

Ruby Betik Dili ve Ruby on Rails

Dr. Erek Göktürk
Identra Yazılım ve Donanım Çözümleri

www.identra.com.tr
erek @ identra.com.tr

Ajanda

- Yorumlanan diller ve derlenen diller
- Ruby'nin kısa tarihçesi
- Ruby kaynakları
 - Ruby dökümantasyonu
 - Ruby derleyicileri
- 1 saatte Ruby
 - Merhaba dünya
 - Ruby betiğini çalıştırmak
 - Değişkenler, veri tipleri, ve sabitler
 - Olağan deyimler
 - Fonksiyon tanımlama
 - Bloklar, yield, ve Proc'lar
 - Yineleme
 - Akış kontrol yapıları
 - Sınıflar, nesnelere
 - Değişkenler
 - Sabitler
 - Metodlar ve mesajlar
- Biraz da pratik yapalım
- Rails (onun kendi ajandası var)

Yorumlanan Diller ve Derlenen Diller

Yorumlanan Diller:

- Her çalıştırmada yeniden ayrıştırma
- Bölgesel (deyimle sınırlı) optimizasyon
- Daha kolay kodlama, test etme, değiştirme
- Prototip geliştirme ve gereksinimleri değişen projeler için özellikle uygun. Bu nedenle çevik metotlarda çok kullanılıyor.

Derlenen Diller:

- Tek sefer ayrıştırma
- Daha fazla optimizasyon
- Kodlama-Derleme-Çalıştırma ayrımı nedeniyle daha zor test etme ve değiştirme
- İç döngü geliştirme, veya gereksinimleri sabit kalacak projeler için uygun

Ayrıştırma : Parsing
İç döngü : Inner loop

Gereksinim : Requirement
Çevik metotlar : Agile methods

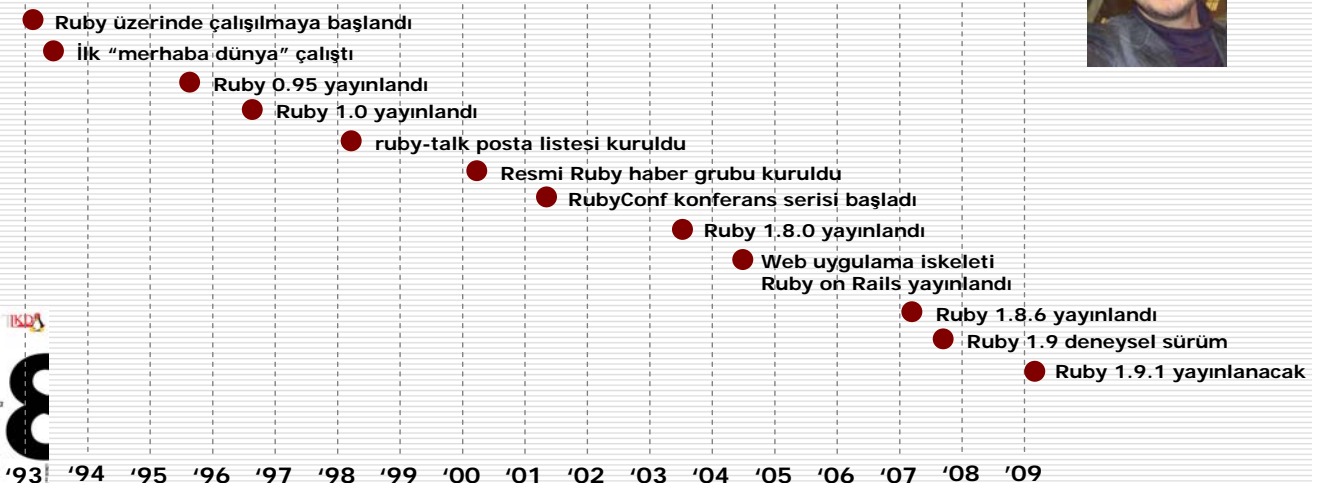
© Identra Yazılım ve Donanım Çözümleri
www.identra.com.tr

Erek Göktürk
April 20, 2009

3

Ruby'nin Kısa Tarihçesi

- Doğuş nedeni: Perl'den daha güçlü ve Python'dan daha nesne-yönelimli bir betik dili isteği.
- Yaratıcı: Yukihiro Matsumoto



