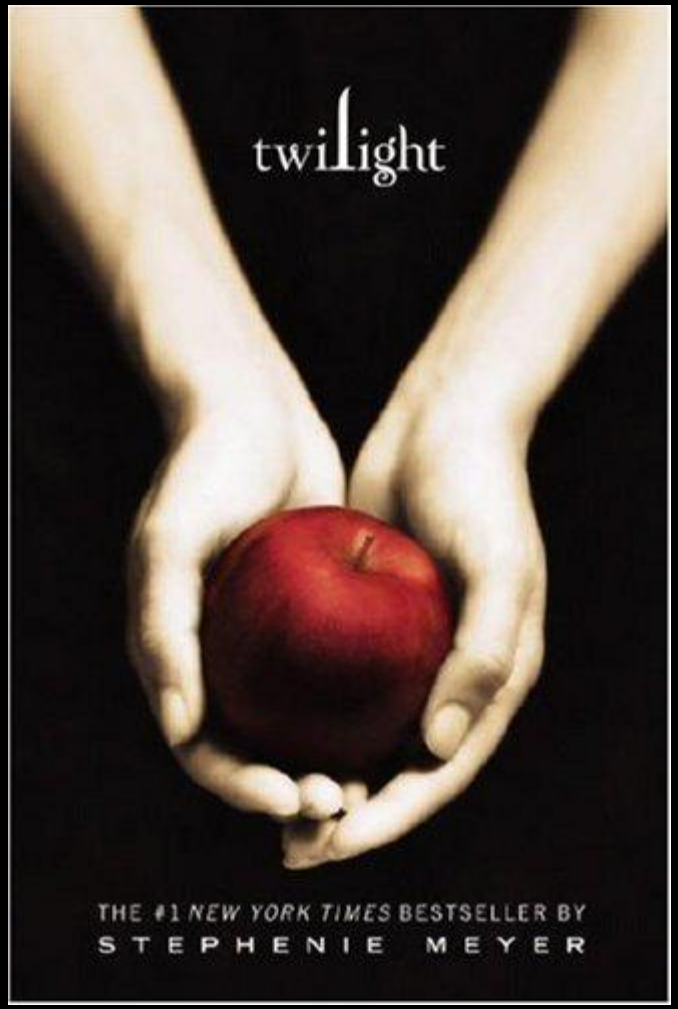
Groovy Vampires

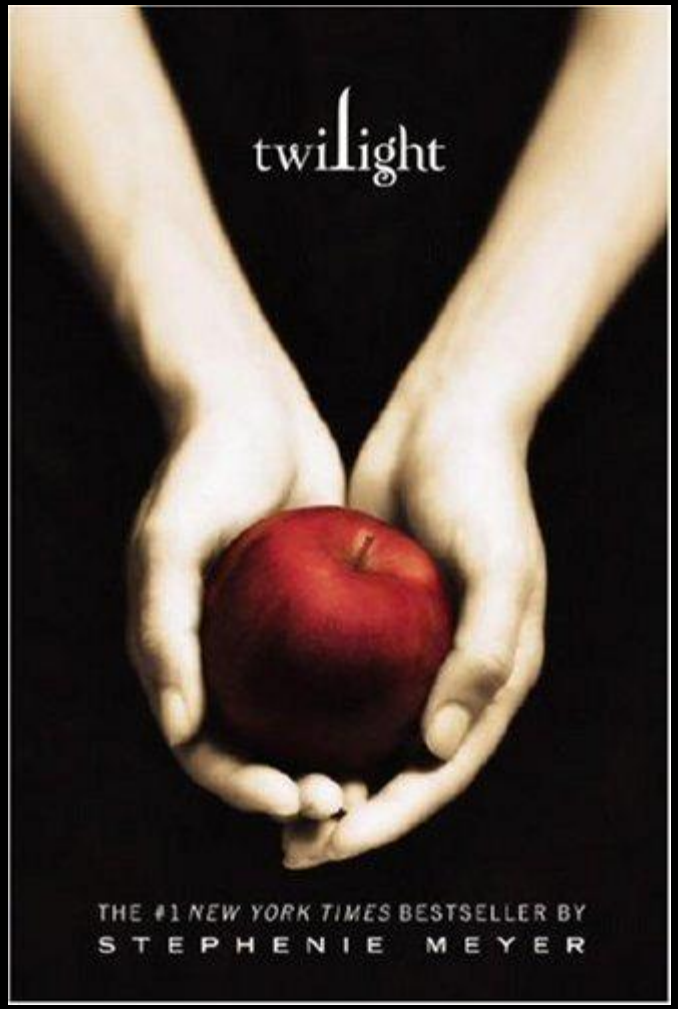
Groovy, REST, NoSQL, and Bad Marketing

BY ALDOUS HUXLEY
Only one
Digital Photography
Computer
College Guides

Two (or more)
New Teen Paranormal Romance
Teen Paranormal Romance
THE HALLOWED ONES
LAURA BICKLE
DANCE OF SHADOWS
YELENA BLACK
THE DIVINERS
Libba Bray
THE GATHERING STORM
BITTER BLOOD
RACHEL CAINE
THE LOST PRINCE
JULIE KAGAWA
FALLEN IN LOVE
LAUREN KATE
Alphabetical By Author
HOUSE OF NIGHT
P. C. CAST + KRISTIN CAST
LEGACY
MOLLY COCHRAN
POISON
MOLLY COCHRAN
KRESLEY COLE
POISON PRINCESS
ANDREA CREMER
RISE
RIFT
devoted
JESSICA RULES THE DARK SIDE
BETH FANTASKEY
fated
LAUREN OLIVER



