Spring ve Java EE 6

Spring ve Java EE 6 kıyaslamalarındaki belki en büyük hata bunların "ya hep ya hiç" mantığı ile yapılmalarıdır. Java EE içine pek çok değişik APInin dahil olduğu şemsiye bir spesifikasyondur. Spring ise bu pek çok API'nin büyük bir kısmı ile beraber problemsiz biçimde çalışmaktadır.

Dün, Bugün ve Yarın

J2EE veya yeni isimlendirmesi ile Java EE 6 önceleri ağırlıklı olarak EJB etrafında şekillenen bir spesifikasyon olmasına rağmen, zaman içerisinde bünyesine Servlet, JSP, JPA, JSF, JAX-WS gibi spesifikasyonları da dahil ederek bugün bir şemsiye spesifikasyona dönüşmüştür. İlk yıllarda özellikle EJB spesifikasyonunun vaaz ettiği hantal mimari ve programlama modeli yüzünden Java EE ile proje geliştirenler ciddi sıkıntılar yaşamışlardır. Tek veritabanı kullanılmasına rağmen JTA ve 2PC transaction yönetimi ile uğraşılması, web container ile EJB container'ların aynı fiziksel makinalarda olmalarına rağmen katmanlar arası iletişimin RMI ile yapılması, spesifikasyonun ORM, web servisleri, güvenlik gibi pek çok alandaki eksiklikleri ve muğlaklıkları nedeni ile çoğu proje gereksiz biçimde kompleks bir hal almış, geliştirme süreçleri sancılı hale gelmiş, üretim ortamında performans sorunları ile karşılaşılmıştır.

Spring'in temelleri ise 2004 yılında Rod Johnson'ın yazdığı, yukarıda bahsettiğimiz problemlere eleştirel bir yaklaşım getiren ve alternatif çözümler sunan "J2EE without EJB" kitabı ile atılmıştır. Kısa sürede kurumsal Java teknolojileri ile geliştirilen uygulamalar için vazgeçilmez bir uygulama çatısı haline gelmiştir. Spring sayesinde, kurumsal Java projelerinde EJB kullanarak bahsedilen problemlerle boğuşmanın şart olmadığı, POJO tabanlı bir programlama modeli ile test güdümlü çevik bir kurumsal yazılım geliştirme sürecinin projelere uygulanabileceği anlaşıldı. İlk çıktığı dönemde Spring, geliştiriciler arasında "glue-framework" olarak tasvir edilirdi. Ancak zaman içerisinde SpringSource firması ürün portföyünü uygulama sunucusundan, Spring uygulama çatısına, geliştirme ortamından, monitör ve yönetim araçlarına kadar çeşitlendirmiş, Spring de Java dünyasında "full stack" bir çözüm olarak algılanmaya başlanmıştır. Zaman içerisinde SpringSource monolitik olarak tanımladığı uygulama sunucularındaki değişik kabiliyetleri kendi ürün ailesi ile karşılar duruma gelmiştir.

Geçmişte EJB 1.0, 2.0 ve 2.1 spesifikasyonları ortaya konarken yapılan en büyük yanlışlık bu alandaki pratiklerden, olumlu ve olumsuz deneyimlerden ve ortaya çıkmış güzel çözümlerden faydalanılmamasıydı. Sonuç olarak da gerçek dünyadan kopuk, eksiklerle dolu ve kullanılması, uygulanması güç spesifikasyonlar ortaya çıkıyordu. Java EE "expert grup"ları hem 5 hem de 6 spesifikasyonlarının oluşturulmasında bu yaklaşımı terk etmiş görünüyorlar. Artık "expert grup"lar spesifikasyonları Spring, Hibernate, Seam, Guice gibi açık kaynak kodlu projelerde ortaya çıkan işe yarar fikirlerden yola çıkarak oluşturmaya gayret ediyorlar. Bu tür spesifikasyonların oluşturulması, Java teknolojileri ile yazılım geliştirenler arasında ortak bir platformun ve çözüm yapılarının oluşmasını sağlamaktadır. Bir nevi spesifikasyonlar Spring, Hibernate vb. açık kaynak kodlu çözümlerin devamında, bir sonraki aşamada ortaya çıkacak yeni innovatif çözümler için temel teşkil etmektedir. Aksi takdirde ortalıkta hiç spesifikasyon

olmasaydı Java dünyasındaki geliştiriciler için edinilmiş bilgi birikimi ve deneyimin diğer projelere aktarılması, taşınabilir çözümlerin ortaya çıkması zorlaşacaktır.

