Deniz Kenarında Kısa Bir Gezinti: Seaside

Zekeriya KOÇ <zkoc@metkoi.com>

Özet

Bu yazı, http://www.seaside.st adresinde bulunan "A Walk On the Seaside" makalesinin çevirisidir.

İçindekiler

1 Seaside

Bu kılavuz Seaside 2.7 web geliştirme çatısına giriş niteliğindedir. Seaside kurulum bilgileri için http://www.seaside.st/Download/ adresini ziyaret edebilirsiniz.

Eğer Seaside'ı kendiniz kurarsanız (mesela SqueakMap kullanarak), sizden bir kullanıcı adı ve şifre girmeniz istenecektir. Bu bilgiler daha sonra bahsedeceğimiz Seaside yapılandırma uygulaması tarafından kullanılacaklar. Herşey kurulduktan sonra Seaside servisini başlatmanız gerekiyor. Seaside, Comanche web sunucusuna arayüz sağlamak için *WAKom* sınıfını sağlar. Seaside ile yapılandırılmış bir Comanche örneğini başlatmak için, aşağıdaki kodu bir Workspace içinde çalıştırın:

```
WAKom startOn: 9090
```

Seaside şu anda 9090 numaralı portu istekler için dinliyor olmalı. İmajınızı bu haliyle kaydederseniz, yeniden her açışınızda Seaside servisi otomatik olarak çalışacaktır.

2 Temel Kavramlar: Oturumlar ve Bileşenler

Seaside servisini test etmek için, tarayıcınızda "http://localhost:9090/seaside/counter" adresini açın. Bu karşınıza olabilecek en basit Seaside uygulamalarından birini getirir: Artırma ve azaltma işlemleri için bağlantılar içeren basit bir sayaç. Bu küçük uygulamayı birazdan kurcalayacağız ama şimdi sadece çalıştığından emin olmanız yeterli: Sayıyı artırmak için "++", azaltmak içinse "–" bağlantılarına tıklayın.

Sayaç uygulamasını denerken tarayıcınızın adres çubuğundaki URL'ye bakın. URL, bize birşeyler anlatabilecek iki adet sorgu parametresi içerir.

İlk olarak, uzun ve rasgele biraraya gelmiş gibi görünen, rakam ve harflarden oluşan bir parametre vardır (şuna benzer: *_s=ZXsNyDAuodddtmEG*) ve bu sorgu parametresi uygulamayı kullandığınız süre boyunca hiç değişmez. Bu anlık kullanıcı oturumunu işaret eden benzersiz bir anahtardır. Oturum kavramı, Seaside uygulamalarındaki merkezi kavramdır. Oturumları, sadece uygulama durumunu saklamak için bir depo olarak kullanan ve genelde uygulamaya özel istek/cevap döngüsüne odaklanan birçok web geliştirme çatısının tersine, Seaside, oturumları bir kanala(thread) ya da işleve(process) çok benzer bir şekilde ele alır: İyi tanımlanmış bir giriş noktasında oturumu başlatır ve uygulama, kullanıcıya siteyi göstererek ve kullanıcıdan giriş almak için bekleyerek, giriş noktasından doğrusal olarak ilerler. Kılavuzun devamında akış kontrolü konusunu tartışırken, bu kavramı daha yakından inceleyeceğiz.

Sayfanın alt kenarı boyunca uzanan gri çubuk üzerinde "New Session" isimli bir bağlantı göreceksiniz. Bu geliştirme esnasında gözüken Seaside araç çubuğudur. Bazı özelliklerini ilerledikçe kullanacağız. "New Session" bağlantısı, şu anki oturumu kolayca terketmenize ve sıfırdan yeni bir oturum oluşturmanızı sağlar. Eğer şu anda bu bağlantıya tıklarsanız, göreceğiniz tek şey; sayacın sıfırlanması ve URL'deki oturum parametresinin değişmesi olacaktır.