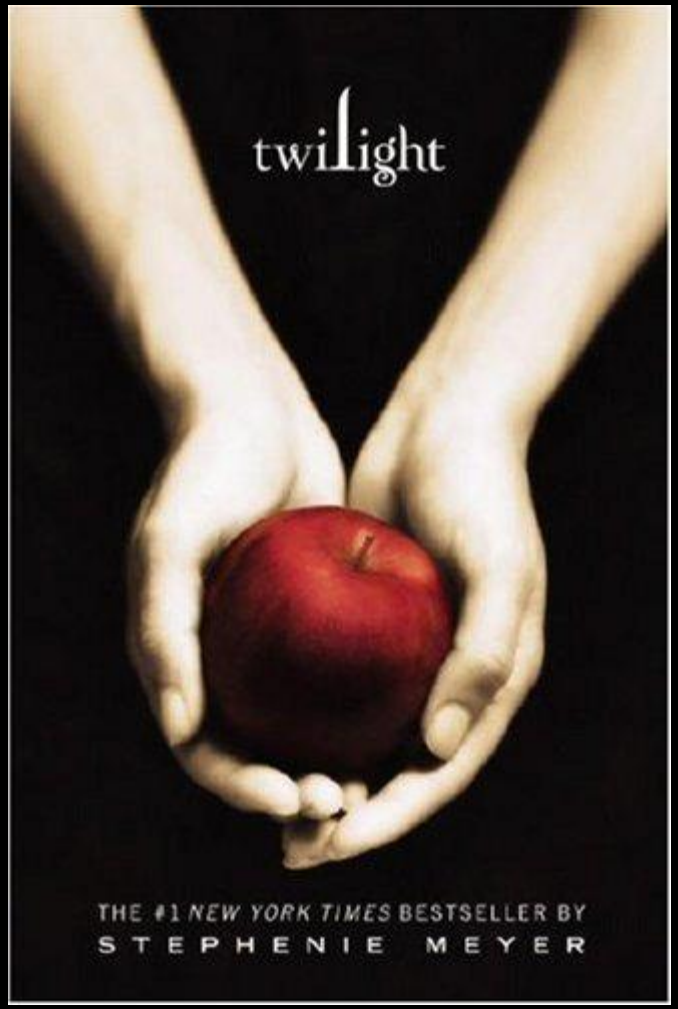
NYT #1 Best Seller



NYT #1 Best Seller

2008 #1 selling book

(over 22 million)



