声明：

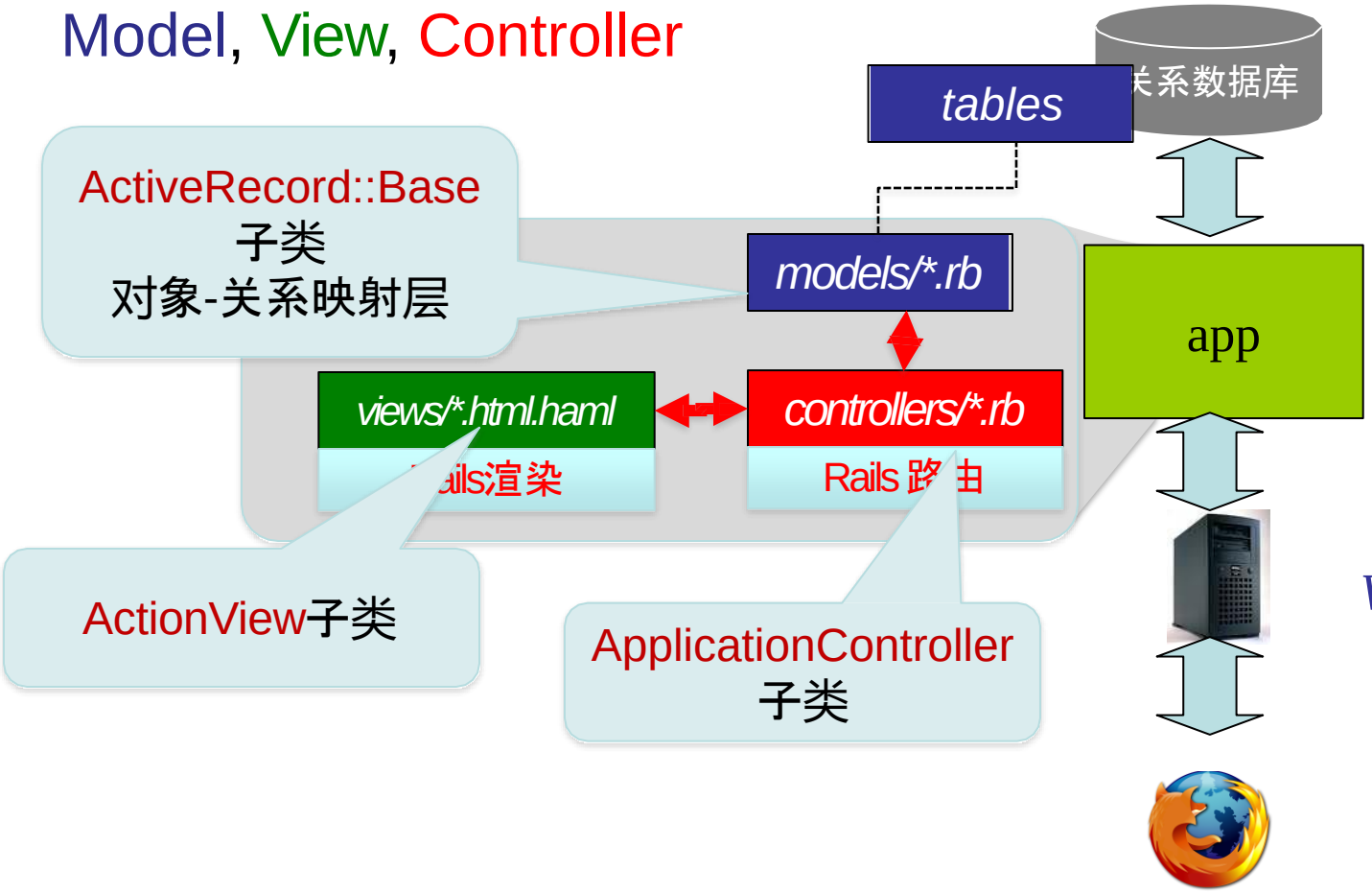
1.本课程所用PPT和作业内容源于edx：BerkeleyX： Engineering Software as a Service。

