JNDI

Yazan : Burhaneddin Volkan

HACETTEPE ÜNİVERSİTESİ

İSİMLENDİRME VE DİZİN KAVRAMLARI 4
1.İsimlendirme kavramları 4
İsimler 4
Bağlantılar (Bindings) 4
Referanslar ve Adresler 5
İçerik (Context) 5
İsimlendirme sistemleri ve İsim Uzayları 5
Dizinleme Kavramları 6
Öznitelikler 6
Dizinler ve dizinleme servisleri 6
Aramalar ve arama filtreleri 7
İsimlendirme ve Dizinleme servislerini birleştirme 7
Dizin-etkin Java uygulamaları 7
Dizinlemenin geleneksel kullanımı 7
Bir nesne deposu olarak dizin 7
JNDI ' a genel bakış 8
Mimari 8
Paketleme 8
İsimlendirme paketi 9
İçerik (Context) 9
İsimler 9
Bağlantılar (Bindings) 10
Referanslar 10
Başlangıç İçeriği 10
Aykırı Durumlar 10
Dizinleme Paketi 10
Dizin İçeriği 11
Aramalar 11
Olay Paketi 11
Olaylar 11
Dinleyiciler (Listeners) 11
LDAP Paketi 12
Uzatılmış İşlemler (Extended Operations) 12
Kontroller (Controls) 12
Talep Edilmemiş bildirimler (Unsolicited Notifications) 12
LDAP içeriği 12
Servis sağlayıcı paketi 13
Tak-Çalıştır Mimarisi (Plug-in Architecture) 13
Java Nesne Desteği (Java object support) 13
Çoklu İsimlendirme Sistemleri (Federasyonlar) 13
GEREKLİ YAZILIM 13
Java Platform Yazılımı 14
JNDI yazılımı 14
İsimlendirme ve Dizinleme Sunucu Yazılımı 14
ÖRNEKLER 14
İsimlendirme Örneği 14
JNDI sınıflarını dahil etmek 15
Bir başlangıç İçeriği oluşturmak 15
Bir Nesneye bakmak 15

NamingException Yakalamak 15
Programı derlemek 16
Programı Çalıştırmak 16
Dizinleme Örneği 16
JNDI dizinleme sınıflarını dahil etmek 16
Bir Başlangıç İçeriği Oluşturma 16
Bir Dizin Nesnesinin Özniteliklerini Alma 17
İstenen Özniteliğe Ulaşma 17
NamingException Yakalamak 17
Programı derleme ve çalıştırma 17
Paketler ve ClassPath 18
JNDI Sınıflarını Dahil etmek 18
Derleme Ortamı 18
Çalışma Ortamı 19
Naming Aykırı Durumları 19
Aykırı Durum Sınıf Hiyerarşisi 19
Numaralandırmalar (Enumerations) 19
javax.naming[api] Paketindeki Aykırı durumlar 20
javax.naming.directory[api] paketindeki aykırı durumlar 20
javax.naming.ldap[api] paketindekiaykırı durumlar 21
Başlangıç İçeriği (Inıtıal Context) 21
Servis sağlayıcısını seçmek 21
Başlangıç İçeriğine Gereken Bilgiyi Sağlamak 22
Başlangıç İçeriğini Oluşturmak 22
İsimler 23
İsimlendirme işlemleri 23
Bir nesneye bakmak 23
İçeriği listeleme 24
Context.list Metodu 24
Context.listBindings Metodu 25
Bir NamingEnumeration nesnesini sonlandırma 26
Bir bağlantıyı Ekleme silme günleme 27
Bir nesneyi yeniden adlandırma 27
Alt İçerikler Oluşturma ve Silme 28
Dizinleme İşlemleri 28
Kurulum 29

İSİMLENDİRME VE DİZİN KAVRAMLARI

1.İsimlendirme kavramları

Bilgisayar sistemlerindeki temel yeteneklerden biri de isimlendirme(*naming*) servisleridir. Bir vesileyle Nesneler isimlere bağlıdır veya nesneler isimlerle bulunur. Hemen hemen tüm programlarda veya sistemlerde nesneleri isimlendirirsiniz. Mesela elektronik bir mesaj sisteminiz olduğunu düşünelim. Mesaj atmak istediğiniz alıcının ismini sağlamak zorundasınızdır. Bilgisayarda bir dosyaya erişirken ismini sağlamanız gerekir.

