Memcached

Ken Collins <metaskills.net>

757.rb – July 14th 2009

About Memcached

- Stands For "Memory Cache Daemon"
- By Brad Fitzgerald, founder of LiveJournal
- Introduced in October of 2003 to keep frequently-used LiveJournal objects in memory. Including logins, colors, styles, journal entries, commets, site text, etc.
- Used by many major sites including Flickr, Slashdot, Wikipedia and Facebook.

What Is Memcached (Or Not)

- A server(s) that caches Name Value Pairs (NVPs) in memory. This could be
- NOT a persistent data store!
- NOT queryable for a list of all objects. Only way to find something is to ask for it by name. (by design)
- No built-in security mechanisms.
- No fail-over/high-availability mechanisms. (server goes down, data is gone)

Memcached Layout

- Basically Client <=> (NVP)Server
- Server simple fast data retrival based on a key.
  - Size of key can not exceed 250 characters.
  - Size of value can not exceed 1 MB.
  - Each memcached server is atomic. Neither knows or cares about any other memcached server.
- Client libraries for most every major language. We will focus on the Ruby client used by Rails.

Installation & Configuration

- Mac OS `$ sudo port install memcached`
- Others... pretty easy. Only dependency is libevent.
  (see http://danga.com/memcached/)
- Key arguments are -p, -m and -d. All using -h
- LaunchDaemon
  `$ memcached -u nobody -m 512 -c 1024 -p 11211 -d`

```
USI=`hostname`
ERMCAP=/etc/termcap
ANPATH=/usr/man:/usr/local/man:$HOME/
ATH1=$HOME/bin:/bin:/usr/bin:/usr/loc
ATH2=/usr/new/bin:/etc:/usr/local/bir
```

Styling & Profiling :)

```
# Memcached
alias mcconf="mate /opt/local/etc/LaunchDaemons/
org.macports.memcached/memcached.wrapper"
alias stopmc="sudo launchctl stop org.macports.memcached"
alias startmc="sudo launchctl start org.macports.memcached"
alias mcr="stopmc ; startmc"
```

Memcached Basic Commands

- **ADD**: Only store data if the key does NOT exist.
- **REPLACE**: Only store data if the key ALREADY exists.
- **SET**: Add or replace data.
- **GET**: Return data.
- **INCR**: Increment a counter.
- **DECR**: Decrement a counter.

Rails With Memcached

- Sessions. (config/initializers/session_store.rb)
  ```
  ActionController::Base.session_store = :mem_cache_store, 'localhost'
  ```
- Everywhere Else! ActiveSupport::Cache (config/initializers/environments/*.rb)
  ```
  config.action_controller.perform_caching = true
  config.cache_store = :mem_cache_store
  ```
- ActiveSupport::Cache Implementations:
  - FileStore
  - MemoryStore
  - SynchronizedMemoryStore
  - DRbStore
  - **MemCacheStore**
  - **CompressedMemCacheStore**
- Accessible as Rails.cache

ActiveSupport::Cache Details

- All implementations inherit from ActiveSupport::Cache::Store which define a common simple interface.

```
read(key, options = nil)
write(key, value, options = nil)
fetch(key, options = {})
delete(key, options = nil)
delete_matched(matcher, options = nil)
exist?(key, options = nil)
increment(key, amount = 1)
decrement(key, amount = 1)
```

#read(key,options=nil)

```
cache.read("ids")                   # => nil
cache.write("ids",[1,2,3])          # => true
cache.read("ids")                   # => [1, 2, 3]
cache.read("ids", :raw => true)     # => "\004\b[\bi\006i\ai\b"
```

#write(key,value,options=nil)

```
cache.write("foo", "bar", :expires_in => 5.seconds)
cache.read("foo")  # => "bar"
sleep(6)
cache.read("foo")  # => nil
```

#fetch(key,options={})

```ruby
cache.write("today", "Tuesday")
cache.fetch("today")  # => "Tuesday"

cache.fetch("city")   # => nil
cache.fetch("city") do
  "Duckburgh"
end
cache.fetch("city")   # => "Duckburgh"

cache.write("today", "Tuesday")
cache.fetch("today", :force => true)  # => nil

cache = ActiveSupport::Cache::MemCacheStore.new
cache.fetch("foo", :force => true, :expires_in => 5.seconds) do
  "bar"
end
cache.fetch("foo")  # => "bar"
sleep(6)
cache.fetch("foo")  # => nil
```