Gelinen noktada Java EE 6'daki yeniliklerin pek çoğunun Spring, Hibernate, Seam gibi çatılardan esinlenmelerle, test güdümlü yazılım geliştirme ve çevik metodolojilerle çalışma yaklaşımlarının sonucunda ortaya çıktığını söyleyebiliriz. Bu nedenle aslında Java EE 5'den sonra 6'nın biraz daha "Spring"leştiğini söylemek yanlış olmaz sanırım. Spring ilk çıktığında her katmanda farklı farklı teknolojilerin bir araya getirilip kullanılması, bu teknolojilerin kullanımlarının kolaylaştırılması gibi noktalardan da reklamını yapıyordu. Ancak 5 ve 6 spesifikasyonlarındaki yenilikler ve iyileştirmelerle Spring'in elinden bu ayrıcalık alınıyor diyebiliriz. Örneğin JdbcTemplate veya HibernateTemplate JDBC ve Hibernate ile çalışmayı oldukça kolaylaştırmalarına rağmen zaman içerisinde Hibernate 3 ve JPA 'nın ortaya çıkması ile HibernateTemplate ve JpaTemplate'in geliştiricilere sağlayabileceği fazla birşey kalmadı. Yine de şu an için bile bu teknolojileri kullanırken Spring'in veri erişim katmanı üzerinden kullanılmaları, JDBC ve ORM işlemlerinin aynı transaction içerisinden yürütülmesi ve lokal veri kaynağı kullanılması, exception hiyerarşisinin ortak bir yapı ile ifade edilmesi gibi noktalardan kolaylıklar arz etmektedir.

Bugün insanların yavaş yavaş Spring'e karşı cephe almalarının arkasındaki temel neden belki biraz da SpringSource'un içine girdiği dönüşümdür. SpringSource kısaca özetlemek gerekirse tc ve dm Server'lardan başlayarak geliştirme ortamına kadar kendi Java EE platformunu oluşturmaya yönelmiştir. VmWare firmasına satılmasıyla birlikte artık iyice kapalı bir sistem olduğu yönünde kaygılar had safhaya ulaşmıştır. Kurumsal Java dünyası ise spesifikasyon üzerinden ilerleyerek, değişik üreticilerin yer aldığı açık bir platform olarak kalmaya önem vermektedir. Ancak zaman zaman spesifikasyon üzerinden ilerlemenin getirdiği zorluklar ve yavaşlıklar da Java geliştiriciler için ciddi problemler doğurmaktadır. Spring ise spesifikasyon yükünü tam olarak üzerinde hissetmediği için güncel trendlere kendini çok daha kolay adapte edebilmektedir.

Spring mi, Java EE 6 mı?

Spring ve Java EE 6 kıyaslamalarındaki belki en büyük hata bunların "ya hep ya hiç" mantığı ile yapılmalarıdır. Java EE içine pek çok değişik APInin dahil olduğu şemsiye bir spesifikasyondur. Spring ise bu pek çok API'nin büyük bir kısmı ile beraber problemsiz biçimde çalışmaktadır. Hatta bu API'lerin kullanımını daha kolay ve daha verimli bir hale getirmeye gayret etmektedir. Tabi bunun yanında kendine has diğer pek çok kabiliyeti de geliştiricilere sunmaktadır.

Java EE 6'nın tasarımında öne çıkan özelliklerinden birisi "convention over configuration" yaklaşımının APIlerinde sistematik biçimde uygulanmasıdır. Java EE 6 API'lerini, örneğin EJB 3.1'i kullanmaya başladığınızda mevcut ayarlar sisteminizin ilk çalışması için yeterli olacaktır. Ne annotasyonlara, ne de herhangi bir XML konfigürasyon dosyasına ihtiyacınız vardır. Spring için ise durum tam tersidir. Daha doğrusu Spring'in yaklaşımında sistem size gizliden birşeyler daha sunmaya çalışmamaktadır. Siz ne isterseniz Spring o kadarını sağlamaktadır. Örneğin servis metodlarınızın transactional olması için Spring'e bunu açık biçimde belirtmeniz gerekir. EJB 3.1'de ise metodlar varsayılan durumda halihazırda transactionaldır. Ancak Spring size bu