2. 本课程所用PPT和作业内容经过了部分删除和修改。

3. 本课程所用代码管理平台和实验部署平台涉及部分互联网开源内容。

Rails作为MVC框架

Model, View, Controller



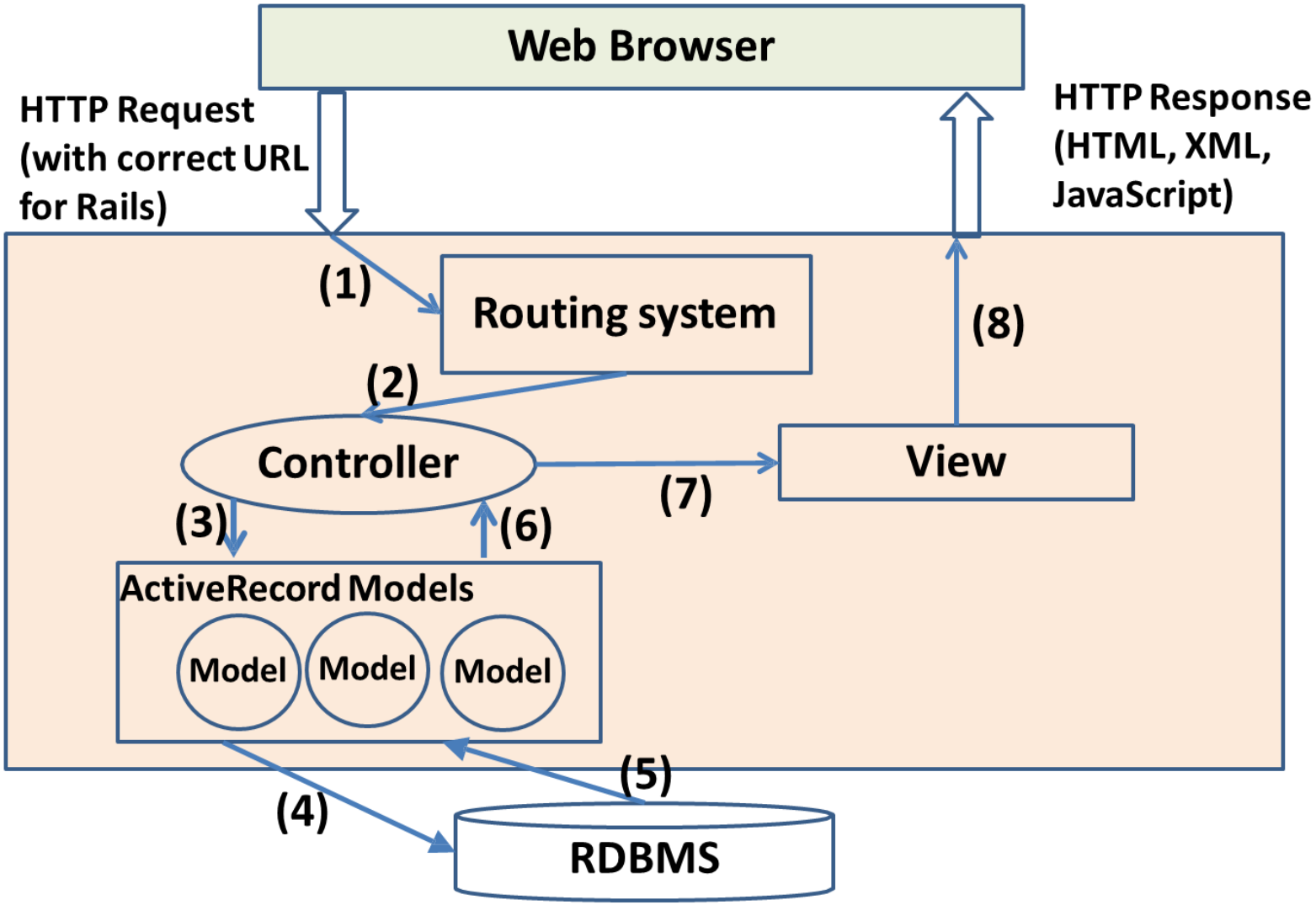
持久存储:
mysql 或sqlite3

业务逻辑:
业务功能代码&
Rack应用服务器

表现:
WEBrick, Apache, etc

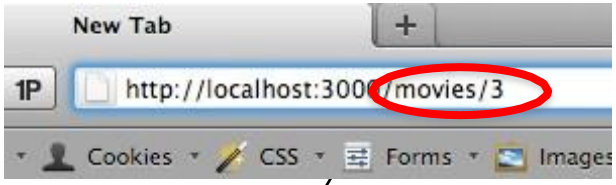
客户端:
Firefox

Rails路由子系统简介



一个Rails应用程序运行之旅

1. 路由(在routing.rb中)将传入的URL映射到控制器动作并提取任何可选参数
   - 路由的“通配符”参数(例如:id)，加上URL中“?”之后的任何内容被放入params[]哈希表中，可在控制器动作中访问
2. 控制器动作设置实例变量，视图可见
   - views/目录下的子目录和文件名匹配控制器动作名称
3. 控制器动作最终导致一个视图被呈现



config/routes.rb

app/controllers/movies_controller.rb

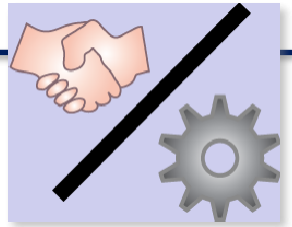
```
def show
  id = params[:id]
@m=Movie.find(id)
end
```

app/views/movies/show.html.haml

```
%li
  Rating:
  = @m.rating
```

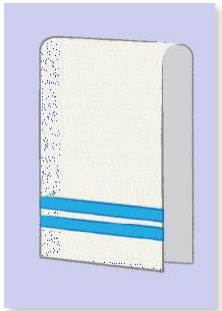
```
get '/movies/:id' => 'movies#show'
```

Rails哲学



- *约定优先于配置* 原则
  - 如果命名遵循某些约定，则不需要配置文件

```
MoviesController#show in movies_controller.rb
    ➔ views/movies/show.html.haml
```



- Don't Repeat Yourself (DRY)
  - 提取公共功能的开发原则
- 两者实现都依赖于Ruby语言特性:
  - 自省和元编程 -introspection and metaprogramming
  - 块(闭包) -blocks (closures)
  - 模块（混合）-modules (mix-ins)

SaaS框架——高级Rails

SaaS框架——高级Rails

- 1. 提炼MVC
- 2. 单点登录(SSO)和第三方身份验证
- 3. 关联和外键
- 4. 间接关联 （Through Association）
- 5. 用于关联的RESTful路由
- 6. 提炼具有可重用范围的查询

提炼MVC

(ESaaS §5.1)

不要重复自己(DRY)—但是如何做到?

- 目标: 强制电影名称必须小于40个字符
  - 在应用程序中创建或编辑电影的每个地方调用“检查”功能?

    这不是提炼(DRY)!

- 如何提炼**横切关注点**(Cross-cutting concern):
  - 逻辑上集中描述，但在实现中可能出现在多个地方?

背景和历史：GoTo和来源

- CACM, 1968 Letter to Editor

Go To Statement Considered Harmful

Key Words and Phrases: go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR Categories: 4.22, 5.23, 5.24

EDITOR:

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of **go to** statements in the programs they produce. More recently I discovered why the use of the **go to** statement has such disastrous effects, and I became convinced that the **go to** statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code).

面向方面程序设计（AOP）
Aspect Oriented Programming

- *通知(Advice)*是实现横切关注点的一段特定代码，例如：某种检查，写日志
- *切入点(Pointcut)*是希望在运行时“注入” 通知的地方
- Advice + Pointcut = Aspect
- 目标：提炼特定代码

Rails中的某种AOP：确认(Validation)

- 针对模型类的一种约束机制，以声明的方式进行刻画指定

http://pastebin.com/2GtWshSb

- 确认(Validation)是AOP意义上的通知(Advice)
  - 在应用程序的许多地方，模型可以修改/更新

- 那么切入点（Pointcut）在哪里呢?