Bir isimlendirme servisinin temel görevi adres belirleyici gibi nesnelere yada en basit olarak bilgisayardaki programların kullandığı nesnelere isim vermektir. Mesela Etki Alanı İsimlendirme Sunucusu (Domain name server) makine isimlerini (www.sun.com gibi) IP adresleriyle (192.9.48.5) eşleştirir. Bir Kütük sistemi (file system) kütük isimlerini (c:\openldap\slapd.exe gibi) programların kullanabileceği kütük işleçleriyle (file handle) eşleştirir. Bu iki örnek isimlendirme servislerinin faklı kullanımlarını gösterir. Internet'te nesne isimlendirmek veya bir bilgisayarda yerel olarak bir kütüğü isimlendirmekte prensipte bir fark yoktur.

İsimler

Bir isimlendirme sisteminde bir nesneye bakmak için o nesnenin ismini (*name*) sağlarsınız. İsimlendirme sistemi ismin uyması gereken sözdizimi (syntax) kurallarını belirler. Bu sözdizime bazen isimlendirme sisteminin isim dönüştürmesi (*naming convention*) denir.

Mesela Unix kütük sisteminin isim dönüştürmesinde bir dosyanın ismi kütük sisteminin köküne (*root*) göreli olarak çevrilir. Kütük yolundaki her bileşenin ismi taksim ("/") karakteriyle (forward slash) ayrılır. Mesela Unix'te kütük adı /usr/Merhaba usr dizinindeki Merhaba kütüğünü gösterir. Burada usr dizini sistemin kökünde yer alır.

DNS isim dönüştürmesinde bileşenler sağdan sola sıralıdır ve nokta(".") işaretiyle ayırtılır. DNS ismi 'gmail.google.com' 'gmail' adlı bir DNS ismini 'google.com' ismine göreli tutar. 'google.com' ismi ise 'com' isminin girişine göreli tutulur.

Tüy sıklet dizin erişim protokolünde(Lightweight Directory Access protocol ,LDAP) ise İsim dönüştürme işlemi bileşenleri sağdan sla sıralı tutar ve virgül (",") karakteriyle ayırır. Nitekim LDAP ismi 'cn=Nil Karaibrahimgil, o=Sun, c=TR' ; 'cn=Nil Karaibrahimgil' ismini o=Sun ismine onu ise c=TR ye göreli tutar. LDAP'ın bir diğer kuralı da her bileşenin bir isim/değer ikilisinden oluşmasıdır. İsim ve değer ise birbirinden eşittir ("=") karakteriyle ayrılır.

Bağlantılar (Bindings)

Bir isimle bir nesnenin ilişkilendirilmesine bağlama (binding) denir. Mesela bir kütük ismi bir kütük ile bağlantılıdır.

DNS makine isimleriyle IP adreslerinin bağlantılarını tutar. Bir LDAP ismi bir LDAP girişine(entry) karşılık gelir.

Referanslar ve Adresler

isimlendirme servisinin türüne göre bazı nesneler doğrudan saklanmazlar. Bunun yerine referansları saklı tutulur. Bir referans (*reference*) bir nesneye nasıl ulaşılacağı konusunda bilgi saklar. Referansları kullanarak enseyle iletişime geçebilir ve nesne hakkında daha çok bilgi edinebilirsiniz.

Mesela bir uçak nesnesi yolcuların bir listesini mürettebatı ve uçuş planını tutsun. Buna karşın bir uçak referansının tutması gereken sadece uçuş numarası ve varış zamanı olabilir. Referanslar nesneler hakkında daha yoğun(compact) bilgiye sahip olup bu bilgilerle diğerleri bulunabilir.