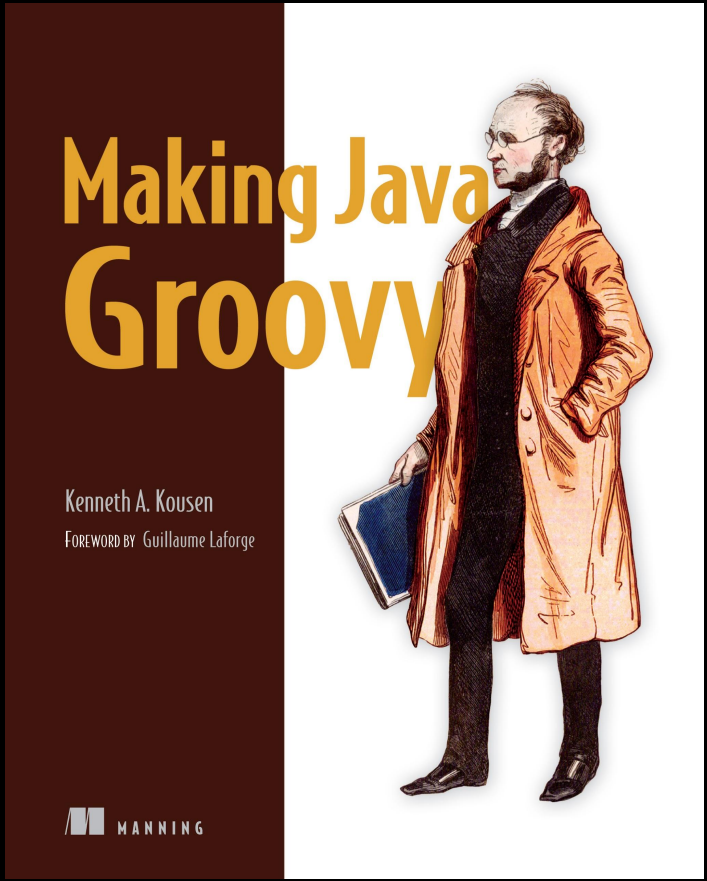
NYT #1 Best Seller

2008 #1 selling book

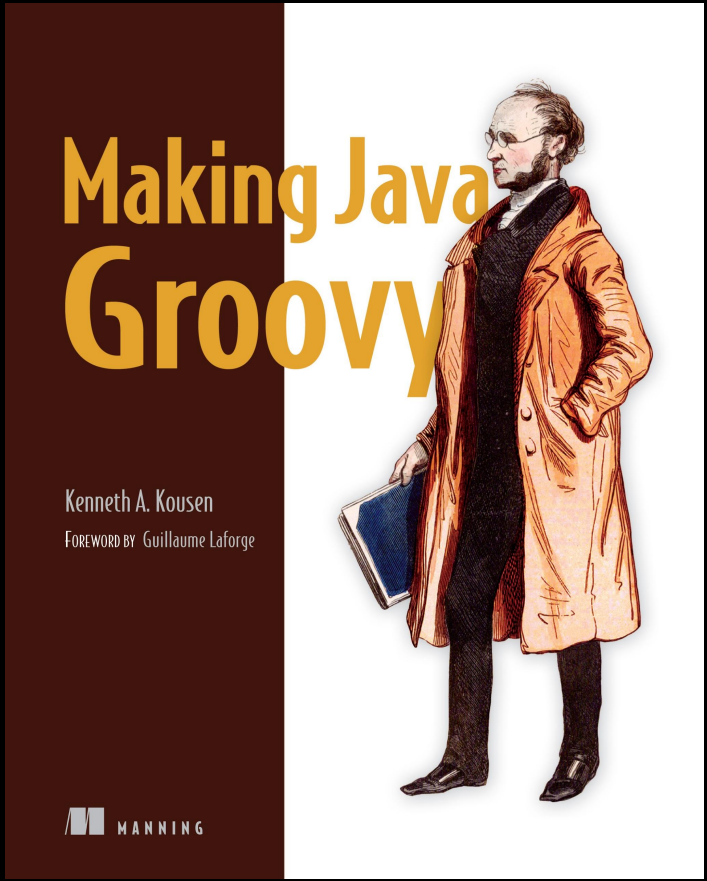
(over 22 million)

Made into a very boring

Major Motion Feature

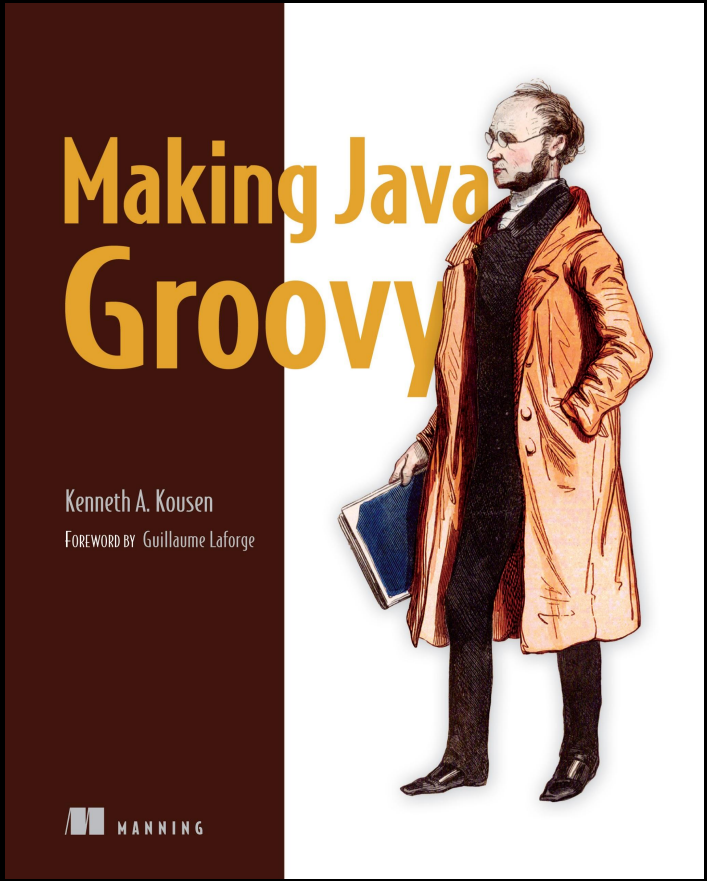


http://manning.com/kousen



http://manning.com/kousen

NYT ignored (so far)