确认申明

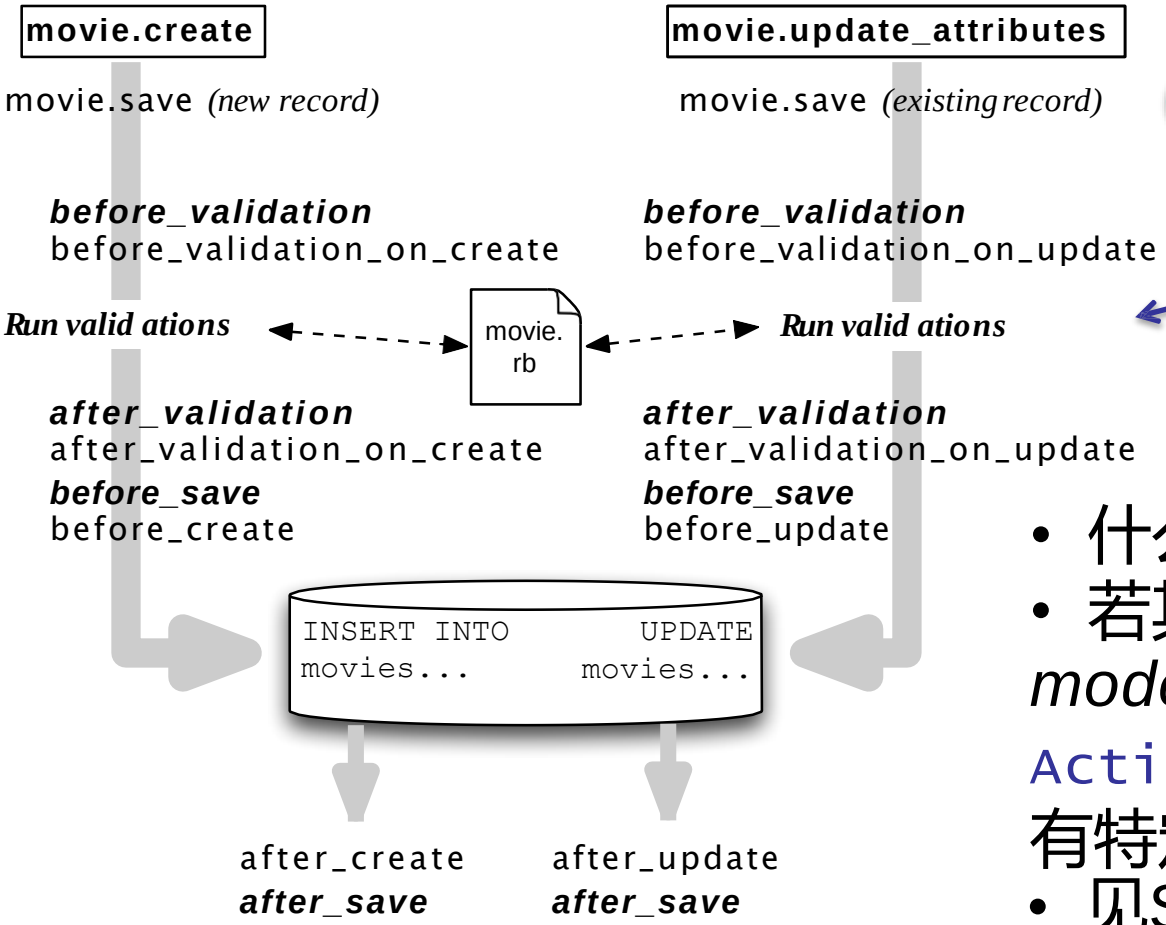
```
1.class Movie < ActiveRecord::Base
2.   RATINGS = %w[G PG PG-13 R NC-17]  #  %w[] shortcut for array of strings
3.   validates :title, :presence => true
4.   validates :release_date, :presence => true
5.   validate :released_1930_or_later # uses custom validator below
6.   validates :rating, :inclusion => {:in => RATINGS},
:unless => :grandfathered?
7.   def released_1930_or_later
8.     errors.add(:release_date, 'must be 1930 or later') if
9.       self.release_date < Date.parse('1 Jan 1930')
10.  end
11.  def grandfathered? ; self.release_date >= @@grandfathered_date ; end

12.End
```

确认激活

```
13.# try in console:
15.m = Movie.new(:title => '', :rating => 'RG', :release_date => '1929-01-
01')
16.# force validation checks to be performed:
17.m.valid?  # => false
18.m.errors[:title] # => ["can't be blank"]
19.m.errors[:rating] # => ["is not included in the list"]
20.m.errors[:release_date] # => ["must be 1930 or later"]
21.m.errors.full_messages # => ["Title can't be blank", "Rating is not
included in the list", "Release date must be 1930 or later"]
```

针对模型生命周期的回调

允许 Pre- 和 Post- Operations



- 什么时候调用 `valid?`
- 若其失败, `save` 将失败

model`.errors` 是一个 `ActiveRecord::Errors`对象, 有特定的行为

- 见Screencast 5.1.1

https://vimeo.com/34754932

另一类：针对控制器的过滤器

- 在控制器动作执行前检查某条件是否满足；过滤器可适用于控制器所有动作，也可适用某些动作；也适用于其子类
  - ApplicationController中的过滤器对所有控制器适用

http://pastebin.com/ybP6Ece1

```
1. class ApplicationController < ActionController::Base
2.   before_filter :set_current_user, :except => '/login'
3.   protected # prevents method from being invoked by a route
4.   def set_current_user
5.     # we exploit the fact that find_by_id(nil) returns nil
6.     @current_user ||= Moviegoer.find_by_id(session[:user_id])
7.     redirect_to '/login' and return unless @current_user
8.   end
```

- *过滤器可以改变执行流*
  - 通过调用redirect_to 或者 render
  - 应该在flash中添加一些内容来向用户解释发生了什么，否则将显示为“静默失败”

确认(validation) vs. 过滤器(filter)

	Validation	Filter
通知 (Advice)	检查**模型**的不变量	检查允许**控制器动作**运行的条件
切入点(Pointcut)	ActiveRecord模型生命周期的钩子	在任何公共控制器方法之前和/或之后
可以改变执行流程吗?	No	Yes
可以在任意函数中定义通知吗?	Yes;为通常情况提供的快捷方式	Yes, 必须提供函数
关于出错信息	每个模型对象都有关联的`errors`对象	捕获记录在`flash[]`, `session[]`, 或者实例变量中

缺点：会使代码更难调试

小结

- 面向方面编程(AOP)是提炼横切关注点的一种方法
- Ruby没有通用的AOP，但是Rails提供了一些“预定义”切入点
  - 确认检查 或者 断言模型生命周期中关键点的前/后条件
  - 控制器之过滤器检查 或者 断言与控制器动作相关的前置/后置条件，并可以更改控制流(重定向、呈现)
- 部分对视图的提炼（不是AOP，见书中内容）

“如果输入了无效的电影数据，应该将用户重定向回输入表单以修复错误.”实现此行为不需要以下哪一项(如果有的话)：

- 确认-Validations
- 控制器之过滤器-Controller filters
- 一个或多个控制器动作中的逻辑

单点登录（SSO）和第三方身份验证

(ESaaS §5.2)

身份认证

- 目标：检索我的Facebook好友在《纽约时报》网站上读什么?
- 《纽约时报》需要能够访问你的Facebook信息
- …但你肯定不想向《纽约时报》透露Facebook密码!
- 我们该怎么做?
  =>第三方认证

The New York Times

你是谁？你在这里能干什么?

- 认证-Authentication：证明你就是你所说的那个人
  - 用户名和密码
  - 持有与公钥匹配的私钥
  - 拥有由受信任第三方签署的加密证书
- 授权-Authorization：证明你可以做你要求做的事
  - 系统记录你有某些做事的权利吗?
  - 你是否有一个“令牌”或“能力”让你能做一些事情?