İkinci parametre (_k), daha kısadır ve rasgele harf ve rakamlardan oluşur. Bu parametre, belirli bir aksiyona işaret etmediği gibi oturum durumunuda saklamaz; tüm oturum bilgisi sunucu tarafında saklanır ve bir Seaside oturumu doğrusal olarak ilerler, o ya da bu aksiyona atlayarak değil. Bunun yerine oturumdaki isteklerini saklar ve Seaside'ın kullanıcının, uygulama boyunca yaptığı istekleri takip etmesini sağlar. Benzer şekilde sayfadaki her bağlantı ya da form alanına, benzersiz bir "id" numarası verilir. Bu numaralar sadece sunucuya veri döndüğünde anlam kazanır. Bunun bazı dezavantajları olsada karşılığında büyük bir esneklik sağlar.

Belirli sayfalar ya da aksiyonlar yerine Seaside uygulamaları, arabirimin bir kısmınının durumunu ve mantığını modelleyen ve birbirleriyle iletişim kurabilen, interaktif *bileşenlerden* oluşur. Bir oturum başladığında, bir bileşenin örneği oluşturulur ve bileşen cevap döngüsüne girer: kendisini kullanıcıya gösterir ve giriş için bekler. Bir giriş olduğunda (bir bağlantının tıklanması, bir formun gönderilmesi vs.), bileşenin karşılık gelen bir metodu çalışır ve metodun işi bittiğinde bileşen kendisini tekar kullanıcıya gösterir. Bu metotlar uygulamanın genel durumunu ya da bileşenin durumunu değiştirebilir ya da yeni bir bileşen oluşturup kontrolü ona devrederek yeni bileşenin kendi cevap döngüsüne girmesini sağlayabilirler.

Örnek sayaç uygulamasını modelleyen bileşenin adı *WACounter*. Şimdi bu sınıfa biraz daha yakından bakalım.

3 Durum, Aksiyon, Görünüm: Bileşen Temelleri

Her bileşenin üç sorumluluğu vardır. Arabirim durumunu korumak, kullanıcı girişine tepki vermek ve kendini html olarak göstermek. WACounter sınıfının bu sorumlulukların her birini nasıl yerine getirdiğini hızlı bir şekilde inceleyeceğiz.

3.1 Durum

Genelde bir uygulamanın durumunun çoğu kısmı bir iş nesnesinde ya da veritabanında saklansada, kullanıcı arabirimi genellikle kendi durumuna sahip olur. Bu, örnek olarak, bir form alanının şu anki değerini ya da hangi veritabanı kaydının gösterilmekte olduğu olabilir. Tüm bu bilgiler arabirimi oluşturan bileşenin durum (instance) değişkenlerinde saklanır.

Bu açıdan bakıldığında WACounter sınıfının koruması gereken tek durum sayaç rakamıdır. Oturumun başlangıcında, WACounter örneği ilk oluşturulduğunda, bileşen durum(instance) değişkeni olan "coun" değişkenini sıfıra eşitler. "++" ve "−" bağlantılarına tıklayarak *WACounter* sınıfının *count* durum (instance) değişkenini değiştirmiş olursunuz. Bu işlemi takip edebilmek için sayfanın altındaki "Toggle Halos" bağlantısına tıklayarak, ekstra hata ayıklama ikonlarını aktif hale getirebilirsiniz. Sayfanın üstünde beliren göz şeklindeki ikona tıklayın. Bu *WACounter* bileşenini içeren, web tabanlı bir "inspector" penceresi açacaktır. Bu sayfada *WACounter* nesnesinin üst sınıfından miras aldığı bir çok özelliğin yanında son sırada "count" özelliğinide göreceksiniz. Buradaki değişkenin değeri her zaman, "inspector" penceresini açtığınız anda, uygulamadaki değerin aynısı olacaktır.

3.2 Aksiyon

Sayaç uygulamasında, WACounter sınıfının karşılık gelen metodlarıyla gerçeklenen iki adet kullanılabilir aksiyon mevcuttur. "++" bağlantısına tıklamak, çok basit bir metod olan *#increase* metoduna bir çağrıya sebep olur. *#increase* metodunun tüm kodu:

```
increase count := count + 1
```

Bu metod tam da tahmin edebileceğiniz şeyi yapar: "count" durum (instance) değişkeninin değerini bir artırır. Dönüş yaptığında cevap döngüsü devam eder ve bileşen kendisini yeni "count" değeri ile tekrar gösterir.