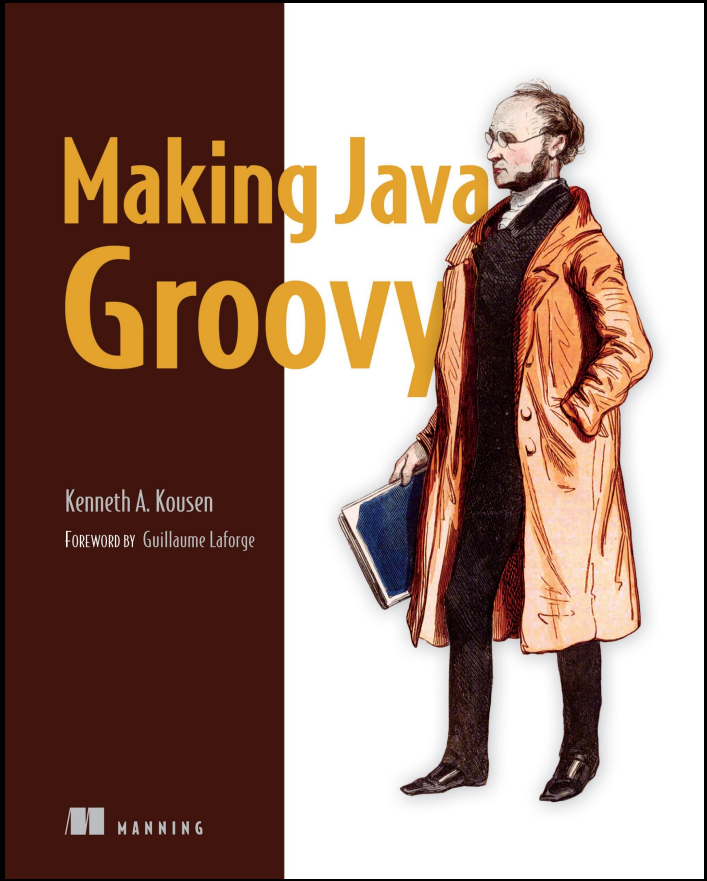


http://manning.com/kousen

NYT ignored (so far)

Great Amazon reviews

(that's actually true)

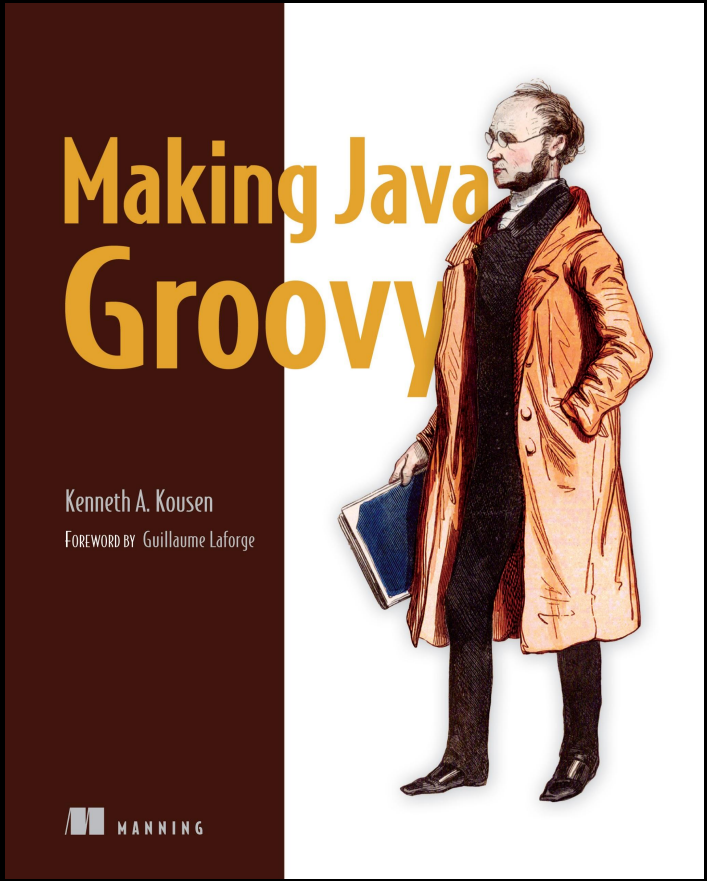


http://manning.com/kousen

NYT ignored (so far)

Great Amazon reviews

Nominated for Jolt award



http://manning.com/kousen

NYT ignored (so far)

Great Amazon reviews

Nominated for Jolt award

(I did that, *but still*)

Clearly what my book needs ...

Groovy vampires!

Rotten Tomatoes

http://www.rottentomatoes.com/

Movie review site

Rotten Tomatoes

http://www.rottentomatoes.com/

Movie review site

API provides REST access (with key)

- GET requests only

developer.rottentomatoes.com

Sign In Register

Rotten Tomatoes by Flixster

Home Documentation Brand Guidelines Forum

MASHERY MADE

Welcome to the Rotten Tomatoes API

Welcome to the Rotten Tomatoes API!

The Rotten Tomatoes API provides access to Rotten Tomatoes' ratings and reviews plus additional movie information, allowing you to build applications and widgets enriched with Rotten Tomatoes data. By implementing the Rotten Tomatoes API, here are some of the things your users can do:

- Access Rotten Tomatoes Critic and Audience ratings and review snippets
- Search for movies and retrieve detailed movie information, like cast, directors, and movie posters
- Get listings for movies that are opening soon and currently in theaters
- Get listings for movies that are releasing soon and currently available on DVD, Blu-ray and digital platforms

Get Started Right Now!

Follow the steps below to start using the Rotten Tomatoes API:

1. Register for a user account.
2. Review and accept the Brand Guidelines and Terms of Service.
3. Browse the documentation (or try our dynamic documentation!). This includes a complete list of APIs and the data you can access from each.
4. You are welcome to join a discussion in the forums to ask questions and provide feedback.