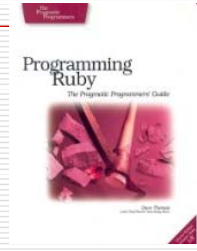
© Identra Yazılım ve Donanım Çözümleri
www.identra.com.tr

Erek Göktürk
April 20, 2009

4

Ruby kaynakları

- Programming Ruby: The Pragmatic Programmer's Guide (ilk sürümüne web'den ulaşılabilir: <http://www.rubycentral.com/book/>)



- Ruby Uygulama Arşivi (<http://raa.ruby-lang.org/>)

RAA - Ruby Application Archive

- RubyForge: Açık kaynak kodlu ruby projeleri (<http://rubyforge.org/>)

RUBYFORGE
by RubyCentral

- Ruby dökümantasyon projesi (<http://www.ruby-doc.org/>)

RUBY-DOC.ORG
Help and documentation for the Ruby programming language.

- Diğer bazı ruby kitapları

Ruby Yorumlayıcıları ve Derleyicileri

- MRI: Matz's Ruby Interpreter Yukihiro Matsumoto'nun referans yorumlayıcısı. 2.0 sürümünün YARV (<http://www.atdot.net/yarv/>) sanal makinesinde çalışması planlanıyor. (<http://www.ruby-lang.org/>)
- JRuby: Java ile yazılmış JVM üzerinde çalışan Ruby yorumlayıcısı. (<http://www.jruby.org/>)
- IronRuby: Microsoft .NET üzerinde çalışan, C# ile yazılan Ruby yorumlayıcısı. (<http://www.ironruby.net/>)
- Rubinius (<http://rubini.us/>)
- Ruby.NET (<http://rubydotnet.googlegroups.com/web/Home.htm>)
- XRuby: Ruby to java compiler (<http://xruby.com/default.aspx>)
- MagLev: Geliştirilmekte olan diğer bir Ruby için sanal makine (<http://ruby.gemstone.com/>)



Ruby
A Programmer's Best Friend



JRuby

Microsoft
IronRuby

Rubinius
Ruby, the Way It Was Meant To Be

RubyNET

XRuby
Bring Ruby to Enterprise





1 Saatte ⌚ Ruby

*Not:
Sanmayın ki Ruby burada
anlatılacaklardan ibaret!
Dahası var...*



Merhaba Dünya

```
puts "merhaba dünya!"
```

Merhaba dünya Ruby'de sadece bir satır.

Bir Ruby Betiğini Çalıştırmak

- MRI kullandığımızı varsayacağız.
- İki yol var:
 1. Ruby yorumlayıcısını çalıştırıp parametre olarak betik dosyasını vermek:

```
ruby merhaba_dunya.rb
```

2. Etkileşimli ruby (irb) arayüzünü kullanıp komutları ona yazmak:

```
> irb ←  
irb(main):001:0> puts "merhaba dünya" ←
```

Değişkenler

- Ruby'de değişkenler bildirilmek zorunda değil.
 - Tipleri yok
 - Değer atadığınız anda var olurlar
 - Kapsamları da ona göre belirlenir!
(Buna daha sonra tekrar gelelim)



```
i="Merhaba dünya!"  
puts i  
j="Ey koca dünya!"  
puts i+" "+j
```

Bildirim : Declaration
Kapsam : Scope

Sabitler

- ❑ Sabitler isimlerinin büyük harfle başlamasıyla ayrılırlar.
- ❑ Sabitlere sadece tek bir defa atama yapılabilir.
 - Yani, sabite atama yapan bir kod satırı sadece bir defa işletilebilir.