#delete(key,options=nil)

```
cache.write("ids",[1,2,3])      # => true
cache.delete("ids")             # => true
cache.read("ids")               # => nil
cache.delete("notthere")        # => false
```

#exist?(key,options=nil)

```
# Doesn't call super, cause exist? in memcached is in fact a read
# But who cares? Reading is very fast anyway.
!read(key, options).nil?
```

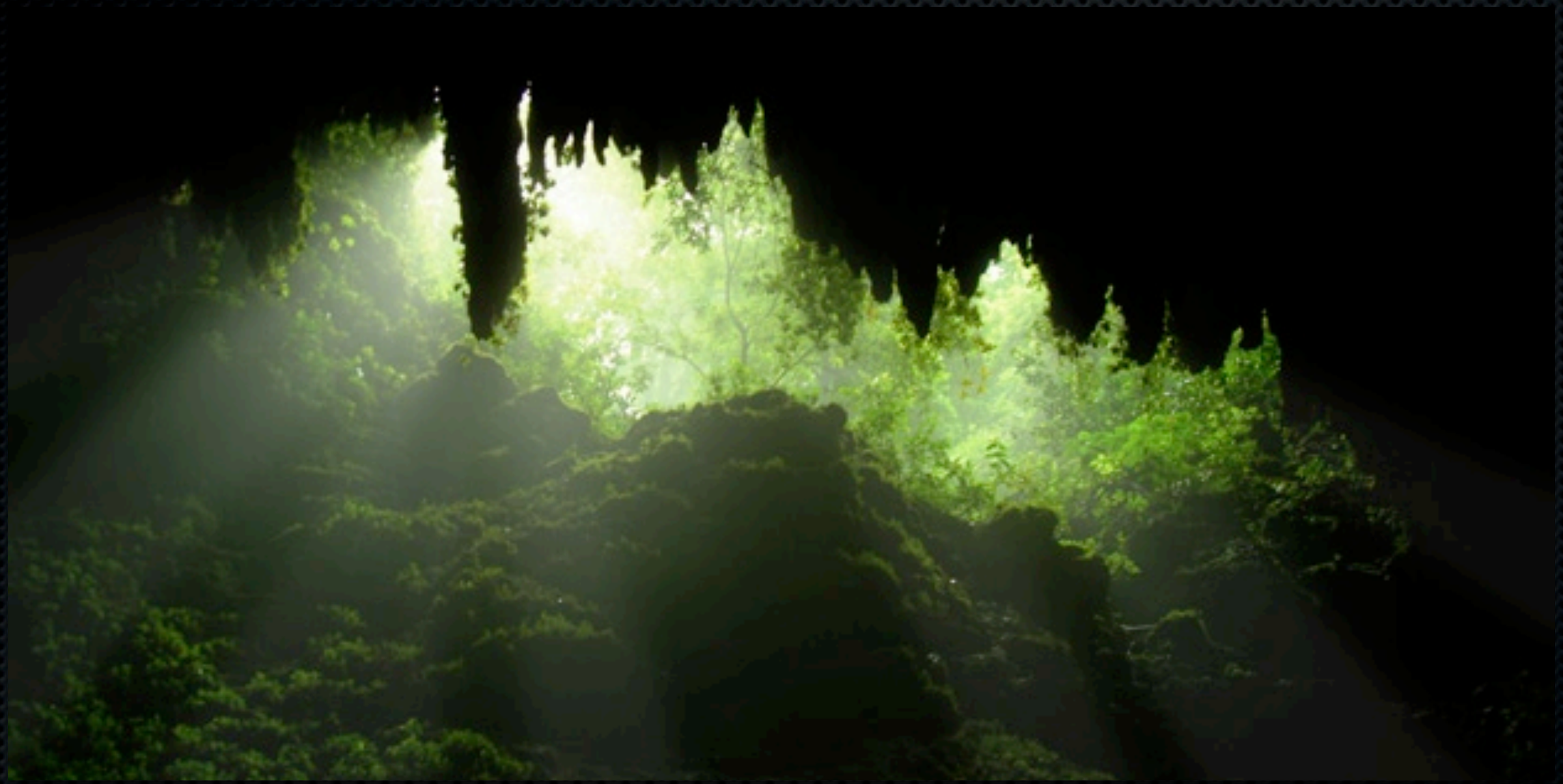
#increment(key,amount=1)
#decrement(key,amount=1)

```
cache.read("key")              # => nil
cache.increment("key")         # => nil
cache.write("key",20)          # => true
cache.increment("key")         # => 1
cache.read("key")              # => nil
cache.increment("key")         # => 2
cache.increment("key")         # => 3
cache.increment("key")         # => 4
cache.decrement("key")         # => 3
cache.increment("key",7)       # => 10
cache.decrement("key",5)       # => 5
```