REST

- Addressable resources
- Uniform interface
- Content negotiation
- HATEOAS

REST

- Addressable resources

http://api.rottentomatoes.com/api/public/v1.0/movies.json?

q=vampire&page_limit=10&page=1&apikey=...

REST

- Addressable resources
- Uniform interface

  Rotten Tomatoes: GET requests only

REST

- Addressable resources
- Uniform interface
- Content negotiation
  - Rotten Tomatoes: JSON only
  - Content type in URL

REST

- Addressable resources
- Uniform interface
- Content negotiation
- HATEOAS
  - Embedded links for each movie
  - Top level self and next links

Sample: Blazing Saddles

- GET requests in Groovy are trivial

```
'...url...'.toURL().text
```

Sample: Blazing Saddles

- URL needs query string

Sample: Blazing Saddles

- URL needs query string
- assemble from map:

```
qs = [k1:v1, k2:v2].collect { k,v → "$k=$v" }
      .join('&')
```

Sample: Blazing Saddles

```
// API key in file
String apiKey = new File('rotten_tomatoes_apiKey.txt').text
```

Sample: Blazing Saddles

```
// API key in file
String apiKey = new File('rotten_tomatoes_apiKey.txt').text

// Base URL
String base = "http://api.rottentomatoes.com/api/public/v1.0/movies.json?"
```

Sample: Blazing Saddles

```
// API key in file
String apiKey = new File('rotten_tomatoes_apiKey.txt').text

// Base URL
String base = "http://api.rottentomatoes.com/api/public/v1.0/movies.json?"

// Assemble query string
String qs = [apiKey:apiKey, q: URLEncoder.encode(
          'Blazing Saddles','UTF-8')].collect { it }.join('&')
```

toString of Map.Entry

Sample: Blazing Saddles

```
// API key in file
String apiKey = new File('rotten_tomatoes_apiKey.txt').text

// Base URL
String base = "http://api.rottentomatoes.com/api/public/v1.0/movies.json?"

// Assemble query string
String qs = [apiKey:apiKey, q: URLEncoder.encode(
          'Blazing Saddles','UTF-8')].collect { it }.join('&')

// Full URL
String url = "$base$qs"
```

Sample: Blazing Saddles

```
JsonOutput.prettyPrint(url.toURL().text)
→ "movies": [
      {
          "id": "13581",
          "title": "Blazing Saddles",
          "year": 1974,
          "mpaa_rating": "R",
          "runtime": 93,
          "release_dates": {
              "theater": "1974-02-07",
              "dvd": "1997-08-27"
          },
          ...
```

Sample: Blazing Saddles

- Inside each movie is a links collection:

```
"links": {
    "self": "http://api.rottentomatoes.com/.../13581.json",
    "alternate": "http://www.rottentomatoes.com/m/blazing_saddles/",
    "cast": "http://api.rottentomatoes.com/.../cast.json",
    "clips": "http://api.rottentomatoes.com/.../clips.json",
    "reviews": "http://api.rottentomatoes.com/.../reviews.json",
    "similar": "http://api.rottentomatoes.com/.../similar.json"
}
```

Sample: Blazing Saddles

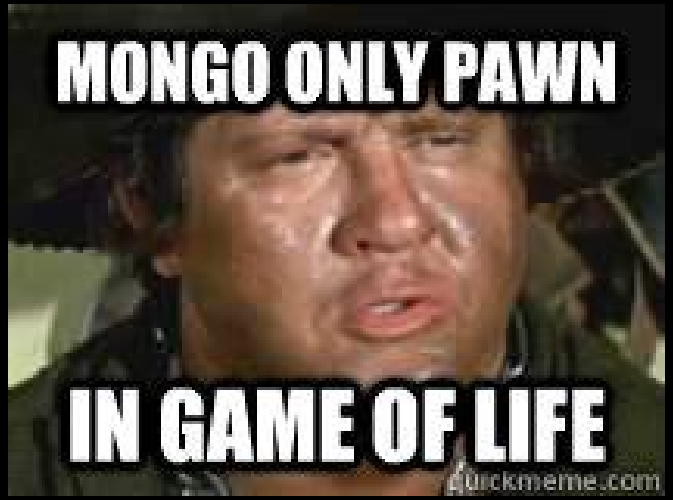
- Pagination

```
"links": {
    "self": "http://api.rottentomatoes.com/.../movies.json?...&page=1",
    "next": "http://api.rottentomatoes.com/.../movies.json?...&page=2"
}
```

Sample: Blazing Saddles

And, of course, Mongo

```
{
    "cast": [{ … }, {
        "id": "415791170",
        "name": "Alex Karras",
        "characters": ["Mongo"]
        }, { … }]
}
```



Which inevitably leads us to:

MongoDB

http://www.mongodb.org/

www.mongodb.org

mongoDB

Docs Try It Out

Agile and Scalable

MongoDB (from "hu**mongo**us") is an open-source document database, and the leading NoSQL database. Written in C++, MongoDB features:

- **Document-Oriented Storage »**
  JSON-style documents with dynamic schemas offer simplicity and power.
- **Full Index Support »**
  Index on any attribute, just like you're used to.
- **Replication & High Availability »**
  Mirror across LANs and WANs for scale and peace of mind.
- **Auto-Sharding »**
  Scale horizontally without compromising functionality.
- **Querying »**
  Rich, document-based queries.
- **Fast In-Place Updates »**
  Atomic modifiers for contention-free performance.