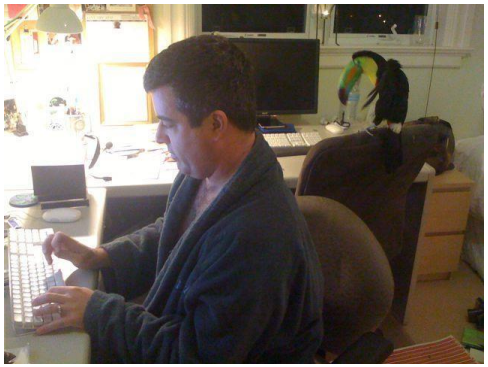
Web 1.0

- 每个网站都有单独的密码系统
- 大多数站点没有RESTful API，所以必须实际“登录”(或模拟)
- 不适合SOA!
  - 如果你每次都需要交互地登录到每个服务，服务很难协作起来
- 期望的解决方案：单点登录(SSO)
  - *但是…不想向服务B泄露服务A的密码*

它是如何工作的?(概念)

- 基础结构：防止破坏的安全令牌
- 使用密码学创建一个字符串：
  - 只有我能解密(解码)
  - 我可以检测它是否被篡改过
  - 如果不知道我的秘钥，其他人是不可能创建它的
- 通常，字符串只包含我自己存储的有价值信息的句柄
  - 接收该字符串 =>我知道我可以“信任”此句柄

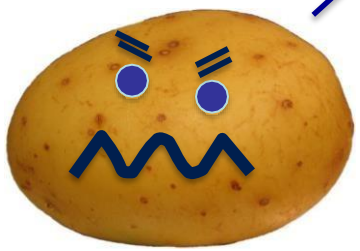
第三方认证Twitter和RottenPotatoes



1. “Login with Twitter”

2. Redirect to Twitter login page

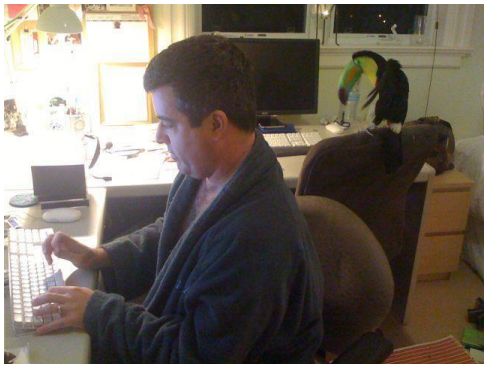
3. “OK to authorize this app?”



RottenPotatoes

twitter

第三方认证Twitter和RottenPotatoes



4. Yes, please give away my personal info

7. "Welcome, Armando"

5. Redirect to RP *callback page* with *access token*

6. Here's a token that proves I'm allowed to know this user's name

RottenPotatoes

twitter

Logos shown for educational purposes only and are the intellectual property of their owners.

它如何工作?(MVC)

- 模型 *session*作为SSO的实体
  - *session*控制器创建和删除会话，处理与身份验证提供者（如Twitter）的交互
- 一旦用户通过身份验证，我们需要一个本地*users*模型来表示他/她
  - session[]记住“当前认证用户”的主键(ID)
- OmniAuth gem为不同的“策略”提供统一的API

关于请求者和提供者之间的第三方身份验证，哪一个是正确的?

- □ 一旦完成，请求者就可以执行你在提供者上执行的任何操作
- □ 如果你在请求者上的凭据被泄露，那么你在提供者上的凭证也被泄露了
- □ 如果提供者撤销访问，请求者不再有任何你的信息
- □ 访问可以限时到一个预先设定的日期

关联 & 外键

(ESaaS §5.3)

RottenPotatoes的评论

- 简单模型：“我给它4颗土豆的评论（最高5颗）”
- 目标：能够很好地表达电影获得了很多好评
- 我们想写代码…但是怎么写?

http://pastebin.com/gU1hqm77

```
1.# it would be nice if we could do this:
2.inception = Movie.find_by_title('Inception')
3.alice,bob = Moviegoer.find(alice_id, bob_id)
4.# alice likes Inception, bob hates it
5.alice_review = Review.new(:potatoes => 5)
6.bob_review   = Review.new(:potatoes => 2)
7.# a movie has many reviews:
8.inception.reviews = [alice_review, bob_review]
9.inception.save!
10.# a moviegoer has many reviews:
11.alice.reviews << alice_review
12.alice.save!
13.# can we find out who wrote each review?
14.inception.reviews.map { |r| r.moviegoer.name } # => ['alice','bob']
```

笛卡儿乘积

table 'artists'

id	name
10	Justin
11	Shakira
12	Britney

table 'reviews'

id	desc	artist_id
30	"Terrible"	12
31	"Passable"	11
32	"Please"	10

Cartesian product: artists JOIN reviews

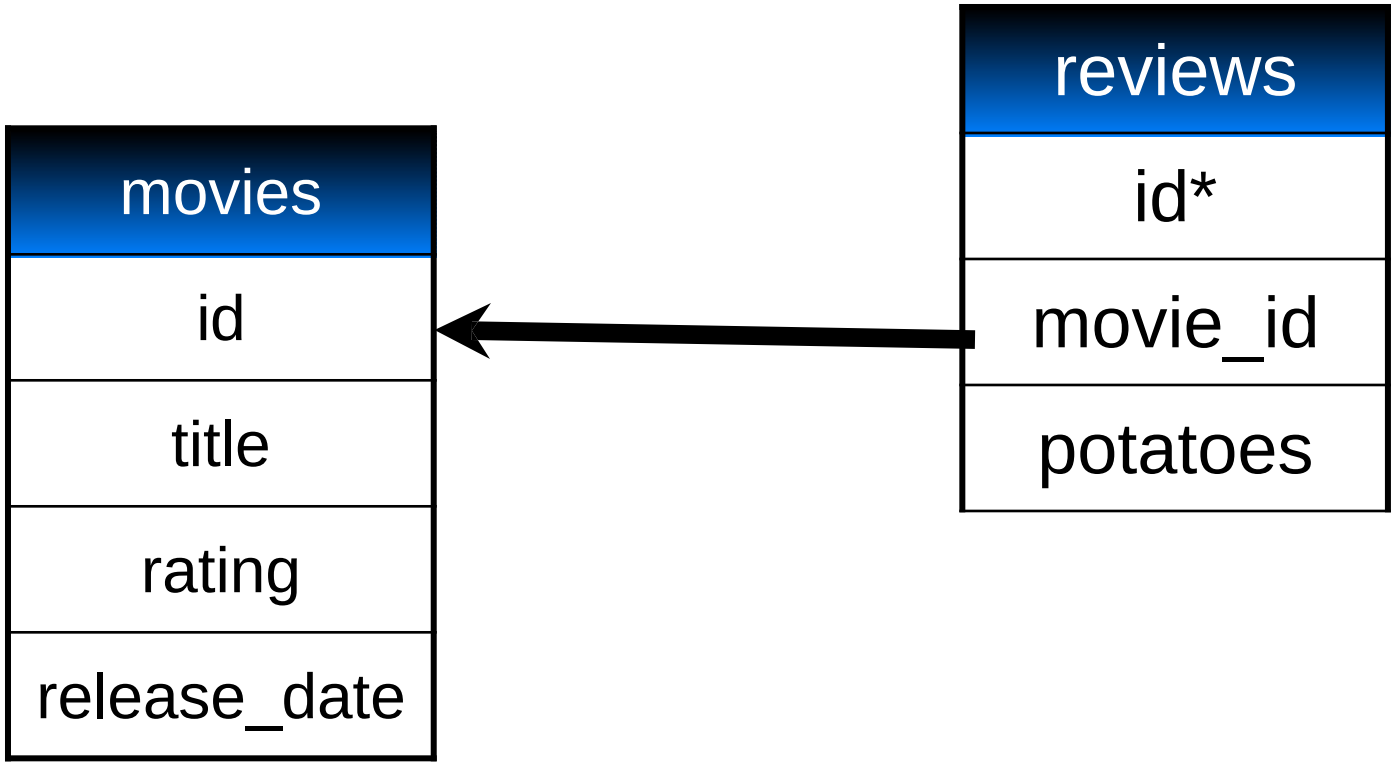
artists.id	artists.name	reviews.id	reviews.desc	reviews.artist_id
10	Justin	30	"Terrible"	12
10	Justin	31	"Passable"	11
10	Justin	32	"Please"	10
11	Shakira	30	"Terrible"	12
11	Shakira	31	"Passable"	11
11	Shakira	32	"Please"	10
12	Britney	30	"Terrible"	12
12	Britney	31	"Passable"	11
12	Britney	32	"Please"	10