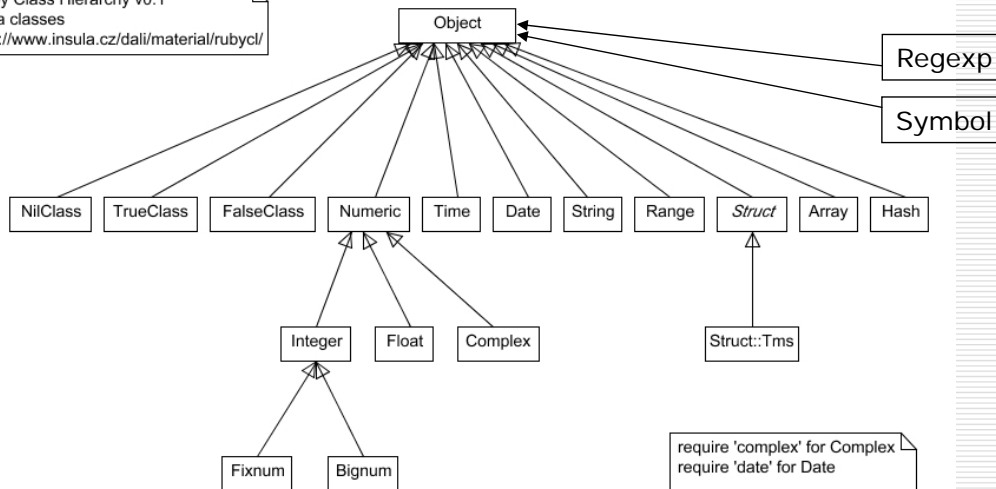
Ne fark var?



```
Sabit="Merhaba dünya!"
if false
  Sabit="Güle güle dünya!"
end
puts Sabit
```

Veri Tipleri – Herşey bir nesne

Ruby Class Hierarchy v0.1
Data classes
<http://www.insula.cz/dali/material/rubycl/>



```
irb(main):001:0> 3.kind_of?(Integer)
=> True
irb(main):002:0>
```

Dizgiler



- Dizgiler tek veya iki tırnakla gösterilir:

```
dizgi1="Merhaba dünya!"  
dizgi2='Merhaba dünya!'
```

- Aradaki fark şudur: Çift tırnakta
 - ters eğik çizgi ile özel karakterler verilebilir,
 - `{deyim}` sözdizimi kullanılarak bir ruby deyiminin sonucu dizgiye eklenebilir.

```
irb(main):001:0> puts "Ali dedi ki: #{dizgi1}\n\n"  
Ali dedi ki: Merhaba dünya!  
  
=> nil  
irb(main):002:0> puts 'Ali dedi ki: #{dizgi1}\n\n'  
Ali dedi ki: #{dizgi1}\n\n  
=> Nil  
irb(main):003:0>
```

Ters eğik çizgi : Backslash

Semboller

- Sözdizimi: `:isim`
- Dizgilerden farklı olarak sembollerde, aynı isim kullanıldığında her zaman aynı sembol nesnesi elde edilir.

```
irb(main):001:0> :a.object_id  
=> 323228  
irb(main):002:0> :a.object_id  
=> 323228  
irb(main):003:0> 'a'.object_id  
=> 70227975709660  
irb(main):004:0> 'a'.object_id  
=> 70227975709640
```



14 - 15



Aralıklar

- Sözdizimi: $(alt\ limit \dots üst\ limit)$
 $(alt\ limit \dots \dots üst\ limit)$

- Alt ve üst limitler
 - .. kullanıldığında aralığa dahil
 - ... kullanıldığında aralığa dahil değil
 - Karşılaştırma işlemi (\leq) ile karşılaştırılan ve **succ** metodunun tanımlı olduğu her sınıftan olabilir!

Köşenot:

inspect metodu her sınıfta tanımlıdır ve çağrıldığı nesnenin insan tarafından okunabilir bir gösterimini oluşturur.

```
irb(main):001:0> ('aab'..'aaf').to_a  
=> ["aab", "aac", "aad", "aae", "aaf"]  
irb(main):002:0>
```



16 - 17



Diziler (Array)

- Sözdizimi:
 $[0. eleman, 1. eleman, \dots, sonuncu\ eleman]$
- Elemanlar herhangi bir tipte olabilir.
- Bir dizide değişik tiplerde elemanlar bulunabilir.
- Diziler dinamiktir ve sınır testleri yoktur.

```
irb(main):001:0> a = [1, 2, "aad", 4.5]  
=> [1, 2, "aad", 4.5]  
irb(main):002:0> a[4]  
=> Nil  
irb(main):003:0> a[1]  
=> 2
```