MongoDB home page, www.mongodb.org

MongoDB

- Document based

MongoDB

- Document based
- BSON → Binary JSON

MongoDB

- Document based
- BSON → Binary JSON
- Open source

MongoDB

- Document based
- BSON → Binary JSON
- Open source
- Full indexing

MongoDB

- Document based
- BSON → Binary JSON
- Open source
- Full indexing
- JavaScript queries

MongoDB

Start server

```
> mongod
```

runs on port 27017 by default

MongoDB

Client program is mongo:

```
$ mongo
```

MongoDB

Client program is mongo:

```
$ mongo
> show databases
```

MongoDB

Client program is mongo:

```
$ mongo
> show databases
> use movies
```

MongoDB

Client program is mongo:

```
$ mongo
> show databases
> use movies
> show collections
```

MongoDB

Client program is mongo:

```
$ mongo
> show databases
> use movies
> show collections
> db.vampireMovies.find()
```

Java Driver

Java driver available

http://docs.mongodb.org/ecosystem/drivers/java/

JavaDocs

http://api.mongodb.org/java/current/

BasicDBObject

```
com.mongodb.BasicDBObject
    extends
    java.util.LinkedHashMap<String,Object>
```

wrapper for domain classes

Groovy

GMongo Project

https://github.com/poiati/gmongo/

Maintained by Paolo Poiati

(not active, but still works)

@Delegate

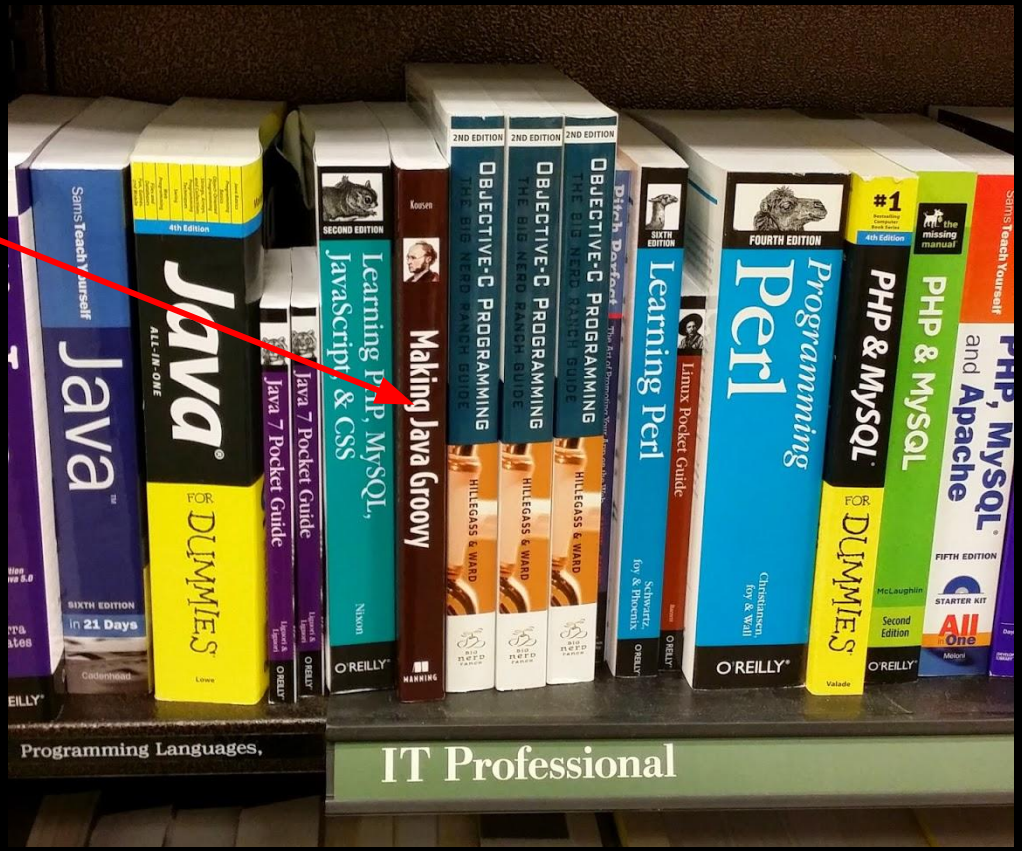
Typical Groovy idiom:

Groovy class wraps Java class

@Delegate

```
class GMongo {
  @Delegate
  Mongo mongo  // from Java driver
  …
}
```

Doesn't exactly draw your eye, does it?