Mesela bir kütüğe bir kütük işlecini referans kabul ederek erişirsiniz.
Bir yazıcı nesnesi yazıcı hakkında her şeyi tutarken yazıcı referansı ise ona nasıl ulaşılacağı – yazıcı sunucusu ve yazıcı protokolü gibi- bilgisini tutabilir.

Her ne kadar referanslar isteğe bağlı bilgi tutabilirlerse de referans olarak nesnelerin adreslerine(iletişim ayakları) başvurmak daha kullanışlıdır. Adresler (*addresses*) bir nesneye ulaşmak için belirli bilgiler tutarlar.

Basitlik için ileride nesnelere de onların referanslarına da nesne denilecektir. Aralarında ayrım yapmak pedagojik olarak gerekmemektedir.

İçerik (Context)

İçerik isim nesne bağlantılarının kümesine verilen addır. Her içeriğin kendi isim dönüştürmesi vardır. Bir içerik isimlerden nesne döndüren en az bir işlem sağlar. Ayrıca isim-nesne bağlama ,bağlantıları koparma ve bağlı isimleri döndürme gibi işlemleri sağlayabilir. Bir içerik nesnesindeki bir isim aynı isim dönüştürmesine sahip başka bir içerik nesnesine bağlanabilir.

Mesela bir dizin olan /usr , Unix kütük sisteminde bir içeriğe sahiptir.Bir kütük içeriğine göreli başka bir kütük içeriği onun altiçeriğini (*subcontext*) oluşturur(Unix kullanıcıları buna altdizin de der). /usr/bin dizini başka bir dizin olan usr ye bağlıdır. Başka bir örnekse DNS etki alanları için geçerlidir. COM etki alanı bir içerğe sahiptir ve Başka bir DNS etki alanına göreli bir DNS etki alanına altiçerik denir. Sun.COM etki alanı COM etki alanının altiçeriğidir.

Son olarak c=TR gibi bir LDAP girişi (entry) bir içeriktir. Başka bir LDAP girişine göreli LDAP girişine altiçerik denir. o=sun,c=TR, girişinde o=sun girişi c=TR 'nin altiçeriğidir.

İsimlendirme sistemleri ve İsim Uzayları

Bir isimlendirme sistemi birbiri ile bağlantılı ve aynı isim dönüştürmesine uyan içeriklerden oluşur ve ortak bir işlem kümesi sunar. DNS isimlendirme sistemini gerçekleştiren bir sistem,yada LDAP isimlendirme sistemiyle iletişim kuran sistemler buna örnek gösterilebilir.

Bir isimlendirme sistemi kullanıcılarına isimlerle alakalı işlemleri sağlamak amacıyla bir isimlendirme servisi gerçekleştirir. Bir isimlendirme servisine onun ara yüzünden ulaşılır. Mesela DNS makine isleriyle IP adreslerini eşleştiren bir isimlendirme servisi sunar. LDAP ise LDAP isimlerini LDAP girişleriyle eşleştiren bir isimlendirme servisi sunar. Bir kütük sistemi kütük isimleriyle kütük ve dizinleri eşleştiren bir isimlendirme servisi sunar.

Bir isim uzayı isimlendirme sistemindeki isimlerin bir kümesidir. Mesela Unix kütük sistemi tüm kütük ve dizin adlarından oluşan bir isim uzayına sahiptir. DNS isim uzayı DNS etki alanları ile girişlerinin adlarını tutar. LDAP isim uzayı LDAP girişlerinin isimlerini tutar.

Dizinleme Kavramları

Çoğu isimlendirme servisleri bir dizinleme servisiyle genişletilmiştir. Bir dizinleme servisi yine isimleri nesnelerle ilişkilendirir aynı zamanda böyle nesnelerin özniteliklere (*attributes*) sahip olmasına izin verir. Bununla beraber sadece nesnelere isimlerle bakmakla kalmaz ayrıca nesnelerin özniteliklerini de alırsınız, yada özniteliklerine bağlı olarak nesneleri arayabilirsiniz.

