided for common cases	Yes, must provide function
Info about errors?	Each model object has associated errors object	Capture in flash[], session[], or instan variable

Con: Can make code harder to debug

- *Advice* is a specific piece of code that implements a cross-cutting concern
- *Pointcuts* are the places you want to "inject" advice at runtime
- Advice+Pointcut = Aspect
- Goal: DRY out your code



- Aspect-oriented programming is a way of DRYing out cross-cutting concerns
- Ruby doesn't have fully-general AOP, but Rails provides some "predefined" pointcuts
  - Validations check or assert pre/post conditions at key points during model lifecycle
  - Controller filters check or assert pre/post conditions related to controller actions
  - and can change control flow (redirect, render)

Building block: *tamper-evident secure token*

Using cryptography, I create a string that:

- Only I can decrypt (decode)
- I can detect if it's been tampered with
- No one else could have created it without knowing my secret key

Usually, string just contains a "handle" to valuable info that I store myself

- Receive string => I know I can "trust" the handle

- Model *session* as its own entity
  - *session controller* creates and deletes session, handles interaction with auth provider
- Once user is authenticated, we need a local *users* model to represent him/her
  - session[] remembers primary key (ID) of "currently authenticated user"

Report
Reproduce and/or **R**eclassify
Regression test
Repair
Release the fix (commit and/or deploy)

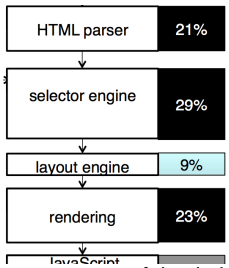
Browser *requests* web resource (URI) using HTTP

- HTTP is a simple request-reply protocol that relies on TCP/IP
- In SaaS, most URI's cause a program to be run, rather than a static file to be fetched

HTML is used to encode content, *CSS* to style it visually

Cookies allow server to track client

- Browser automatically passes cookie to server on each request
- Server may change cookie on each response
- Typical usage: cookie includes a *handle* to server-side information
- That's why some sites don't work if cookies are completely disabled



roblem: *a* depends on *b,* but *b*
terface & implementation can
hange, even if *functionality* stable
olution: "inject" an *abstract*
terface that *a* & *b* depend on
- If not exact match, Adapter/Façade
- "inversion": now b (and a) depend on interface, vs. a depending on b
uby equivalent: Extract Module
isolate the interface

Design patterns represent *successful solutions* to classes of problems

- Reuse of design rather than code/classes
- A few patterns "reified" in Rails since useful to SaaS

Can apply at many levels: architecture, design (GoF patterns), computation

Separate what changes from what stays the same

- program to interface, not implementation
- prefer composition over inheritance
- delegate!
- all 3 are made easier by duck typing

Architectural ("macroscale") patterns

- Model-view-controller
- Pipe & Filter (e.g. compiler, Unix pipeline)
- Event-based (e.g. interactive game)
- Layering (e.g. SaaS technology stack)

Computation patterns

- Fast Fourier transform
- Structured & unstructured grids
- Dense linear algebra
- Sparse linear algebra

interaction the user can do is *controller action*
l that handles that interaction
<HTTP method, URI> to
on

Action
Show info about movie whose ID=3
Create new movie from attached form data
Update movie ID 5 from attached form data
Delete movie whose ID=5

Methods within a class	Relationships among classes
Code smells	Design smells
Many catalogs of code smells & refactorings	Many catalogs of design smells & design patterns
Some refactorings are superfluous in Ruby	Some design patterns are superfluous in Ruby
Metrics: ABC & Cyclomatic Complexity	Metrics: Lack of Cohesion of Methods (LCOM)
Refactor by extracting methods and moving around code within a class	Refactor by extracting classes an moving code between classes
SOFA: methods are **S**hort, do **O**ne thing, have **F**ew arguments, single level of **A**bstraction	SOLID: **S**ingle responsibility per class, **O**pen/closed principle, **L**isk substitutability, **I**njection of dependencies, **D**emeter principle

Start and stop meeting promptly
Agenda created in advance; no agenda, no meeting
Minutes recorded so everyone can recall results
One speaker at a time; no interrupting talker
Send material in advance, since reading is faster
Action items at end of meeting, so know what each should do as a result of the meeting
Set the date and time of the next meeting

A class should have *one and only one* reason to change

- Each *responsibility* is a possible *axis of change*
- Changes to one axis shouldn't affect others

What is class's responsibility, in ≤25 words?