Sadece eğlenmek için bunu değiştirmeyi deneyelim. Göz şeklindeki ikonun hemen solunda, sizi şu anki bileşenin sınıf tanımına götürecek, işlevsel ve web tabanlı bir "System Browser" penceresi açılır. *#increase* metodunu bulun ve bir yerine her tıklamada sayacı iki artırmasını sağlayın. "Accept" butonuna basıp "System Browser" penceresini kapatıp sayacı denerseniz değişikliğinizin hemen devreye girdiğini göreceksiniz.

3.3 Görünüm

Seaside bir bileşeni görüntülemek istediği zaman, ilgili bileşene bir *WAHtmlRenderer* nesnesini parametre olarak kullanarak, *#renderContentOn*: mesajını gönderir. Bu daha önce Java appletleri hazırlamış ya da Squeak içinde Morph sınıfından yeni bir sınıf türetmiş birine tanıdık gelecektir, şöyle ki; verilen nesne, kendisini görüntülemesi beklenen bir tuvaldir. Bizim durumumuzda "tuval", çizgiler ya da şekiller yerine HTML etiketlerini nasıl icra edeceğini bilmektedir. İcracı (renderer) bir akış gibi davranır. İcracıya gönderilen her mesaj icra edilmekte olan dökümana, yazı ya da eleman ekler. *WACounter* bileşeninin *#renderContentOn:* metodu şu şekildedir:

```
renderContentOn: html
```

```
html heading: count.
html anchor
  callback: [ self increase ];
  with: '++'.
html space.
html anchor
  callback: [ self decrease ];
  with: '–'
```

İcracıya herbiri farklı bir HTML elemanı oluşturan üç farklı mesaj gönderilmektedir. Bunlardan ilki, *#heading*: mesajıdır. Basit bir başlık bölümü oluşturur ve bunu oluşmakta olan dökümana ekler. Eğer sayacın değeri 42 ise, dökümanın sonuna "42" bilgisini ekleyecektir. #space yine basit bir mesajdır. Dökümana bir boşluk karakteri ekler. *#anchor* ise bunlara nazaran daha ilginçtir. Açıkça sayacın altında beliren "++" ve "–" bağlantılarını üretir. *with:* parametresi bağlantının görüntülenecek metninin belirtmektedir. Fakat bu bağlantılar hangi noktayı işaret etmektedir? Hangi hedefe giderler?

Bu sorulara kısaca verilebilecek cevap "Dert etmeyin" olabilir. Seaside çatısında bağlantıların varış noktaları yoktur. Yaptıkları çağrılar vardır. Bir düğme ya da bağlantı oluşturduğunuzda çağrı, bir kod bloğu ile ilişkilendirilir. İlgili düğme ya da bağlantı tıklandığında bu kod bloğu çalıştırılır. Bu durumda tahmin edebileceğiniz gibi "++" bağlantısına tıklanması *self increase* metoduna bir çağrıya sebep olacaktır.

Haydi bu metodu birazcık değiştirelim: İşlemler için bağlantılar kullanmak yerine, form düğmeleri kullanacağız. Squeak sınıf tarayıcısında *WACounter>>renderContentOn:* metodunu bulun ve aşağıdaki gibi değiştirin:

```
renderContentOn: html
  html form: [ html heading: count.
          html submitButton callback: [ self increase ]; text: '++'.
          html space. html submitButton callback: [ self decrease ];
text: '–' ]
```

Kodda yaptığımız temel değşiklik *#anchor* metoduna olan çağrıları *#submitButton* ile değiştirmek oldu. Her iki metod da aynı şekilde kullanıldığından, uygulamanızın arayüz gereklerine göre aralarında geçiş yapmak çok kolay olacaktır. Ama düğmeler sadece bir web fomunun içindelerse çalışırlar. *#form:* metodunun yapısı çok basittir, sadece bir kod bloğunu parametre olarak alır. Bu seferlik *#form:* metodunun bir çağrısı yok. Bunun yerine sadece form etiketlerinin içinde kalacak HTML elemanlarını belirlemek için kullanılıyor.