Mesela bir telefon şirketinin dizinleme servisi kayıtlı kullanıcıların isimlerini adresleri ve telefonlarına ulaşmak için dizinler(mapping). Bir bilgisayarın dizinleme servisi buna çok benzer ama çok daha güçlüdür. Çevrimiçi kullanılabilir olduğundan ve farklı çeşit bilgi tutabildiğinden bilgisayardaki dizinleme servisi kullanıcılar programlar ve hatta bilgisayarın kendisi ve başka bilgisayarlar tarafından kullanılır.

Bir dizin nesnesi bilgisayar ortamındaki bir nesneyi gösterir. Ve mesela bir yazıcıyı bir kişiyi bir bilgisayarı yada bir ağı gösterebilir. Bir dizin nesnesi gösterdiği nesneyi tanımlayan öznitelikler içerir.

Öznitelikler

Bir dizin nesnesinin öznitelikleri olabilir. Mesela bir yazıcı nesnesi onun hızını çözünürlüğünü ve rengini tutan bir dizin nesnesinde tutulabilir. Kullanıcıları tutan bir dizin nesnesinde kullanıcının e-mail hesabı çeşitli telefon numaraları, gerçek adresi ve bilgisayar hesap bilgileri öznitelik olarak tutulabilir.

Bir özniteliğin bir öznitelik tanımlayıcısı (*attribute identifier*) ve bir öznitelik değerler kümesi bulunur. Bir öznitelik tanımlayıcısı o özniteliği tanımlayan öznitelik değerlerinden bağımsız bir simgedir(*token*). Mesela iki farklı bilgisayar hesabında da 'adres' özniteliği olsun. 'adres' o özniteliğin öznitelik tanımlayıcısıdır. Bir öznitelik değeri ise o özniteliğin içeriğidir. Mesela 'adres' özniteliğinin tanımlayıcısı 'adres' ve değeri 'Bilgisayar müh. Bölümü Beytepe Ankara' olabilir.

Dizinler ve dizinleme servisleri

Bir dizin birleştirilmiş dizin nesnelerinin bir bütünüdür. Bir dizinleme servisi ise bir dizindeki nesneleri yaratma ekleme silme veya özniteliklerini değiştirmek için işlemler sağlayan servistir. Bu servise kendi ara yüzünden ulaşılır.

Dizinleme servisleri için bir sürü örnek mevcuttur. Novell Dizinleme Servisi (NDS) Novell tarafından kütük ve yazıcı servisi gibi bir çok ağ servisi hakkında bilgi sağlar. Ağ Bilgilendirme Servisi (Network information service NIS) Solaris işletim sisteminde , alakalı makineler ağlar yazıcılar ve kullanıcılar gibi sistemle alakalı bilgileri depolamak amaçlı kullanılan dizinleme servisidir. SunONE dizinleme servisi ise LDAP Internet tabanlı standardına uygun genel amaçlı bir dizinleme servisidir.

Aramalar ve arama filtreleri

Bir dizin nesnesini dizinleme servisine ismini sağlayarak arayabilirsiniz. Yada LDAP tabanlılar gibi bir çok dizin arama fikrini destekler.Ararken bir isim yerine aradığınız nesnenin sahip olması gereken öznitelikler üzerinden mantıksal ifadelerle sorgu yapabilirsiniz. Sorgunun kendisine arama filtresi (*search filter*) denir. Bu tip aramalara bazen tersine arama(reverse lookup) yada içerik tabanlı arama(*content based searching*) denir. Dizinleme servisi filtreye göre arama yapar ve sağlayan nesneleri döndürür.

Mesela dizinleme servisine yaşı 20 ila 25 arasındaki tüm kullanıcıları bulmasını veya IP adresi 192.113.50 ile başlayan tüm makineleri bulmasını isteyebilirsiniz.