MemCacheStore Specifics

- #clear – Performs a flush_all command on the memcached client. Deletes all data.
- #stats – Returns a server keyed hash built from each the memcached client's server connection.

```
>> cache.stats
=> {"localhost:11211"=>{"get_hits"=>1518, "bytes"=>212577,
"rusage_system"=>0.890843, "pid"=>18081,
"connection_structures"=>11, "accepting_conns"=>1, "threads"=>5,
"limit_maxbytes"=>536870912, "evictions"=>0, "cmd_flush"=>1,
"pointer_size"=>32, "time"=>1247589002, "version"=>"1.2.8",
"listen_disabled_num"=>0, "bytes_written"=>191926,
"total_items"=>2191, "cmd_get"=>1883, "total_connections"=>12,
"curr_connections"=>10, "uptime"=>77901, "cmd_set"=>2191,
"rusage_user"=>0.438621, "bytes_read"=>269582, "get_misses"=>365,
"curr_items"=>1998}}
```



Going Deeper...

Multiple Servers

- Client implements a #get_server_for_key which determines a server hash and who gets the data.
- Lets make another server:

```
$ memcached -u nobody -m 512 -p 11212 -U 11212 -d
```

```ruby
Rails::Initializer.run do |config|
  #...
  config.cache_store = [:mem_cache_store,'localhost:11211','localhost:11212']
end

(1..1000).to_a.each { |n| Rails.cache.write(n.to_s,'somedata') }

Rails.cache.stats.each do |server,stats|
  puts "TotalItems [#{server}]: #{stats['total_items']}"
end

# TotalItems [localhost:11211]: 446
# TotalItems [localhost:11212]: 554
```

Client Compression

- Done. Thanks Rails.
  Just use :compressed_mem_cache_store

```
module ActiveSupport
  module Cache
    class CompressedMemCacheStore < MemCacheStore
      def read(name, options = nil)
        if value = super(name, (options || {}).merge(:raw => true))
          if raw?(options)
            value
          else
            Marshal.load(ActiveSupport::Gzip.decompress(value))
          end
        end
      end

      def write(name, value, options = nil)
        value = ActiveSupport::Gzip.compress(Marshal.dump(value)) unless raw?(options)
        super(name, value, (options || {}).merge(:raw => true))
      end
    end
  end
end
```