konfigürasyonları tekrar tekrar yapmak yerine bir kere yapıp, müteakip durumlarda yeniden kullanabilmenizi de mümkün kılmıştır. Örneğin, @Service steryotipini genişleterek kendinize özel servis annotasyonu oluşturup, metodlarınızın varsayılan durumda transactional olmasını sağlayabilirsiniz. Her iki yaklaşımın da kendine has artı ve eksileri vardır. "Convention over configüration" yaklaşımı sistemin ilk oluşturma ve çalıştırma süresini kısaltmasına rağmen, mevcut yapıyı da bir kalıp içerisine sokmaktadır. Örneğin, eğer EJB'lerinizi varsayılan durumda transactional yapmak istemiyorsanız bu sizin için problem olacaktır. Spring için ise uygulamanın ilk oluşturulma ve çalışma safhası daha uzun olmakta, ancak uygulamanın yapısı üzerinde her türlü farklılandırmaya gitmek de daha kolaydır. Örneğin, istediğimiz zaman Spring tarafından yönetilen servis beanlarımızı transactional yapabiliriz, istediğimiz zaman da bunların transactional özelliklerini kaldırabiliriz.

Spring sayesinde yaygınlaşan, bağımlılıkların container tarafından yönetilmesi yani "dependency injection" yaklaşımı bu versiyonda spesifikasyona "Context and Dependency Injection" (CDI) adı ile girerek, EJB ve container tarafından yönetilen diğer kaynakların yanında POJOlar için de uygulanabilir hale gelmiştir. Dependency injection'ın JavaEE spesifikasyonuna girmesi ile birlikte Spring'in xml ve annotasyonlarının standart dışı olması onun hanesine eksi puan olarak yazılmaya başlamıştır. İnternet üzerindeki blogların ve yazıların pek çoğunda da temel eleştirilerden birisi Spring'in XML tabanlı konfigürasyonudur. Oysa ki uzun bir zamandır neredeyse hiç XML kullanmadan Spring bean'larını tanımlamak ve kullanmak mümkündür. Annotasyon tabanlı veya programatik bean konfigürasyonu kabiliyetleri Spring 3 ile daha da geliştirilmiştir. Ayrıca Dependency injection'ı sadece bir bean'ı başka bir bean'a enjekte etmek olarak görmek doğru olmaz. Nesneler arası bağımlılıkları oluşturma ve birbirine enjekte etme yöntemlerindeki çeşitliklik Spring tarafında daha fazladır. Hatta @Configurable annotasyonu ve AspectJ sayesinde Spring tarafından yönetilmeyen nesnelere bile "dependency injection" yapmak mümkündür. CDI içerisinde yer alan scope'lar ise Java EE'yi Spring'e kıyasla öne çıkarmaktadır. Özellikle web programcılarının uzun zamandır ihtiyacını duyduğu ve Seam, WebFlow gibi çatılarla dışarıdan projelerine dahil ettikleri "conversation scope" desteği spesfikasyona dahil edilmiştir. Spring de conversation scope ve tarayıcı pencere yönetimini 3.1 sürümünde kullanıcılarına sunmayı planlamaktadır.

Neredeyse bir asırdır bekliyoruz diyeceğimiz web.xml'in parça parça tanımlanabilmesi ve profil desteği spesifikasyondaki temel yeniliklerden bir diğeridir. Bu sayede her proje tipine göre bir takım teknolojilerin bir araya getirilmesi ile oluşturulacak konfigürasyon yığıtları mümkün olabilecektir. Java EE 6 web profili sadece Servlet ve JSP gibi APIleri içeren hafif sıklet profildir. Bunun üzerine EJB 3.1, JTA, JPA, JSF ve WebBeans konarak bir diğer profil oluşturulmaktadır. Bu profilin üzerine de JMS, JAX-WS gibi APIler eklenerek bütün platformu kapsayan tam bir profil sunulmaktadır. Java EE profilleri, Spring'in neye ne kadar ihtiyacınız varsa o kadarını kullanın yaklaşımının spesifikasyona bir yansımasıdır diyebiliriz.