İsimlendirme ve Dizinleme servislerini birleştirme

Dizinler nesneleri genellikle belli bir hiyerarşide tutarlar. Mesela LDAP tüm dizin nesnelerini dizin bilgi ağacı(Directory Information Tree DIT) denen bir ağaç veri yapısında tutar. Bu ağaçta (DIT) mesela organizasyon nesneleri grup nesnelerini grup nesneleri ise personel nesnelerini tutabilir. Dizin nesneleri böyle düzenlendiğinde öznitelikleri saklamanın yanı sıra isimlendirme içeriklerin rolünü de oynarlar.

Dizin-etkin Java uygulamaları

Dizinleme servisleri ağ üzerinde bilgi işlemin(*network computing*) hayati bir unsurudur. Bir dizinleme servisini kullanarak uygulamaları ve uygulama yönetmeyi , paylaşılan bilgiyi merkezileştirerek basitleştirebilirsiniz. Java programlama dilinin kullanılmasıyla ağ ortamında yazılan pratik uygulamaların sayısı artmış dizinleme servislerine erişmek bir zaruret haline gelmiştir.

Dizinlemenin geleneksel kullanımı

Bir uygulamanın dizin-etkin uygulama olabilmesi için isimlendirme ve dizinleme servislerini kullanması gerekir. Dizin etkin Java uygulamaları ve appletler ağda çalışan herhangi başka bir uygulama gibi dizinleri geleneksel yöntemlerle , özniteliklerini yüklemek ve erişmek için, kullanabilir. Mesela bir Java mail istemci programı dizini mail alıcıların adreslerini saklamak için bir adres defteri olarak kullanabilir . Bir Java mail transfer ajanı ise aynı dizini mail yönlendirme bilgisini tutmak için , bir Java ajanda programı ise kullanıcı tercihlerini almak için kullanabilir.

Uygulamalar dizin sağlamış olduğu ortak altyapıyı paylaşabilirler. Bu paylaşım sistemde ve ağ üzerindeki uygulamaları daha uyumlu ve yönetilebilir kılar. Mesela yazıcı yapılandırması veya mail yönlendirme bilgileri bir dizinde tutularak tüm yazıcı veya maille alakalı uygulamalar tarafından türetilip dağıtılabilir.

Bir nesne deposu olarak dizin

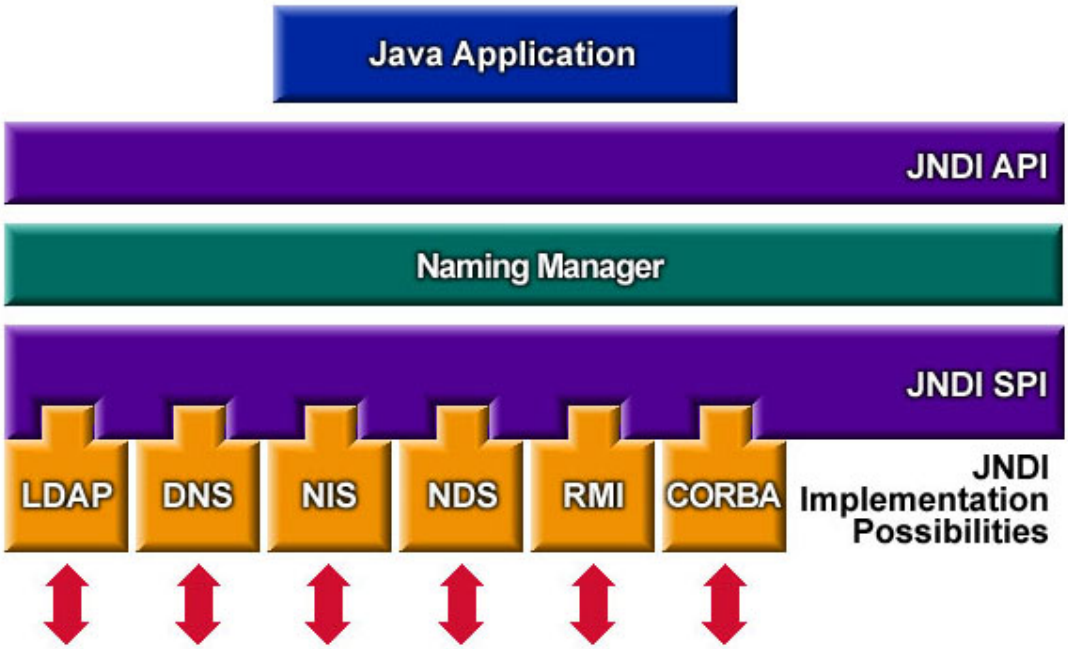
Dizinleri geleneksel olarak kullanmanın yanı sıra Java uygulamaları dizinleri Java nesnelerinin bir nesne havuzu olarak ta kullanabilir.mesela bir yazıcı istemci Java uygulaması dizinden bir yazıcı nesnesi arayabilir ve o nesneye çıktı için veri katarı gönderebilir.