Filtered Cartesian product: artists JOIN reviews ON artists.id = reviews.artist_id

artists.id	artists.name	reviews.id	reviews.desc	reviews.artist_id
10	Justin	32	"Please"	10
11	Shakira	31	"Passable"	11
12	Britney	30	"Terrible"	12

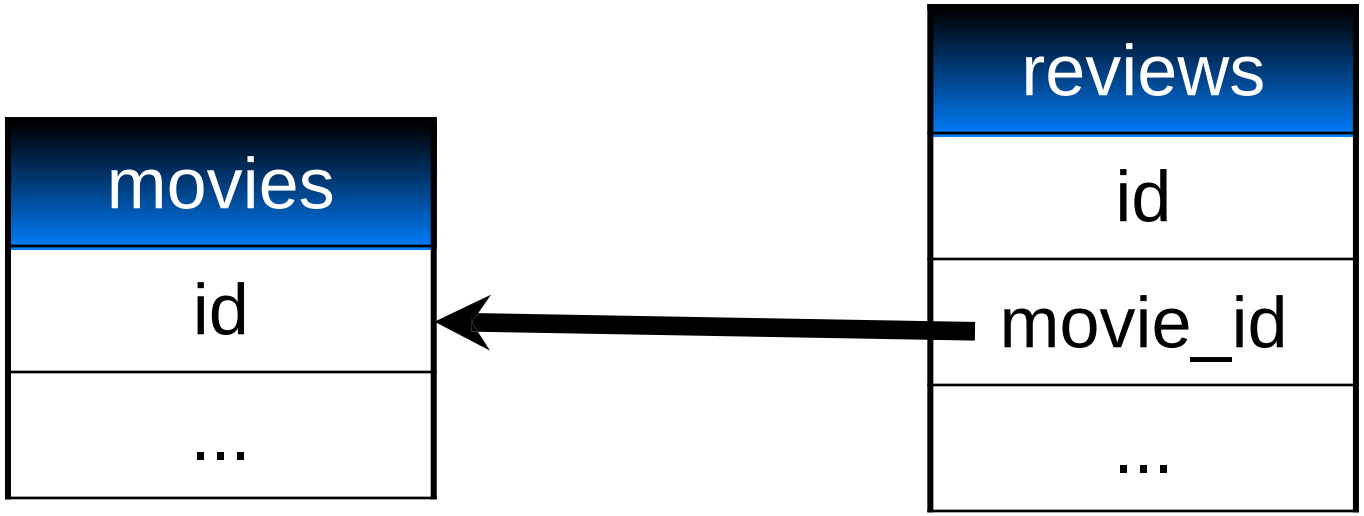
根据关系数据库模型表达“Has Many”

- 一个表中的外键(FK)指向另一个表的主键(PK)



联结（Join）

- 联接（Join）是使用PKs和FKs合并来自2个或更多表的记录的操作




```
SELECT *
FROM movies, reviews
WHERE movies.id = reviews.movie_id
```

条件笛卡儿积

关于用笛卡尔乘积来表示关系，哪个陈述是错误的?

- 你可以表示一对一关系，也可以表示一对多关系
- 你可以表示多对多关系
- 完全笛卡尔积的大小与连接条件无关
- 在has-many关系中，每个表必须有一个对应另一个表的外键

ActiveRecord对关联的支持

(ESaaS §5.3)

ActiveRecord关联

- 允许更加ruby化地操作数据库管理的关联
- 在正确设置之后，不必担心Key和join-联接

```
class Movie < ActiveRecord::Base
  has_many :reviews
end
class Review < ActiveRecord::Base
  belongs_to :movie
end
```

基本思路...

- reviews表 获得一个外键（FK）字段，对应于Movie 的主键，表示一个评论所属的电影
- 去引用(dereference)方法 movie.reviews == (惰性地)执行数据库联接，查找reviews表movie_id == movie.id的评论
- 去引用(dereference)方法 review.movie == 查找电影其主键id == review.movie_id
- 注意！必须使用迁移添加外键FK字段！

```
1.class AddReviews < ActiveRecord::Migration
2.  def self.up
3.    create_table :reviews do |t|
4.      t.integer 'potatoes'
5.        t.references 'movie'
6.      t.references 'moviegoer'
7.    end
8.  end
9.end
```

http://pastebin.com/hfvramxQ

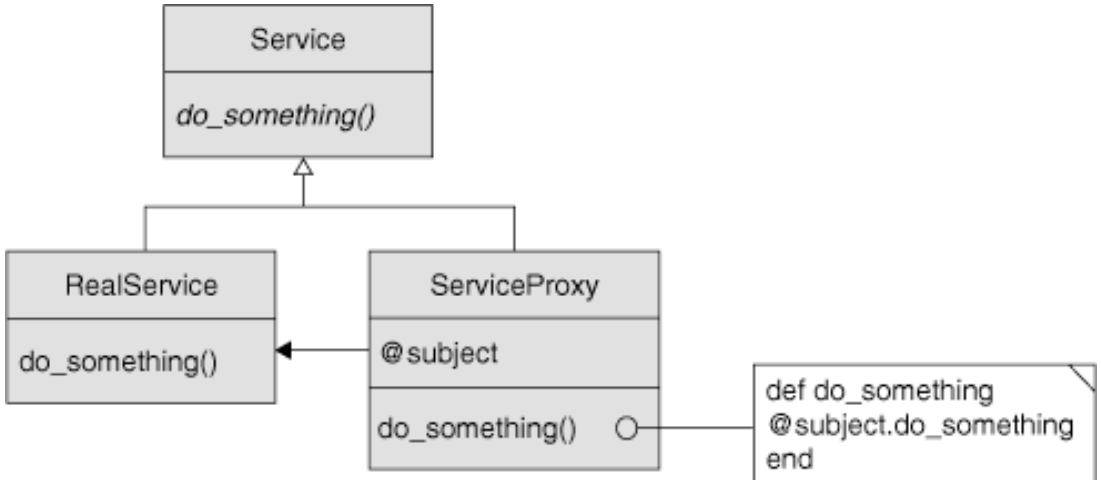
关联的代理(Proxy)方法(惰性)

- 现在可以做如下这样操作了:
  `@movie.reviews` *#列举评论*
- 反过来也可以:
  `@review.movie` *# 评论的是哪部电影?*
- 可以为一部电影添加新的评论:

```
@movie = Movie.where("title='Fargo'")
@movie.reviews.build(:potatoes => 5)
@movie.reviews.create(:potatoes => 5)
   # 它们与new() & create()有什么不同?
@movie.reviews << @new_review
   # 及时在数据库里更新 @new_review's 的 FK!
@movie.reviews.find(:first,:conditions => '...')
```

相关设计模式: Proxy

- Proxy-代理实现了与“真正的”服务对象相同的方法，但是“拦截”了每个调用
  - 认证/保护访问
  - 延迟工作(惰性的)
    - Rails例子：关联(例如 `Movie.reviews`)



哪些Ruby语言机制适合于实现被ActiveRecord模型使用的关联?