- Part of the craft of OO design is *defining* responsibilities and then sticking to them

Models with many sets of behaviors

- eg a user is a moviegoer, and an authentication principal, and a social network member, ...etc.
- really big class files are a tipoff

Load page once; no further page reloads since everything else done via AJAX

Uses same MVC flows we already know!

- but rendered content from server is handed t *your* JS code, not to browser

So, what should controller method rende

- Certainly not a full HTML page
- HTML snippet? ➔incorporate into page
- "Raw" text? ➔hard to send structured data
- XML or JSON!

AJAXy app:
+ Likely faster page load time, since user's browser can open various JavaScript connections parallel
+ Much less load on server since it won't spend time waiting for responses from sources
– Higher load on sources even though they're returning same info over and over during repeated visits over short timescales
– Testing client code requires stubbing/Webmocking multiple sources
Conventional Rails app:
+ Can take advantage of caching to suppress refresh of sources at the expense of having sometimes-stale information
+ Easy to test by stubbing out all sources at server
– Hard to get concurrency across requests to sources
At least one solidly valid pro and con for each approach were necessary for full credit

Because it's embedded in browser, avaScript code can:

. be triggered by user-initiated *events* (mouse down, mouse hover, keypress, …)

. make HTTP requests to server *without* triggering page reload

. be triggered by network events (e.g. server responds to HTTP request)

. examine & *modify* (causing redisplay) current document

nterpreted, dynamically typed

lo classes; *prototypal inheritance* instead (later)

asic object type is a hash; keys sometimes called *lots* or *properties*

```
ar movie={title: 'Up',
 "releaseInfo": {rating: 'G', year: 2010}}
```

- Slot names can always be in quotes; OK to omit if slot name is also a legal JS variable name

lse JavaScript console window in modern rowsers to try things interactively

he *global object* defines some constants that are art of JS's runtime environment

- `this` in global scope refers to global object

ach HTML page gets its own JS global object

- So each page must separately load any desired JS:

```
= javascript_include_tag 'application'
```

(usually in `application.html.haml`) ➔

```
<script src="/public/javascripts/application.js">
</script>
```

Why use helper vs explicit tag? *asset pipeline*

Best practice: JS code in multiple separate files

unctions are *first class objects* & *closures*

A function is a lambda expression

```
ar make_times = function(mul) {
 return function(arg) { return arg * mul; }

' or:  function make_times(mul) { ... }

imes2 = make_times(2)
imes3 = make_times(3)
imes2(5) ➔ 10
imes3.call(null, 5) ➔ 15
```

Every object has a prototype

No Classes; object inheritance based on "prototypes"

- each newly-created object has a prototype
- `obj.__proto__` (except in some versions of IE)
- when slot lookup fails in an object, its prototype is consulted, and so on up the chain
- so object "inherits" both value-slots and function-slots

Prototypal inheritance or *differential inheritance*

"Constructor" style functions

a/k/a JS's single worst design flaw

Call a function using `new` ➔ creates new object (`this`) whose prototype is whatever the function's prototype is, *and returns it*

Call a function on that object ➔ `this` is bound to that object (receiver)

Call a function without a receiver ➔ `this` *assumed to be Global Object,* `window`

Call a "constructor-like" function without `new` ➔ *return value is undefined*

jQuery

A powerful framework for DOM manipulation

- adds many useful features over browsers' built-in JSAPI
- homogenizes incompatible JSAPI's across browsers

Don't forget to load it…*on every page??*

- Rails 3 includes by default, or load from Google

Defines a single polymorphic global function `jQuery()`, aliased as `$()`

Four ways to call it…keep them straight!

Client-side JavaScript code can interact with HTML page elements because this functionality
(a) is part of the JavaScript language
(b) is part of the browser
(c) is provided by the JSAPI (b) and (c) only ✔

Which is NOT true about functions in JavaScript?

They can be anonymous

They always return a value, even if that value might be `undefined`

They can be passed a function as an argument

They can execute concurrently with other functions ✔

When using JavaScript for client-side form validation, which is NOT true?