4 Cevap Döngüsü

Şimdiye kadar bir bileşenin sorumlulukları hakkında bilgi sahibi olduğunuza göre artık bunların birbirleriyle nasıl bağlantılı olduğuna geçebiliriz. Her bileşen bir cevap döngüsü çalıştırır; kendini görüntüle, kullanıcı girişi için bekle, giriş bilgilerini işle ve kendini tekrar görüntüle. Örneğimizde metodlara göre bir sıralama yaparsak şöyle bir liste çıkar karşımıza:

- *WACounter* sınıfının bir örneği oluşturulur. "sayaç" sıfır değerine yapılandırılır.
- Seaside *#renderContentOn:* metodunu çağırır. Bu, sayacın değerini ve bazı çağrılara bağlanmış bağlantılar içeren bir HTML dökümanı oluşrurarak kullanıcıya gönderir.
- Kullanıcı bağlantılardan birine tıklar. Bu *#increase* ya da *#decrease* metodlarından birini çağırır. Çağrı, sayacı yeni bir değere getirir. Aksiyona metodu kontrolü bileşene geri verir.
- Sayaç, yeni değeri ile yeniden görüntülenir.

5 Bileşenler Arası Etkileşimler

WAComponent sınıfı, uygulama kontrolünü başka bir bileşene devretmek için *#call:* adında özel bir metod sağlar. Bu metod, bir bileşeni parametre olarak alır ve anında kontrolü alan bileşenin kendisini görüntülemesi için cevap döngüsünü başlatır. Bunu test etmek için WAFormDialog bileşenini kullanabiliriz. Kullanıcının negatif sayılara geçmesi durumunda ona bir mesaj göstermek için #decrease metodunu biraz değiştirelim:

```
decrease count = 0
  ifFalse: [count := count - 1]
  ifTrue: [self call: (WAFormDialog new addMessage: 'Let"s stay away
from negatives.'; yourself)]
```

Eğer sayaç sıfır ise bu kod, WAFormDialog sınıfının bir örneğini oluşturur (bir bilgi mesajı ile birlikte) ve görüntüler. Denemek için yeni bir oturum başlatın ve "–" bağlantısına tıklayın. Sayaç uygulaması kaybolacak, büyük puntolarla bir uyarı mesajı ve bir buton görüntülenecektir.

Bunun gibi basit bir diyalog kutusu oluşturmak için WAComponent sınıfı, *#inform:* metodunu sağlar. *#decrease* metodunu yeni *#inform* ile şöyle değiştirebiliriz:

```
decrease count = 0
  ifFalse: [count := count - 1]
  ifTrue: [self inform: 'Let"s stay away from negatives.']
```

Mesajla birlikte "Ok" etiketli bir düğme görüntülendiğini farketmişsinizdir. Bu düğmeye tıklandığında ne olur? Diyalog gider ve sayaç uygulaması geri gelir. Arkaplanda, *WAFormDialog*, kontrolü çağıran bileşene geri veren *#answer:* metodunu çağırmıştır. Başka bir bileşeni çağırmak basit bir alt yordam çağrısı yapmak kadar basittir. Hoşunuza gidecekse, *#call:* metodunu, bir yığıta veri eklemek ve #answer metodunu da yığıttan veri almak gibi düşünebilirsiniz.