(a)在ActiveRecord::Base中构建行为

(b)将行为放在它们自己的模块（Module）中

(c)将行为放在它们自己的类(Class)中

☐ 只有 (a)

☐ 只有(a) 或 (b)

☐ 只有(a) 或 (c)

☐ 任意(a), (b), 或 (c) 合适

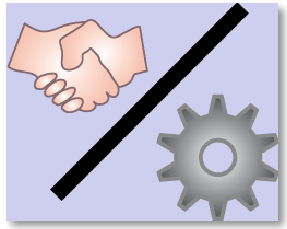
- ActiveRecord的*关联*模块(Module)
  - 使用Ruby元编程创建新方法，通过构造适当的数据库查询来“遍历”关联.
  - 还必须自己添加必要的外键字段进行数据库迁移（构建新的数据库）.

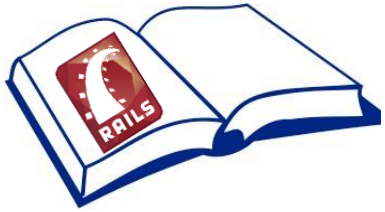
关联–构建细节

(ESaaS §5.3)

它如何工作?

- 模型必须有属性对应于拥有对象的外键
  - 例如, `reviews`表中的`movie_id`字段
- ActiveRecord 模块负责管理数据库和内存中ActiveRecord对象的这个方面
- 不要自己管理它!
  - 很难维护
  - 如果数据库模式不遵循Rails约定，可能会引起崩溃





更多的Rails烹饪法

添加一对多关联:

1. 添加has_many 到拥有模型，添加belongs_to 到被拥有模型

```
class Movie < ActiveRecord::Base
  has_many :reviews
end
class Review < ActiveRecord::Base
  belongs_to :movie
end
```

2. 创建迁移，将引用拥有方的外键添加到被拥有方
3. 执行迁移
4. rake db:test:prepare重新生成对数据库模式的测试

假设我们在表reviews中设置了外键movie_id。如果我们在Movie中添加了has_many :reviews，但忘记在表Review中添加belongs_to :movie，将会发生什么？

- ☐ 我们能运行movie.reviews，但review.movie不能工作
- ☐ 当试图保存Review时，我们将得到一个数据库错误
- ☐ 我们将无法确定一个给定的评论与哪部电影是相关的
- ☐ 上述全部

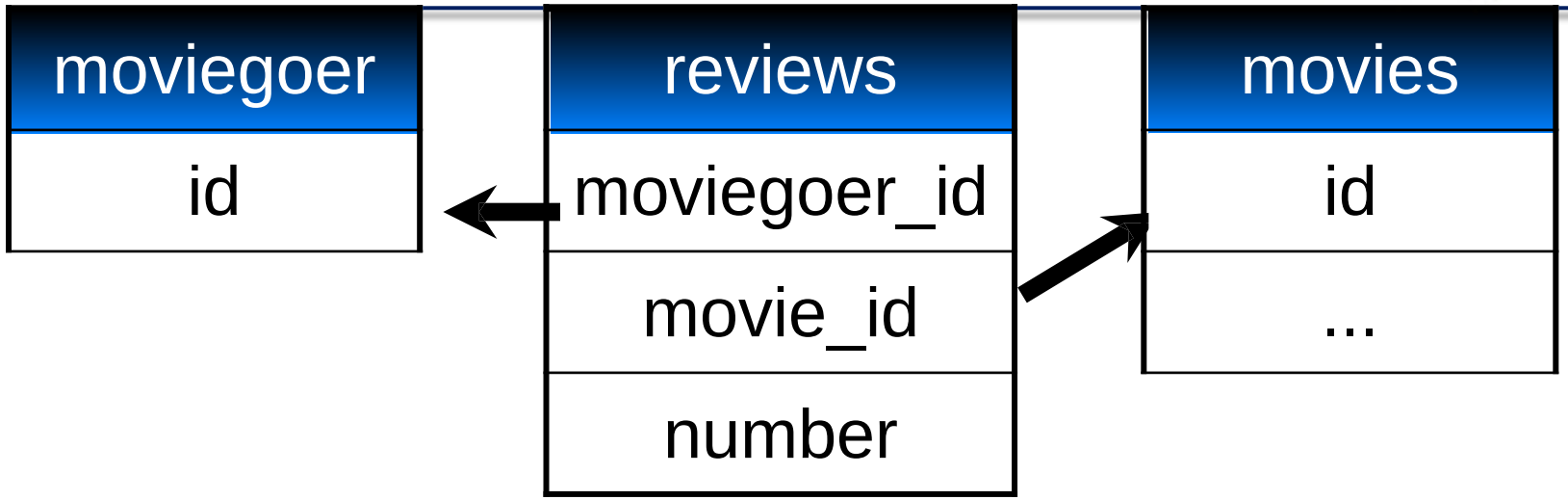
间接关联
（Through-Associations）
(ESaaS §5.4)