Populate MongoDB

- Perform GET request at RT
- Parse results into JSON objects
- Append to DB collection
- Handle pagination through links

Populate MongoDB

- Perform GET request at RT

  Same as before, with `q:'vampire'`

http://api.rottentomatoes.com/api/public/v1.0/movies.json?apiKey=...

&q=vampire

Populate MongoDB

- Perform GET request at RT
- Parse results into JSON objects

```
def vampMovies =
    new JsonSlurper().parseText(url.toURL().text)
```

Populate MongoDB

- Perform GET request at RT
- Parse results into JSON objects
- Append to DB collection

```
db.vampireMovies << vampMovies.movies
```

Populate MongoDB

- Handle pagination

```
def next = vampMovies?.links?.next
while (next) {
    println next
    vampMovies = slurper.parseText("$next&apiKey=$key".toURL().text)
    db.vampireMovies << vampMovies.movies
    next = vampMovies?.links?.next
}
```

8.5.1 Populating Groovy vampires

This example came about because I was wandering in a bookstore recently and noticed that while there was only one bookshelf labeled "Computer," there were three others labeled "Teen Paranormal Romance." Rather than lament the decline of Western Civilization I chose to take this as evidence that I needed to add Groovy vampires to my book.

Consider the web service provided by the movie review site Rotten Tomatoes, http://developer.rottentomatoes.com. If you register for an API key, you can make HTTP GET requests that search for movies, cast members, and more. The data is returned in JSON form. The base URL for the API is located at http://api.rottentomatoes.com/api/public/v1.0. All requests start with that URL.

For example, searching for information about the movie *Blazing Saddles*[17] is done by accessing http://api.rottentomatoes.com/api/public/v1.0/movies.json?q=Blazing%20Saddles&apiKey=... (supply the API key in the URL). The result is a JSON object that looks like the following listing.

Listing 8.15 A portion of the JSON object representing the movie *Blazing Saddles*

```
{
    "total": 1,
    "movies": [
        {
            "id": "13581",
            "title": "Blazing Saddles",
```

[15] Other than the rise of dynamic languages on the JVM, of course.

[16] See www.mongodb.org/ for downloads and documentation.

[17] That's not a vampire movie, obviously, but the urge to save Mongo in MongoDB is irresistible. "Mongo o pawn in game of life" is a brilliant line and arguably the peak of the Alex Karras oeuvre.

Mapping to Classes

Map JSON to Classes

- Gson very popular

See 'json_to_gson.groovy' script

Mapping to Classes

Alternative:

- Manually map classes to JSON

Mapping to Classes

Entity classes:

- Movie
- CastMember
- MPAARating (enum)
- Rating (audience, critics)

Mapping to Classes

Add static method to Movie:

```
static Movie fromJSON(data)
```

extract data from map

populate objects

Much better

Making Java Groovy
AUTOGRAPHED COPY
Kenneth A. Kousen
MANNING
FROST BITE
RICHELLE MEAD
SHADOW KISS
RICHELLE MEAD
BLOOD PROMISE
RICHELLE MEAD
VAMPIRE ACADEMY
THE ULTIMATE GUIDE
RICHELLE MEAD
being friends with boys
the Indigo Spell
eclipse
echo
alyson noël
STEPHENIE MEYER
Evermore
Blue Moon
Shadowland
Night Star
THE STATISTICAL PROBABILITY OF LOVE AT FIRST SIGHT
JENNIFER E. SMITH
THIS IS WHAT HAPPY LOOKS LIKE
JENNIFER E. SMITH

Serve up data locally

Build web app around MongoDB

- Ratpack: http://ratpack.io

Ratpack

- Asynchronous I/O
- Optimized for Java 8 and Groovy
- Dependency Injection via Guice

Lazybones

Lazybones project from Peter Ledbrook

Generates project templates

Lazybones

> lazybones create ratpack vampires

Vampire Server

Implement class to return vampire movies

Vampire Server

Implement class to return vampire movies

Wraps GMongo instance

Vampire Server

Implement class to return vampire movies

Wraps GMongo instance

Like a MovieDAO class

- methods map to HTTP verbs

Ratpack

Use GET handler in Ratpack.groovy

Return Movie instances as required

Ratpack

- Testing is easy
  - Spock spec for Vampire server
  - Integration spec for Ratpack

Groovy
Vampires,
p. 220

Grails

- Alternative server:
  - Grails
    - REST capabilities
    - Mongo plugin

Grails

REST via annotation on domain class

```
@Resource
class Movie { … }
```

→ no explicit controller needed

Grails

URL Mappings

```
'movies'(resources: 'movie')
```

Grails

Set preference for JSON data

```
@Resource(formats = ['json'])
class Movie { … }
```