İlgi yönelimli programlamanın (AOP) uygulama geliştirmede kullanılması açısından Spring, spesifikasyona göre çok daha öndedir. Java EE 6 ile "interceptor" desteği sağlanmış ve "around advice" bir biçimde karşılanır olmuştur. Ancak AspectJ gibi bir AOP dilinin doğrudan projeniz içerisinde kullanımı ve bu AOP dilinin bütün kabiliyetlerinden yararlanmak Spring ile çok daha kolaydır. Bununla beraber spesifikasyonla birlikte gelen "metod interceptor"ler, annotasyonlarla ilgili sınıfla ilişkilendirildiklerinden iki yapı arasında sınıf düzeyinde bir bağımlılık ortaya çıkarmaktadır. Bu da farklı nesnelerin farklı interceptorler ile

ilişkilendirilmelerine engel teşkil etmektedir. Aslında bu annotasyon merkezli konfigürasyonların temel bir zaafıdır.

Java EE 6'nın Spring ile arasındaki mesafeyi oldukça kapadığı doğrudur, ancak hala bazı konularda Spring'in açık ara önde olduğunu söyleyebiliriz. Bunlardan birisi de güvenliktir. Önceleri "Acegi Security Framework" adı ile ortaya çıkan ve Spring üzerine kurulu, web uygulamaları için güvenlik çatısı olan bu framework daha sonra Spring'in ürün ailesi içerisinde dahil edilmiştir. Spring Security olarak tanınan çözüm, kurumsal web uygulamalarındaki kimliklendirme ve yetkilendirme ihtiyaçlarına spesifikasyondan çok daha iyi ve platform bağımsız çözümler sunmaktadır. Java EE spesifikasyonu hala güvenlik konusundaki açıkları ve belirsizlikleri giderip, containerdan bağımsız bir hale gelememiştir. Spesifikasyondaki güvenlik mekanizması ile geliştirdiğiniz uygulamanızı Weblogic, Websphere ve Tomcat gibi üç ayrı container'a deploy etmeye çalışırsanız, ne demek istediğimizi daha kolay anlayabilirsiniz. Bütün bunlara ilaveten FactoryBean, post processor'lerin, constructor injection'ın, list ve map tanımlamalarının ve geliştirme sürecinde ihtyaç duyulan daha pek çok yardımcı sınıfın Spring'le beraber olması hala onu Java EE spesifikasyonundan birkaç adım öne koymaktadır.

Sonuç

Java dünyasının spesifikasyonlardan en büyük beklentisi, ilgili teknolojilerde standartlaşmayı sağlamasıdır. Ancak spesifikasyonların, hemen herkes tarafından gerekli görülen bazı özellikleri üreticilerin kararına bırakması bu standartlaşmayı zorlaştıran durumlardan birisidir. Örneğin JPA 1.0 içerisinde Criteria API'nin yer almamış olması JPA kullanan her projenin pratikte alttaki ORM gerçekleştirimine bağımlı olması demekti. Bu durum ancak JPA 2 spesifikasyonunda giderilebilmiştir. Güvenlik tarafında da spesifikasyonun açık bıraktığı pek çok konu vardır. Dolayısı ile kurumsal uygulamaların kimliklendirme ve yetkilendirme ihtiyaçlarında ya üreticiye bağımlı olmaları, ya da kendilerine has çözümler geliştirmeleri kaçınılmaz olmaktadır. Sonuçta kurumsal projelerin farklı üreticiler arasında geçişler sağlayabilecek oranda platform bağımsız geliştirilmesi pratikte mümkün olamamaktadır. Bütün bu standartlaşma çabalarının, Spring geliştiricileri ve Java "expert grup" arasındaki mücadelenin biz kurumsal Java teknolojileri ile yazılım geliştirenler açısından yararlı sonuçlar doğuracağını söyleyebiliriz. Kısacası nasıl ki JPA yeni Hibernate oldu, Java EE 6, belki Java EE 7 de yeni Spring olma yolunda ilerliyor diyebiliriz.

Yazar: ODTÜ Bilgisayar Mühendisliği'nden 1999 yılında mezun olan Kenan Sevindik o dönemden bu yana pek çok büyük ölçekli kurumsal yazılım projesinde görev yapmıştır. Halihazırda kurumsal Java teknolojileri ile yazılım geliştirme, eğitim ve danışmanlık hizmeti sunan yazarımız, değişik ortamlarda teknoloji içerikli konuşma ve sunumlar da yapmaktadır. Edindiği bilgi birikimi ve deneyimleri http://www.harezmi.com.tr/blog adresinden sanal alemde sizlerle paylaşmaktadır. Kendisi ile iletişime geçmek için ksevindik@gmail.com adresine mesaj gönderebilirsiniz.