多对多关联

- 场景：观影者评价电影
  - 电影观众会有很多评论
  - 一部电影也可以有很多评论

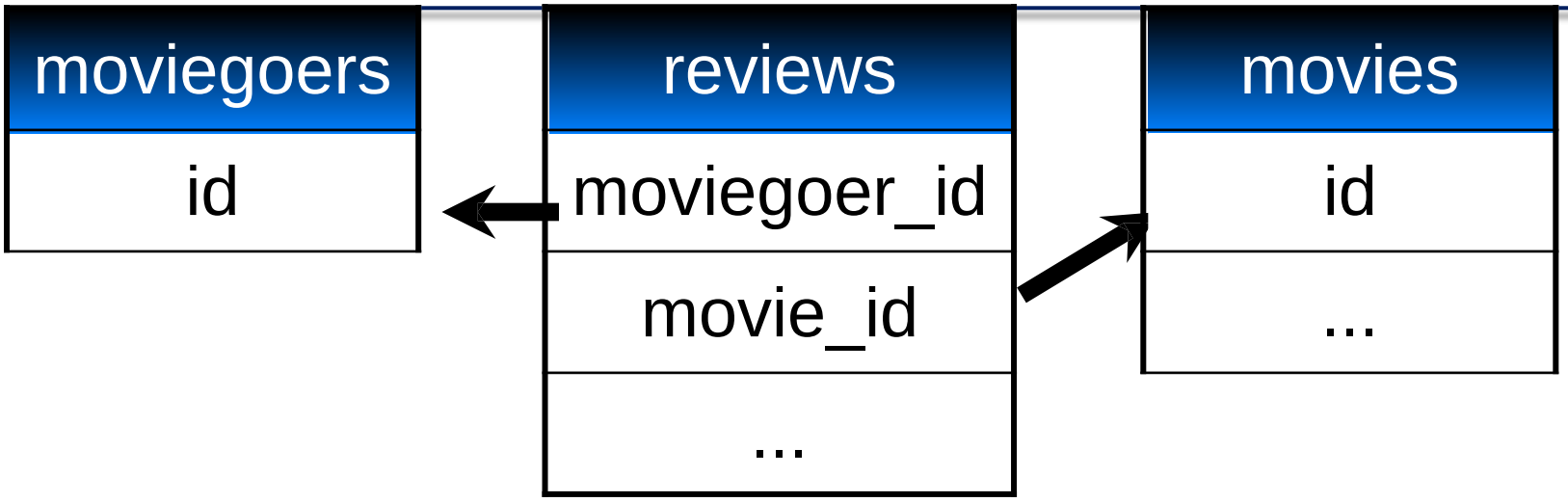
- 为什么我们不能使用`has_many & belongs_to`?
- 解决方案：创建一个新的ActiveRecord模型来建模多个关联

多对多关联



- moviegoer: `has_many :reviews`
- movie: `has_many :reviews`
- review: `belongs_to :moviegoer`
  `belongs_to :movie`
- 如何得到被某个电影观众评论的所有电影?

has_many :through



- 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`

Through

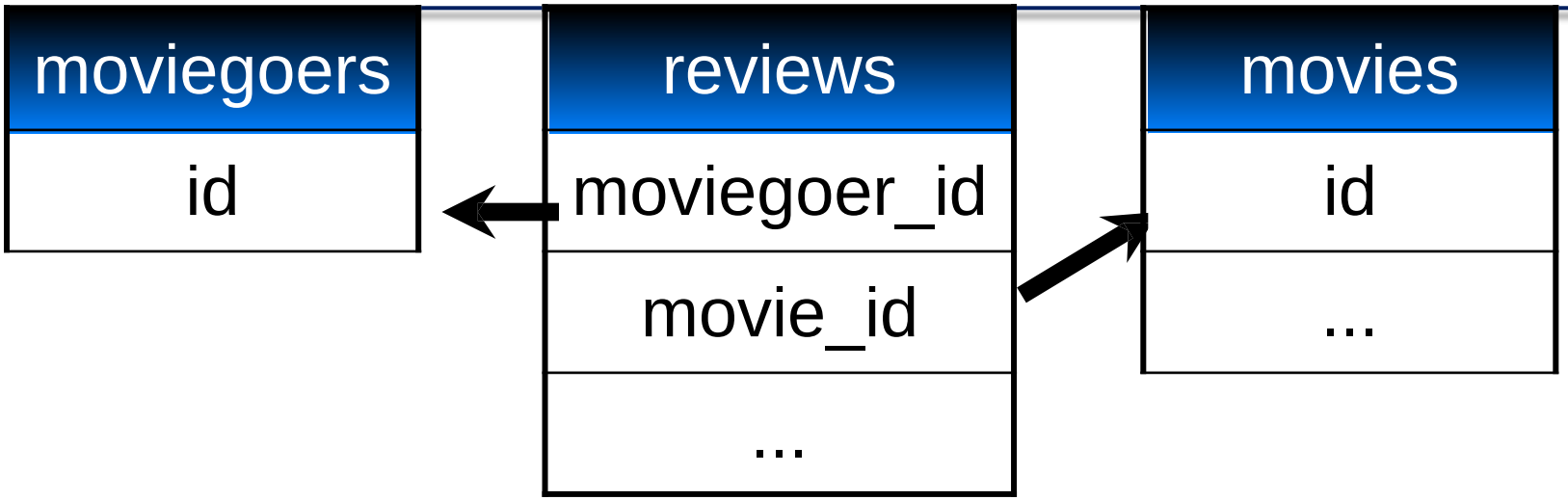
- 现在可以获得:

```
@user.movies # 用户评分的电影
@movie.users # 评价这部电影的用户
```

- 我的所有R级电影的评论(评分)

```
@user.reviews.select {
  |r| r.movie.rating == 'R' }
```

has_many :through




```
@user.movies
```

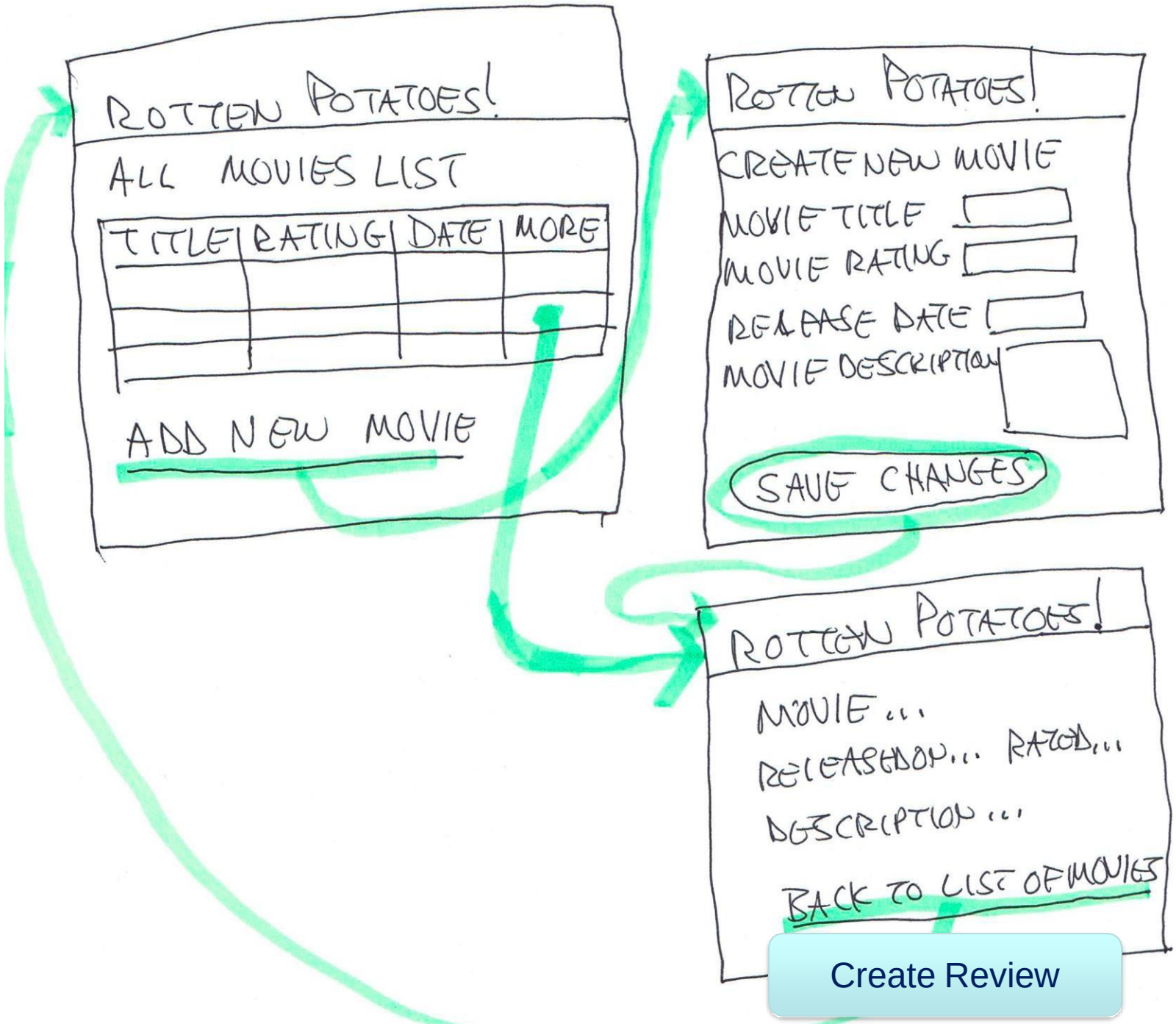
```
SELECT * FROM movies
  JOIN moviegoers ON
    reviews.moviegoer_id = moviegoers.id
  JOIN movies ON
    reviews.movie_id = movies.id
```