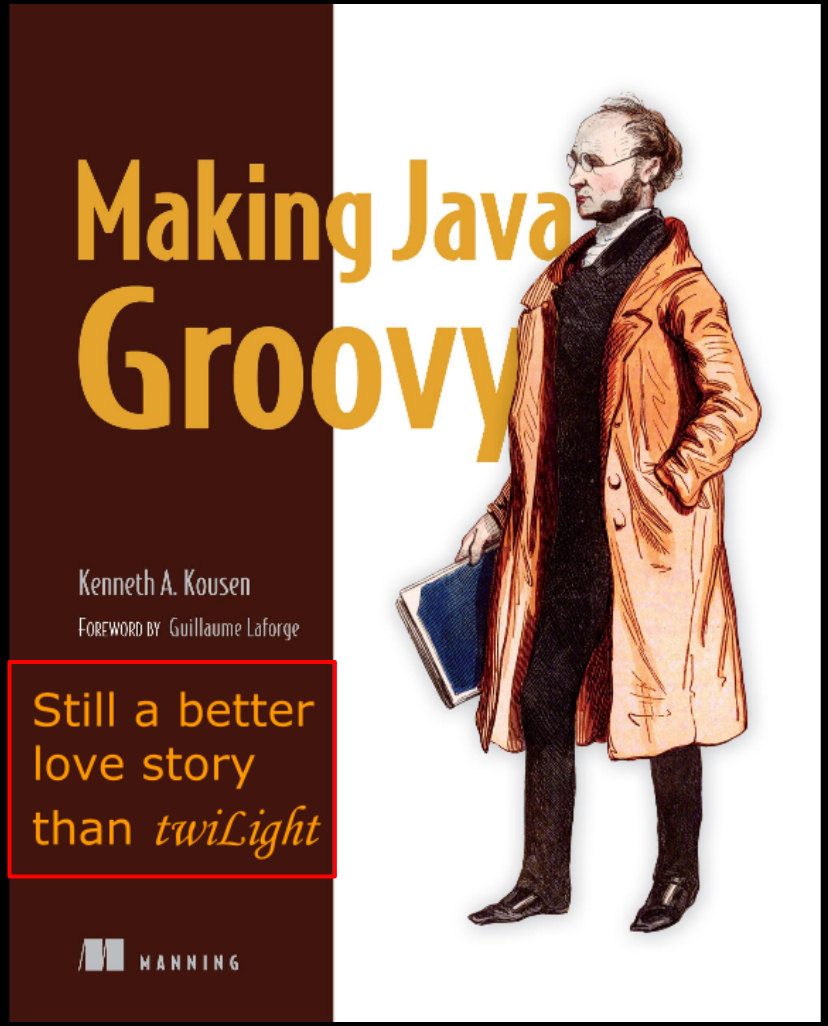
Grails

URL mappings report

```
grails> url-mappings-report
```

Grails

```
Controller: movie
 |   GET    | /movies                                    | Action: index          |
 |   GET    | /movies/create                             | Action: create         |
 |   POST   | /movies                                    | Action: save           |
 |   GET    | /movies/${id}                              | Action: show           |
 |   GET    | /movies/${id}/edit                         | Action: edit           |
 |   PUT    | /movies/${id}                              | Action: update         |
 |  PATCH   | /movies/${id}                              | Action: patch          |
 |  DELETE  | /movies/${id}                              | Action: delete         |
```


Making Java
Groovy
Kenneth A. Kousen
Foreword by Guillaume Laforge
Still a better
love story
than twiLight
MANNING

Grails

MongoDB plugin

http://grails.org/plugin/mongodb

Add to BuildConfig.groovy:

```
compile ":mongodb:3.0.1"
```

Grails

Add to domain classes:

```
ObjectId id
```

Grails

Add to domain classes:

```
ObjectId id
static mapWith = "mongo"
```

Grails

Add to domain classes:

```
ObjectId id
static mapWith = "mongo"
static embedded = [..., …, …]
```

Grails

Add to DataSource.groovy:

```
grails {
    mongo {
        host = 'localhost'
        port = 27017
        databaseName = 'movies'
    }
}
```

Grails

Now can use GORM methods

```
Movie.withCriteria {
    ratings {
        gte 'critics_score', 50
    }
    order('ratings.critics_score', 'desc')
    maxResults(10)
}
```

Maybe vampire fans don't buy Groovy, so maybe try a different approach...

Making Java Groovy

Kenneth A. Kousen

Foreword by Guillaume Laforge

Groovy Soup for the Chicken Soul (sort of)

MANNING

Or perhaps...

Making Java HODOR

Kenneth A. Kousen

Foreword by Guillaume Laforge

Westeros Edition

You win, or you throw a `GroovylessCodeException`

MANNING

Or maybe better ...

Making Java HODOR

Kenneth A. Kousen

Foreword by Guillaume Laforge

Westeros Edition

Hodor! Hodor? Hodor!

MANNING

Conclusions

REST API at Rotten Tomatoes

- GET only (like most public services)

MongoDB stores JSON natively

Conclusions

Groovy JDK makes GET requests easy

Hypermedia links for individual movies

self, cast, clips, reviews, …

Hypermedia links for pagination

Conclusions

GMongo project wraps Java API

Great use of @Delegate

Conclusions

Ratpack is fast and easy

(but not -- yet -- well documented)

Natural for REST

Shows lots of promise

Conclusions

- Grails 2.3+ adds REST
  - Uses annotations or RestController
  - URL mappings

- MongoDB plugin

GROOVY
I HAZ IT
DIYLOL.COM