18 - 19



Hashler

- Sözdizimi:
 $\{ \text{anahtar} \Rightarrow \text{değer} , \text{anahtar} \Rightarrow \text{değer} , \dots \}$
- Anahtarlar ve değerler herhangi bir sınıftan nesne olabilir!
- Farklı tiplerden anahtar ve değerler bir hash'in içinde beraber bulunabilir.

```
irb(main):001:0> a = {1 => 1.4, :kazuman => "aad"}  
=> {1 => 1.4, :kazuman => "aad"}  
irb(main):002:0> a[1]  
=> 1.4  
irb(main):003:0> a[:kazuman]  
=> "aad"
```

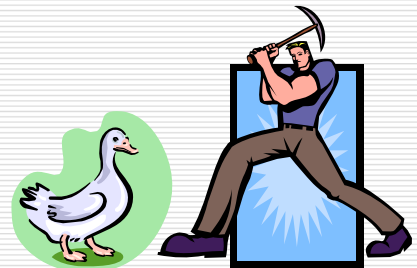


20 - 21

Olağan Deyimler (Regular Expressions)



- Olağan deyimlerin nasıl yazıldığına burada girmeyeceğiz. En azından basit düzeyde nasıl yazıldığını bildiğinizi varsayıyorum.
- Sözdizimi:
/olağan deyim / farklılaştırıcılar



```
irb(main):001:0> a = /kaz/  
=> /kaz/  
irb(main):002:0> "Baba bana kaz ve kazma al." =~ a  
=> 10
```



22 - 23



Fonksiyon Tanımlama

□ Sözdizimi:

```
def fonksiyon adı (parametre , parametre , ... ) ←  
    ruby deyimleri ←  
end
```

```
irb(main):001:0> def dizgiyap(bunu)  
irb(main):002:1>   bunu.to_s  
irb(main):003:2> end  
=> nil  
irb(main):004:0> dizgiyap([1,2,3])  
=> "123"
```



24 - 25



Bloklar

- Fonksiyonlar parametre olarak bir kod parçası alabilirler!
- Blokla fonksiyon çağırma sözdizimi:
fonksiyon adı (*parametreler*) { |*blok parametreleri*| ←
 ruby deyimleri ←
}
- Bloğun döndüğü değer, son işletilen ruby deyiminin döndüğü değerdir.
- { yerine **do**, } yerine **end** kullanılabilir.
- Blok kullanan fonksiyon sözdizimi:
def *fonksiyon adı* (*parametre* , *parametre* , ...) ←
 ruby deyimleri ←
 yield(*blok parametreleri*) ←
 ruby deyimleri ←
end



Bloklar (devam)

- ❑ Bloğun döndüğü değer, son işlenen ruby deyiminin döndüğü değerdir.
- ❑ { yerine **do**, } yerine **end** kullanılabilir.

```
irb(main):001:0> def dizgiyap(bunu)
irb(main):002:1>   yield(bunu).to_s
irb(main):003:2> end
=> nil
irb(main):004:0> dizgiyap(3) { |x| x+1 }
=> "4"
```



Bloklar (devam)

- ❑ Bloklar "closure" dediğimiz yapılardır: tanımlandıkları kapsamı beraberlerinde taşırlar.

```
irb(main):001:0> blockvar = 123
=> 123
irb(main):002:0> def stringify(this)
irb(main):003:1>   puts yield.to_s, " this is #{this}"
irb(main):004:2> end
=> nil
irb(main):005:0> stringify(3) { blockvar }
123 this is 3
=> nil
irb(main):006:0> blockvar = 432
=> 432
irb(main):005:0> stringify(3) { blockvar }
432 this is 3
=> nil
```



Proc'lar

- ❑ Bloklar Proc nesnelere dönüştürülebilir. Böylece
 - ➔ Bir fonksiyona birden fazla blok parametre olarak verilebilir.
 - ➔ Kod parçaları değişkenler içinde tutulabilir.
- ❑ lambda: Bloktan proc yaratır.

```
irb(main):001:0> dizgiyap = lambda do |bunu|
irb(main):002:1*   bunu.to_s
irb(main):003:1> end
=> #<Proc:0x00007f7be731aff70@(irb):0>
irb(main):004:0> dizgiyap.call([1,2,3])
=> "123"
```