Gerçekte ortada bir yığıt yoktur. *#answer* metodunun yaptığı biraz daha komplike bir iştir. Gönderilen orjinal *#call:* mesajının geri dönüşünü sağlar ve uygulama *#call:* çağrısında kaldığı yerden devam eder. *#call:*, *#decrease* metodunun son işlettiği komut olduğundan, *#decrease* metoduda geri döner ve bileşen tekrar cevap döngüsünde kendisini görüntüler. Bu sezgisel olarak hemen kavranamayabilir. Bu yüzden meydana gelen işlemleri adım adım gösterelim:

- WACounter>>decrease metodu, WAComponent>>inform: metodunu çağırır.
- WAComponent>>inform: yeni bir WAFormDialog bileşeni oluşturur ve bunu WAComponent>>call: metoduna geçirir. Buraya "çağrı noktası" diyrelim.
- WAFormDialog kendi cevap döngüsüne girer ve kendisini görüntüler.
- Kullanıcı "Ok" düğmesine tıklar. Bu WAFormDialog bileşeninin WAComponent>>answer: metodunu çağırmasına neden olur.
- Şimdi işin ilginç kısmı: WAComponent>>answer asla dönüş yapmaz(return). Bunun sebebi programda bir atlama yapmış olmasıdır. Kontrol, #call: çağrısından hemen sonraya, "çağrı noktasına" geçer.
- WAComponent>>call:, kontrolü WAComponent>>inform: motoduna geçirir.
- WAComponent>>inform:, kontrolü WACounter>>decrease metoduna geçirir.
- WAcounter>>decrease, kontrolü sayaç bileşeninin cevap döngüsüne geçirir ve bileşen görüntülenir.

#call: metodu ile ilgili olarak etkileyici olan bir diğer özellikte, eğer çağrılan bileşen #answer: metoduna bir parametre geçirirse #call: metodu bu değeri döndürür. Bir diğer deyişle çağrılan bileşen bir sonuç değeri gönderebilir. Bu yaklaşım yığıta eleman ekleyip çıkartmaktan çok daha güçlüdür. Buna örnek olarak, kullanıcının tıkladığı düğmeye bağlı olarak "true" ya da "false" değerini döndürebilen bir bileşen oluşturan #confirm: metodunu uygulayabiliriz. Bunun için #decrease metodunu şu şekilde değiştirelim:

```
decrease count = 0
  ifFalse: [count := count - 1]
```

```
ifTrue: [(self confirm: 'Do you want to go negative?')
        ifTrue: [self inform: 'Ok, let"s go negative!'. count := -100]].
```

Şimdi uygulamayı kurcalarsanız göreceksiniz ki, bu tek metodun içinde 3 tane sayfa gösterimi var. Bir onay sayfası, bir mesaj sayfası ve ardından orjinal sayaç uygulaması (ekstra olarak arada tarayıcınızın "Geri" düğmesini kullanın ve sonuca bakın). Bu Seaside uygulamalarının tipik yapısıdır: bir sürü birbirini izleyen sayfalar yerine, her biri kendinden önceki ve sonraki sayfayı bilen, herbiri kullanıcıdan belirli bir bilgiyi alan ve ileten ve bunları uygulama mantığında tek bir yerde toplayan sayfalar. Bu yapının sonucu şaşırtıcı derecede yeniden kullanılabilir (reusable) kod parçaları olabilir.

#call ve #answer yöntemi Seaside bileşenlerinin yeniden kullanılabilirliklerini sağlamanın tek yolu değildir tabii ki. Farketmemiş olabilirsiniz ama ekranda her zaman en az iki bileşen bulunmaktadır. Bunlardan biri WACounter bileşenidir, peki ya öteki? Bunun cevabı bütün örnek boyunca gördüğünüz WACounter bileşeninin bir başka bileşen içinde gömülü durumda olduğudur. Bu, araç çubuğunu icra eden WAToolFrame sınıfının bir örneğidir. Bu tarz gömülü olma durumu Seaside içinde sık rastlanan bir durumdur. Genelde sayfalar iç içe geçmiş bileşenlerden oluşturulur. Gömülü bilşenler konusunun detayları bu kılavuzunu kapsamı dışında kalmaktadır ama basit bir örneği için içinde bağımsız bir sürü WACounter bileşeni barındıran, WAMultiCounter örneğini inceleyebilirsiniz. Eğer incelerseniz araç çubuğundaki "Toggle Halos" bağlantısını kullanın ve kurcalayın.