JNDI ‘ a genel bakış

Java isimlendirme ve dizinleme servisi (Java Naming and Directory Interface) Java dilinde yazılmış uygulamalara isimlendirme ve dizinleme işlevselliği katan bir uygulama programlama ara yüzüdür(Application Programming Interface). Herhangi bir isimlendirme ve dizinleme servisinden bağımsız olarak tasarlanmıştır. Böylelikle birçok dizine –yeni , görünen ya da çoktan kurulmuş- ortak bir yolla erişim sağlanır.

Mimari

JNDI mimarisi bir API ve bir servis sağlayıcı ara yüzünden (service provider interface SPI) oluşur. Java uygulamaları JNDI API ‘sini birçok isimlendirme ve dizinleme servisine aynı yolla ulaşmak için kullanırlar. SPI, birçok isimlendirme ve dizinleme servisini etkinleştirip onların şeffaf olarak takılıp çıkarılmasını ve JNDI API’ sini kullanan Java uygulamalarının bu servislere erişmesini sağlar.



Paketleme

JNDI ,Java 2 SDK v1.3 ve sonraki sürümlerinde dahil edilmiştir. Java Standart Extension olarak JDK 1.1 ve Java 2 SDK, v1.2 ‘nin kullanımı için de ulaşılabilir. v1.1 ve v1.2 platformlarını isimlendirme ve dizinleme hizmeti vermek için genişletir.

JNDI ‘ ı kullanmak için JNDI sınıflarına ve bir de servis sağlayıcısına (service provider) ihtiyacınız var. Java 2 SDK v1.3 aşağıdaki isimlendirme/dizinleme servisleri için 3 tane servis sağlayıcısı içerir.

- Tüy sıklet dizin erişim protokolü (Lightweght Directory Access Protocol LDAP)
- Genel nesne istemci broker mimarisi (CORBA) genel nesne servisleri (COS) isimlendirme servisi
- Java uzaktan metot çağırma (RMI) kaydı (*registry*)

Diğer servis sağlayıcılar JNDI 'in (http://java.sun.com/products/jndi/serviceproviders.html) web sitesinden veya diğer sağlayıcılarım sitelerinden indirilebilir. JNDI 'ı JDK 1.1 veya Java 2 SDK v1.2 ile kullanmak için ilk önce JNDI nesnelerini http://java.sun.com/products/jndi adresinden indirebilirsiniz.

JNDI beş pakete ayrılmıştır. Bunlar;

- javax.naming
- javax.naming.directory
- javax.naming.event
- javax.naming.ldap
- javax.naming.spi

İsimlendirme paketi

javax.naming[api] paketi isimlendirme servislerine erişmek için ara yüzler ve sınıflar içerir.