Yineleme

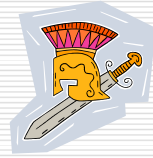
- ❑ Kaplar (diziler, hashler, dizgiler) ve bazı diğer sınıflarda yineleme amacıyla kullanılan bazı metodlar tanımlı.
- ❑ Yineleme, yani elemanların üzerinden tek tek geçme blok alan fonksiyonlar kullanarak yapılıyor!
- ❑ each: Verilen bloğu kaptaki her elemanla sıra ile çağırır.

```
irb(main):001:0> (0..3).each { |e| puts e.to_s }
0
1
2
3
=> (0..3)
irb(main):002:0>
```

Yineleme (devam)

- each gibi metodlar sınıflar arasında bazı farklılıklar gösterir.

```
irb(main):001:0> a = { :ben => :hur, :sparta => :kus }
irb(main):002:0> a.each { |an, d|
irb(main):003:1*   puts "#{an.to_s}#{d.to_s}"
irb(main):004:1> }
spartakus
benhur
=> { :sparta => :kus, :ben => :hur }
irb(main):005:0>
```



Yineleme (devam)

- Diğer bazı yineleme metodları ve benzer yapıda çalışanlar:
each_index, map, select, find
- Benzer ama daha ilginç bir metod: inject

```
def toplam(dizi)
  dizi.inject(0) { |top, e| top+e }
end
```

- Berim teorisi (theory of computation) almış olan varsa inject fonksiyonu tanıdık gelecektir (μ primitive function).

Akış Kontrolü

- Sözdizimi:

```
if koşul [then] ←  
  ruby deyimleri ←  
elsif koşul [then] ←  
  ruby deyimleri ←  
else ←  
  ruby deyimleri ←  
end
```

- If'in koşul doğru değilse deyimleri işleteni de var: unless. Sözdizimi:

```
unless koşul [then] ←  
  ruby deyimleri ←  
else ←  
  ruby deyimleri ←  
end
```

- If ve unless, tek bir deyimle ve else bölümü olmadan yazılacaksa değiştirici (modifier) formunda da kullanılabilirler. Sözdizimi:

```
ruby deyimi if koşul  
ruby deyimi unless koşul
```

Akış Kontrolü (devam)

- Seçenekli işletim: case. Sözdizimi:

```
case deyim ←  
when karşılık [, karşılık]... [then] ←  
  ruby deyimleri ←  
when karşılık [, karşılık]... [then] ←  
  ruby deyimleri ←  
...  
else ←  
  ruby deyimleri ←  
end
```



43 - 45

Akış Kontrolü (devam)

Döngüler



□ Döngü: `loop do`
ruby deyimleri
`end`

□ Ön kontrollü koşullu döngüler:

`until koşul [do]`
ruby deyimleri
`end`

`while koşul [do]`
ruby deyimleri
`end`

ruby deyimi `until koşul`

ruby deyimi `while koşul`

□ Son kontrollü koşullu döngüler:

`begin`
ruby deyimleri
`end while koşul`

`begin`
ruby deyimleri
`end until koşul`

□ Yineleme: `for değişken in deyim [do]`
ruby deyimleri
`end`



46 - 47

Akış Kontrolü (devam)

Döngüler (devam)



□ `break`:

Döngüyü durdurur ve döngüden sonraki deyimden devam edilir.

□ `redo`:

Döngü koşulunu kontrol etmeden döngüyü yeniden işletir.

□ `next`:

Bir sonraki yenilemeye geçer.

□ `retry`:

Döngü koşulunu kontrol ederek döngüyü yeniden işletir.



Sınıflar ve Nesneler

□ Sınıf tanımlama sözdizimi:

```
class sınıf_ismi < üstsınıf
  ruby deyimleri
end
```

□ Initialize: Kurucu metodu

□ **super**: Bir metodun üstsınıftaki aynı isimli metodu çağırmak için kullandığı kelime.