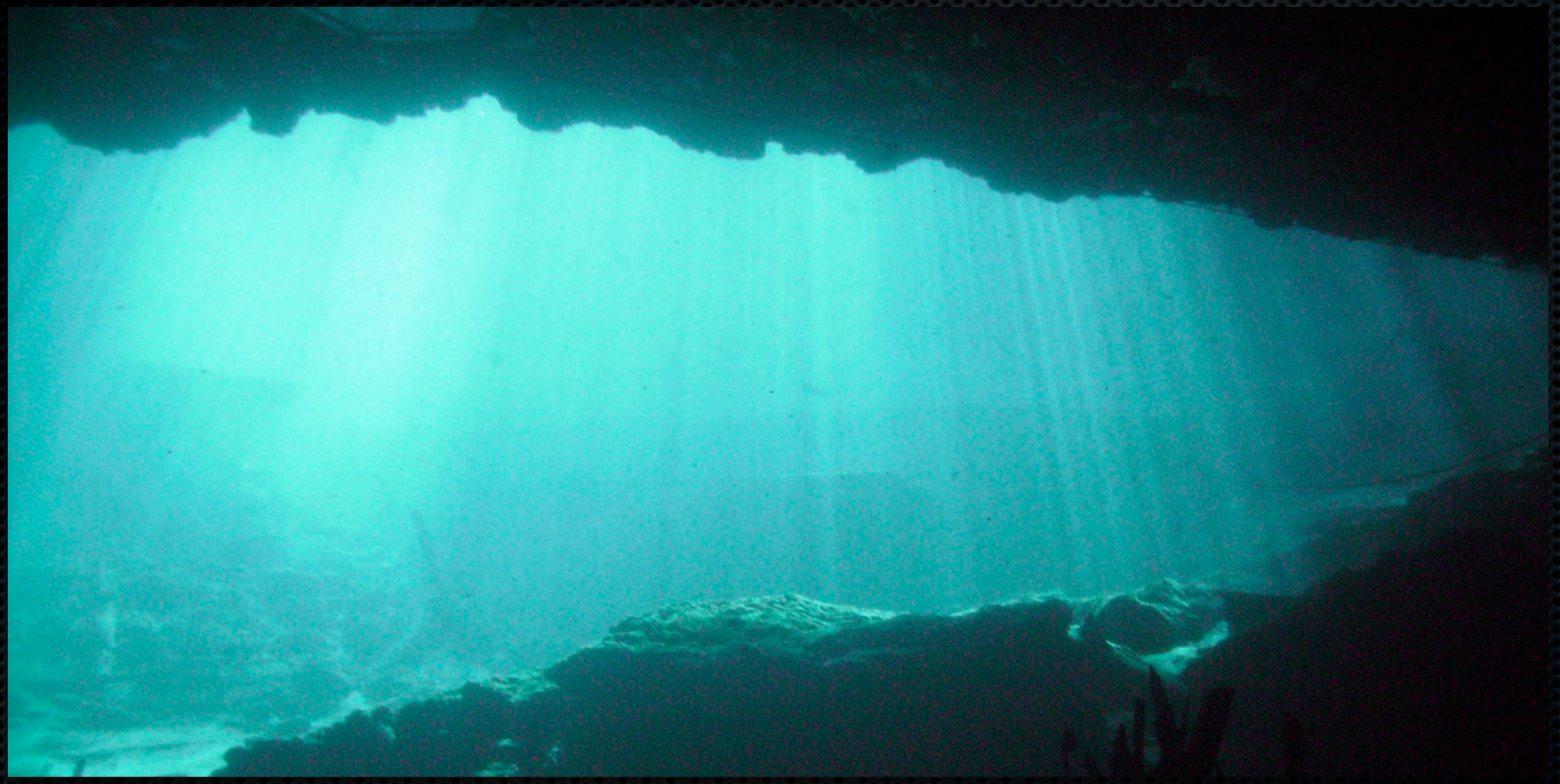
What To Cache?

- Basic types like strings, integers, arrays or hashes. The general rule is to consider the same objects that you might put in a session, ignoring size.

```
cache.write "myarray", [1,2,3]      # => true
a1 = cache.read("myarray")          # => [1, 2, 3]
a2 = cache.read("myarray")          # => [1, 2, 3]
a1.object_id == a2.object_id        # => false
cache.read("myarray") << 4          # => [1, 2, 3, 4]
cache.read("myarray")               # => [1, 2, 3]

class MyClass
  attr_accessor :this, :that
end

mo = MyClass.new                    # => #<MyClass:0x26c7198>
mo.this = [1,2,3]
cache.write "myobject", mo          # ture
mo1 = cache.read("myobject")        # => #<MyClass:0x26ae6c0 @this=[1, 2, 3]>
mo1.this                            # => [1, 2, 3]
```