以下哪一个(如果有的话)不会为现有的电影m保存一个新的关联评论:

- `Review.create!(:movie_id=>m.id, :potatoes=>5)`
-
```
r = m.reviews.build(:potatoes => 5)
r.save!
```
-
```
m.reviews << Review.new(:potatoes=>5)
m.save!
```
- 以上都可以工作达到效果

用于关联的RESTful路由

(ESaaS §5.5)



Create Review

创建/更新through-关联

- 在创建一个新的评论时，如何跟踪与之关联的电影和观影者?
  - 在创建时需要此信息
  - 但是**路由帮助程序(helper)**，例如`new_movie_path`（由路由文件中的 `resources :movies` 提供）只“携带”该模型本身的ID

嵌套的RESTful路由

- 在config/routes.rb中:

```
resources :movies
```

变成

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

- 嵌套路由:通过“浏览”一部电影来访问评论

嵌套的资源路由规范（规则）提供了一组RESTful URI helper，用于针对reviews资源(模型)的CRUD动作，reviews资源是由一个movie对象所拥有的

Nested RESTful Routes

Helper method	RESTful Route and action	
movie_reviews_path(m)	GET /movies/:movie_id/reviews	index
movie_review_path(m)	POST /movies/:movie_id/reviews	create
new_movie_review_path(m)	GET /movies/:movie_id/reviews/new	new
edit_movie_review_path(m,r)	GET /movies/:movie_id/reviews/:id/edit	edit
movie_review_path(m,r)	GET /movies/:movie_id/reviews/:id	show
movie_review_path(m,r)	PUT /movies/:movie_id/reviews/:id	update
movie_review_path(m,r)	DELETE /movies/:movie_id/reviews/:id	destroy

available as `params[:movie_id]`
available as `params[:id]`

ReviewsController#create

```
# POST /movies/1/reviews
# POST /movies/1/reviews.xml
def create
  # movie_id 嵌套的路由
  @movie = Movie.find(params[:movie_id])

  # build 自动地设置movie_id外键
  @review =
      @movie.reviews.build(params[:review])

  if @review.save
    flash[:notice] = 'Review successfully created.'
    redirect_to(movie_reviews_path(@movie))
  else
    render :action => 'new'
  end
end
```

ReviewsController#new

```
# GET /movies/1/reviews/new
def new
  # movie_id 由于嵌套路由
  @movie = Movie.find(params[:movie_id])
  # new 自动设置movie_id外键
  @review ||=  @movie.reviews.new
  @review = @review || @movie.reviews.new
end
```

- 另外一种可能性: 在before-filter中做

```
before_filter :lookup_movie
def lookup_movie
  @movie = Movie.find_by_id(params[:movie_id]) ||
    redirect_to movies_path,
     :flash => {:alert => "movie_id not in params"}
end
```

视图

```
%h1 Edit

= form_tag movie_review_path(@movie,@review),
    :method => :put  do |f|

  ...Will f create form fields for a Movie or a Review?

= f.submit "Update Info"

= link_to 'All reviews for this movie',
        movie_reviews_path(@movie)
```

•记住，这些是为了方便. *不变式*是：评论在创建或编辑时必须与某个电影相关联。

如果我们还有moviegoer has_many reviews，我们能使用 moviegoer_review_path() 作为一个helper吗?

- 可以，因为约定优于配置原则，它应该按原样工作
- 可以，但我们必须在routes.rb中声明描述reviews作为moviegoers的嵌套资源
- 不行，因为到任何特定资源和操作只能有一条RESTful路由
- 不行，因为涉及评论的through关联不止一个，会导致模糊

提炼具有可重用范围的查询

(ESaaS §5.6)

用声明式作用域“定制”关联

- 适合孩子看的电影?
- 至少有N个评论的电影?
- 至少平均评分为N的电影?
- 最近被评论的电影?
- *所有这些组合?*

范围可以“堆叠”

```
Movie.for_kids.with_good_reviews(3)
Movie.with_many_fans.recently_reviewed
```

- 范围的评估是惰性的!

```
1.class Movie < ActiveRecord::Base
2.  has_many :reviews
3.  def inspect ; "#{title} (#{rating}) -
#{release_date.strftime('%D')}" ; end
4.scope :for_kids, :conditions => ['rating IN
(:ratings)', :ratings => %w(G PG)]
5.  scope :with_good_reviews, lambda { |cutoff|
6.    joins(:reviews).
7.      group(:movie_id).
8.having("AVG(reviews.potatoes) > #{cutoff}")  9. }

10.end
```

http://pastebin.com/BW40LAHX

```
1 # in controller:
2 def good_movies_for_kids
3  @m = Movie.for_kids.with_good_reviews(3)
4 end
5 # in view:
6  - @m.each do |movie|
7  %p= pretty_print(movie)
```

数据库查询的执行发生在哪里?

- ☐ 只有行 3
- ☐ 只有行 6-7
- ☐ 行 3 和行 6-7
- ☐ 依赖于for_kids的返回值

关联 小结

(ESaaS §5.7–7.9)

关联小结

- 关联是应用体系结构的一部分
  - 提供操纵RDBMS外键的高级、可重用的关联构造
    - Mix-in允许关联机制与任何ActiveRecord子类一起工作
- *Proxy*方法提供了类似枚举的行为
  - 一个n-m关联就像一个可枚举类型
    - Proxy方法是设计模式的一个例子
- 嵌套路由可以帮助以RESTful方式管理维护关联
  —但它们是可选的，并不是魔法