İçerik (Context)

Javax.naming paketi nesneleri aramak bağlamak/koparmak yeniden adlandırmak ve alt içerikler oluşturup yok etmek için Context[api] ara yüzünü tanımlar.

En çok kullanılan işlem lookup()[api] dir. lookup() 'a aramak istediğiniz nesnenin adını sağlarsınız o da size o isme bağlanmış nesneyi döndürür. Mesela aşağıdaki kod kesimi bir yazıcı arar ve dokümanı yazılmak üzere yazıcı nesnesine iletir.

```
Printer yazici = (Printer)ctx.lookup("BolumYazicisi");
yazici.print(rapor);
```

İsimler

İçerik ara yüzünde her metodun iki tane farklı aşırı yüklenen gerçekleştirimi vardır. Bunlardan biri Name[api] 'yi argüman olarak alan diğeri ise Java.lang.String isimlerini argüman alanlardır. Name şablon bir isim nesnesini gösteren bir ara yüzdür. İçerik ara yüzündeki metotlar için bileşik isimleri argüman olarak vermek için CompositeName[api] 'in bir nesnesi kullanılır. Böylece farklı isim uzaylarına yayılan isimler kullanabilirsiniz. Bir Name argümanını gösteren bir diğer gösterim ise bileşen isim (compound Name)dir . ve Name ismin bileşenlerden oluştuğunu varsayar. Name 'i kabul eden aşırı yüklenen metotlar isimleri işleyen onları birleştiren yada bileşenlerine ayıran uygulamalar için daha kullanışlıdır.

Bir java.lang.String nesnesi de bir bileşik ismi gösterir. java.lang.String 'i kabul eden metotlar daha basit uygulamalar için daha uygun görünür. Bu uygulamalar zaten sadece bir ismi okur ve buna karşılık gelen nesneyi arar.

Bağlantılar (Bindings)

listBindings()[api] isim – nesne bağlantılarının bir numaralandırmasını (enumeration) döndürür. Her bağlantı Binding[api] 'nin bir örneğidir(instance). Bir bağlantı nesnesinde bağlanan nesnenin ismi , bağlanan nesnenin sınıf ismi ve nesnenin kendisi tutulur.

list()[api] de listBindings() in benzeridir fakat list()NameClassPair[api] numaralandırmasını döndürür. NameClassPair nesnenin ismini ve nesnenin sınıfının ismini tutar. Nesnenin tamamını değil de sadece nesneyle alakalı bilgileri arayan küçük çaplı uygulamalar için daha uygundur. Her ne kadar ListBindings() aynı tüm bilgileri içeriyor olsa da potansiyel olarak daha pahlı bir işlemdir.

Referanslar

Nesneler isimlendirme ve dizinleme servislerinde farklı şekillerde tutulabilirler. Java nesnelerini saklamayı destekleyen bir servis o nesneyi seri hale getirip te saklayabilir. Buna rağmen bazı servisler Java nesnelerini saklamayı desteklemez.
Bundan başka dizindeki bazı nesneler sadece Java nesneleri tarafından erişilen nesneler olmayabilir. Böyle bir durumda seri hale getirilmiş Java nesneleri en uygun saklama biçimi olmayabilir. Böyle bir durumda nesnelerin referanslarını(*reference*) tutmak daha uygun bir yaklaşım olabilir.

JNDI , referansları göstermek için Reference[api] sınıfı tanımlar. Bir referans nasıl nesnenin kopyasının oluşturulabileceğine dair bilgiler içerir. JNDI gerektiği zaman referanslardan nesnenin kendisinin bir kopyasını oluşturarak istemcilere bu nesneyi döndürür. Böylelikle istemciler dizinde Java objelerinin tutulduğu illüzyonuna kapılabilirler.

Başlangıç İçeriği

JNDI da tüm isimlendirme ve dizinleme işlemleri belirli bir içeriğe göreli olarak yapılır. Yani kesin kök olan bir içerik yoktur. Bu yüzden JNDI bir ilk başlangıç içeriği InitialContext[api] tanımlar. InitialContext servisler için bir başlangıç noktası mahiyetindedir. Bir başlangıç içeriğiniz varsa bunu başka nesneler veya içerikler aramak için kullanabilirsiniz.