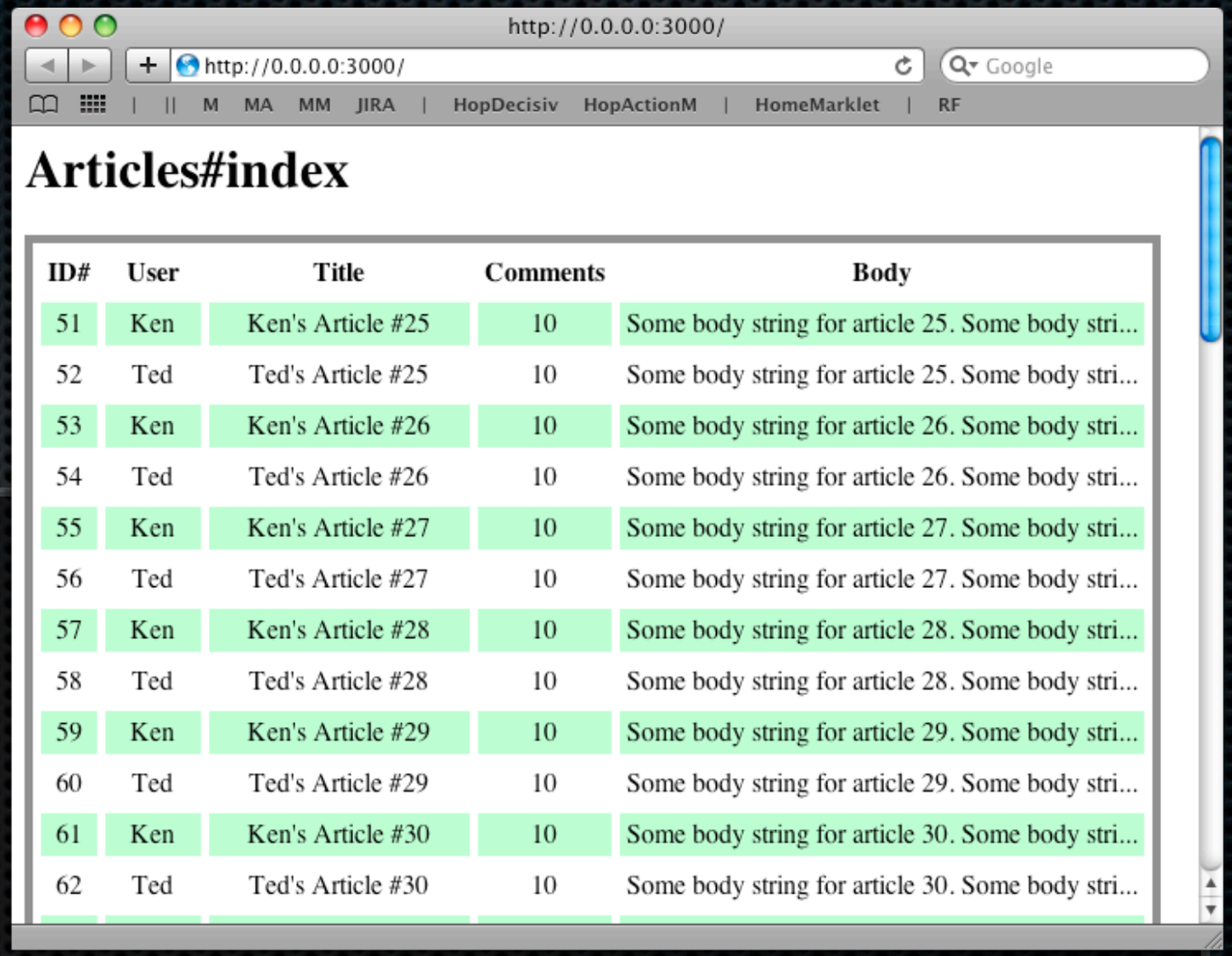


Caching In Action...

Live Discussion. Abstract:

- Review Nahum Wild's Article on Spandex Memcached which is not part of Rails 2.3.
  - http://www.motionstandingstill.com/spandex-memcache-store-is-now-in-rails-23/2009-02-04/
  - https://rails.lighthouseapp.com/projects/8994/tickets/1653-per-request-in-memory-cache-for-all-communication-with-the-memcache-servers
- Review Nahum Wild's Article on Starting Simple Rails Caching. Lesson is start with small line items vs page.
  - http://www.motionstandingstill.com/starting-simple-with-rails-caching/2008-11-27/
- Demostrate The Above With "play_cache" rails app. Included in this keynote with screen highlights on next slide

Demo App




```
<h1>Articles#index</h1>

<table>
  <tr>
    <th>ID#</th>
    <th>User</th>
    <th>Title</th>
    <th>Comments</th>
    <th>Body</th>
  </tr>
  <% @articles.each do |article| %>
  <tr class="<%= cycle('even','odd') %>">
    <% cache(['list',article]) do %>
    <td class="center"><%= article.id %></td>
    <td class="center w50"><%= article.user.name %></td>
    <td class="center w150"><%= article.title %></td>
    <td class="center"><%= article.comments.count %></td>
    <td><%= truncate(article.body, :length => 50) %></td>
    <% end %>
  </tr>
  <% end %>
</table>
```

Other Resources

- GUI Version Of Memcached For Mac
  - http://github.com/andrewfromcali/mcinsight/tree/master
- CacheMoney Rails Plugin
  - http://github.com/nkallen/cache-money/tree/master