□ **new**: Nesne yaratma metodu



Sınıflar ve Nesneler (devam)

```
class FenG
  def liste
    ["gamze", "ali", "erdem"]
  end
end

class FenG6 < FenG
  def liste
    super << ["bahar"]
  end
end

sınıf = FenG6.new()
sınıf.liste      # ["gamze", "ali", "erdem ", "bahar"]
```



52 - 53

Sınıflar ve Nesnelere (devam)



Sınıf, ve Nesne Değişkenleri, Sabitler

- Sınıf değişkenlerinin ismi @@, nesne değişkenlerinin ismi @, sabitlerin ismi büyük harfle başlar.
- Erişim:
 - Sabitlere sınıf dışından kapsam işlemi (::) ile erişilebilir.
 - Sınıf ve nesne değişkenlerine erişim için erişim metodlarının tanımlanması gerekir.



54 - 56

Sınıflar ve Nesnelere (devam)



Sınıf, ve Nesne Değişkenleri, Sabitler

```
class Sunum
  DILLER = ["turkce", "ingilizce", "tarzanca"]
  @@varsayilan_dil = DILLER[0]
  def initialize
    @dil = @@varsayilan_dil
  end
  def dil
    @dil
  end
  def dil=(d)
    @dil = d
  end
end

Sunum::DILLER # OK. ["turkce", "ingilizce", "tarzanca"]
Sunum::varsayilan_dil # Hata!
Sunum::dil # Hata!
emo_egitim = Sunum.new
emo_egitim.dil # OK. "turkce"
emo_egitim.dil = "rusca"
emo_egitim.dil # OK. "rusca"
```



Metodlar ve Mesajlar

- Ruby, Simula/Smalltalk tarzı bir yaklaşıma sahip:
 - Metod çağırısı = nesneye mesaj gönderme

```
class Sunum
  def dil
    @dil
  end
  def dil=(d)
    @dil = d
  end
  def self.varsayilan_dil
    @@varsayilan_dil
  end
  def self.varsayilan_dil=(d)
    @@varsayilan_dil = d
  end
end
```

```
Sunum::varsayilan_dil # OK. "turkce"
Sunum::varsayilan_dil = "rusca"
Sunum::varsayilan_dil # OK. "rusca"

emo_egitim = Sunum.new
bogazici_ders = Sunum.new
emo_egitim.varsayilan_dil = "turkce"
bogazici_ders.varsayilan_dil = "ingilizce"
emo_egitim.varsayilan_dil # OK. "ingilizce"

bogazici_ders.send(:dil) # OK. "turkce"
```

Ruby deniz derya..
Biraz da pratik yapalım..



egrep

egrep:

- Komut satırından bir olağan deyim ve bir dosya ismi alacak.
- Dosyada olağan deyimden uyduğu parçalar olan satırları yazacak.

Satır sayısı bahisleri açılmıştır.

```
File.open(ARGV[1]) do |dosya|
  dosya.each_line do |dosya|
    puts dosya_satiri if dosya_satiri =~ /#{ARGV[0]}/
  end
end
```

CSV'den XML'e

csv2xml:

- Komut satırından bir girdi, bir çıktı dosya ismi, bir de kayıt ismi alacak.
- Girdi dosyasında CSV biçiminde bulunan veriyi çıktı dosyasına XML olarak yazacak.

Satır sayısı bahisleri açılmıştır.

```
require 'csv'
csv = CSV::parse(File.open(ARGV[0]) {|f| f.read} )
fields = csv.shift.map { |fn| fn.strip }
File.open(ARGV[1], 'w') do |f|
  f.puts "<?xml version='1.0'>\n<records>"
  csv.each do |record|
    f.puts((0..(fields.length - 1)).inject(" <#{ARGV[2]}>") { |acc, c_idx|
      acc + "\n <#{fields[c_idx]}>#{record[c_idx].strip}</#{fields[c_idx]}>"
    } + "\n </#{ARGV[2]}>" )
  end
  f.puts '</records>'
end # End file block - close file
```

Ajanda

- RoR nedir?
- RoR'ın kısa tarihçesi
- RoR kaynakları
 - RoR dökümantasyonu
- Kısaca RoR
 - Bir RoR uygulamasının yapısı
 - Merhaba dünya!
 - MVC mimarisi
 - Veritabanından veri alalım
 - Scaffolding

RoR Nedir?