JavaScript code can inspect DOM element attributes to see what the user typed

JavaScript code can prevent the "Submit" button from submitting the form

Some validations may be impractical to perform on client so must be done on server

The server doesn't have to repeat validations already performed by JavaScript ✔

What: occurrences that affect the user interface

- User interacts with a page element
- Previously-set timer expires
- Data received from server in response to AJAX call (later)

Events are usually (not always) associated with a DOM element

Bind a *handler* (function) for a particular event type on one or more elements

Manipulating the DOM using events: overview

Identify element(s) you will want to do stuff to, make sure they're conveniently selectable using `$()`

Similarly, identify element(s) on which interactions will trigger stuff to happen

Create failing test for handler using TDD

Write handler function(s) that cause the desired stuff to happen

In a *setup function,* bind the handlers (third way to call `$()`)

http://pastebin.com/RSVXzqLR

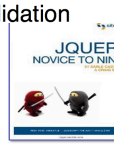
What about links & buttons that are clickable *without* JavaScript?

- handler runs first
- if handler returns *false*, no other action taken
- otherwise, other actions follow handler
- example use: client-side form validation

Select elements with `$()` (or wrap to give them secret jQuery powers)

Inspect them…

`text()` or `html()`
`is(:checked)`, `is(:selected)`, etc.
`attr('src')`



Add/remove CSS classes, hide/show

Create setup function that *binds* handler(s) on element(s)

- common ones: `onclick`, `onsubmit`, `onchange`
- Pass func to `$()` (alias for `document.ready()`)

`$()` or `jQuery()` with any CSS3 expression

```
$('#movies'), $('.heading'), $('table')
```

Warning! ≠ `document.getElementById()`

Equivalent: `$(window.document).find(`*selector*`)`

Warning! may return multiple elements, but it's not necessarily a JavaScript array!

- but there is an iterator for it

Way #2: Create element

```
var newDiv = $('<div class="main"></div>');
```

Resulting elements have jQuery superpowers!

AJAX == Asynchronous Javascript And Xm

JSAPI call `XmlHttpRequest` (a/k/a `xhr`) contacts server *asynchronously* (in background) and *without redrawing page*

- Normal HTTP request, w/special header:
  `X-Requested-With: XmlHttpRequest`

Controller action receives request via route

What should it render in response?

```
render :layout => false
render :partial => 'movies/show'
render :json => @movies   (calls to_json)
render :xml => @movies   (calls to_xml)
render :text => @movie.title
render :nothing => true
```

The basic AJAX cycle, as se from browser JSAPI

```
r = new XmlHttpRequest;
r.open(method,URL,async);
```

`method` ∈ GET,POST,HEAD,PUT,DELETE...

`async`: true means script should not block (important!)

```
r.send("request content");
r.abort();
```

- Callbacks during XHR processing

```
r.onReadyStateChange=function(XmlHttpRequest r)
  { ... }
```

- function inspects `r.readyState` ∈ uninitialized,open, sent,receiving,loaded

- `r.status` contains HTTP status of response
- `r.responseText` contains response content string

`javascript_include_tag 'application'`

your code in `app/assets/javascripts/*.js`

Define *handler function* that...

- optionally inspects element state, attributes, …
- calls `$.ajax()`

Define controller action & route to receive reques and determine what will be returned

Define *callback function* to receive server reply

- Unpack JSON objects & modify DOM?
- Replace existing HTML element (e.g. `<div>`) in place?
- Add/change/remove CSS classes/properties?

Which is FALSE concerning AJAX/ XHR vs. non-AJAX interactions?

AJAX requests can be handled with their own separate controller actions

In general, the server must rely on explicit hint (like headers) to detect XHR

The response to an AJAX request can be any content type (not just HTML)

If the server fails to respond to an XHR request, the browser's UI will freeze ✔

•Putting it together example:

```
describe('Clicking Hide button', function() {
  it('hides Movie div', function() {
    $('a#hide').trigger('click');
    expect($('div#movie')).toBeHidden();
```

Strategy: create "spy" method that *replaces* real method *that is a property of an object*

```
pyOn(MoviePopup,'new').andReturn(value)
                .andCallThrough()
                .andCallFake(func)
```

- Why no quotes around MoviePopup?

Examine calls on spy:

```
xpect(MoviePopup.new.mostRecentCall.args).
  toContain("Gravity")
xpect($.ajax.mostRecentCall.args[0]['url']).
  toEqual("/movies/1")
```