- Ruby on Rails bir web tabanlı, çok katmanlı, uygulama geliştirme çerçevesi.
- Açık kaynak kodlu
- Programcı mutluluğu için optimize edilmiş!



Alexa'da en yüksek sıralardaki rails kullanan siteler

1. scribd.com [328]
2. www.justin.tv [413]
3. www.hulu.com [510]
4. yellowpages.com [626]
5. www.urbandictionary.com [791]
6. twitter.com [904]
7. aboutus.org [961]
8. cookpad.com [971]
9. slideshare.net/ [1234]
10. kongregate.com [1256]

RoR'ın Kısa Tarihçesi

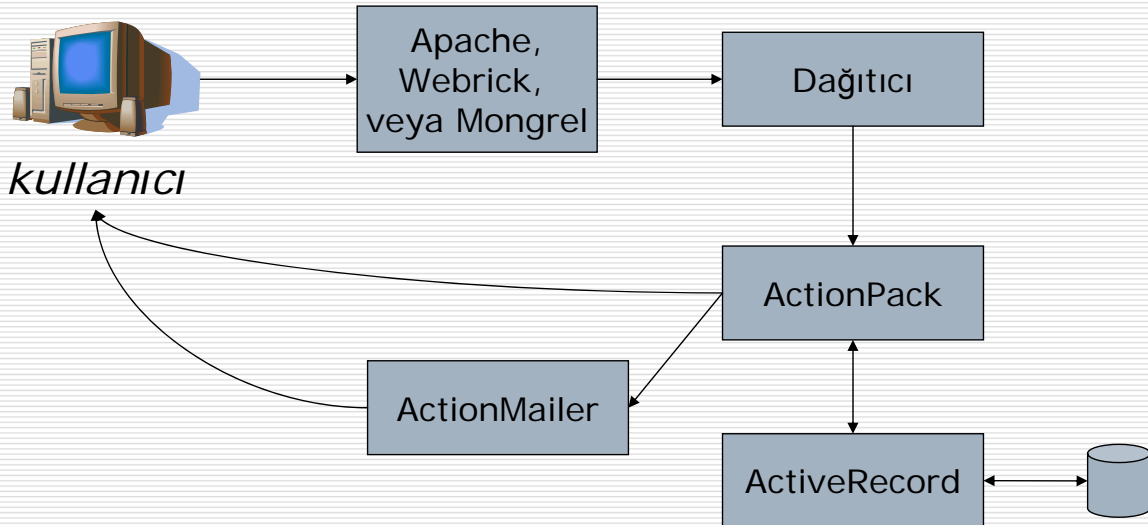
- 37signals şirketinin Basecamp'inde kullandığı sistemin web framework haline getirilmesi ile doğdu.
- RubyConf 2004'te ilk kez tanıtıldı.
- 2008'de 2.2. sürümü yayınlandı.

RoR Kaynakları

Resmi RoR sitesi:
<http://www.rubyonrails.org/>

RoR API dökümantasyonu:
<http://api.rubyonrails.org/>

Bir RoR Uygulamasının Yapısı



Merhaba dünya!

- Yapmamız gerekenler:
 - Rails uygulaması yaratacağız.
 - Controller yaratacağız.
 - Controller'ın index metodu istendiğinde içinde "Merhaba dünya" yazan bir html dosyası yaratıp kullanıcıya göndereceğiz.



MVC Mimarisi

- Model: Uygulamanın verileri ve bu verilerin dönüşümü ile ilgili kodlar
- View: Kullanıcıya göz kulak olmakla sorumlu kodlar
- Controller: View'lerle modelleri bir araya getirip orkestrasyonu yapan kodlar



Veritabanından Veri Alalım

- Yapılacak işler:
 - Bir model yaratacağız.
 - Bırakacağız rails database'de değişiklikleri yapsın.
 - Database'e biraz veri koyacağız.
 - Sonra üç dört satırda veriyi okuyup kullanıcıya göstereceğiz.



Scaffolding

- Daha iyisini de yapabiliriz.
- Diyelim ki çok hızlı bir şekilde prototip yaratmamız gerekiyor. Nasıl yaparız?

Scaffolding



Benden bu kadar.
Soru-cevap devam edebiliriz.