Aykırı Durumlar

JNDI isimlendirme ve dinleme hizmetlerini sunarken oluşabilecek aykırı durumlar için belirli bir sınıf hiyerarşisi tanımlar. Bu hiyerarşinin kökünde NamingException[api] sınıfı bulunur. Özel bir aykırı durum yaklaşımı kullanmak isteyen programlar bu sınıfın altsınıfını yakalayabilirler(catch). Aksi halde NamingException yakalamaları gerekir.

Dizinleme Paketi

javax.naming.directory[api] paketi isimlendirme servislerinin yanı sıra dizinleme servislerine de erişirken daha çok işlevsellik sağlamak amacıyla javax.naming[api] paketinden genişletilmiştir. Bu paket uygulamalara depolanan nesnelerle alakalı öznitelik

bilgilerine erişilebilmesini ve belirli öznitelik değerlerine göre arama yapılabilmesini sağlar.

Dizin İçeriği

DirContext[api] ara yüzü bir dizinle içeriği(*directory context*) sunar. Dizin nesnelerinin özniteliklerini sınama ve güncellemeye olanak veren işlemleri tanımlar.

getAttributes()[api] (İsmini sağladığınız) bir dizinle alakalı özniteliklere erişmek için kullanılır. Öznitelikler modifyAttributes()[api] yi kullanarak güncellenir.

DirContext ayrıca Context[api] ara yüzünden genişlediği için bir isimlendirme içeriği gibi davranabilir. Bunun manası herhangi bir dizinleme servisinin bir isimlendirme servisi ile aynı hizmeti verebilir olmasıdır. Mesela bir kişi için oluşturulan bir dizin nesnesi hem özniteliklere sahip olup hem de isimlendirme nesneleri için bir içerik –O kişinin yazıcısı dosya sistemi gibi kişinin dizin nesnesine göreli bilgiler-sağlayabilir.

Aramalar

DirContext dizinlerde içerik tabanlı aramayı gerçekleştiren metotlar içerir. Kullanımın en basit ve genel formu uygulamanın bir grup belirli değerlere sahip öznitelik tanımlaması ve bu kümeyi uyumluluk kontrolü için search()[api] metoduna sunması şeklindedir. search()[api] 'nin diğer aşırı yüklenmiş metotları daha karmaşık arama filtrelerini destekler.

Olay Paketi

javax.naming.event[api] paketi isimlendirme ve dizinleme servislerinde olay bildirmeyi sağlamak için sınıflar ve ara yüzler içerir.

Olaylar

Bir NamingEvent[api] bir isimlendirme/dizinleme servisi tarafından üretilen bir olaydır. Olayın *type* adı verilen olaya dair bir tür değişkeni vardır.Mesela olay türleri isim uzayını etkileyecek şekilde kategorize edilirse; "nesne yüklendi" yada "nesne değiştirildi" gibi değerler alırlar. Bir NamingEvent nesnesi nesne üzerinde yapılan değişiklikler hakkında önceki ve sonraki durumu gibi daha çok bilgi içerir.

Dinleyiciler (Listeners)

NamingListener[api] nesnesi Namingevent nesnelerini dinleyen nesnelerdir. Olay türlerinin her kategorisinin bir NamingListener sınıfı dinleyicisi bulunur. Mesela bir NamespaceChangeListener[api] isim uzayındaki değişiklerle ilgilenen bir dinleyici sunar. Ve bir ObjectChangeListener[api] ise nesne değişiklik olayları ile ilgilenir.

Olay bilgilendirmelerini alabilmek için bir dinleyici bir EventContext[api] yada bir EventDirContext[api] ile kayıt ettirilir(*register*). Bir kere kaydedildiği zaman bir dinleyici isimlendirme/dizinleme servisindeki ilişkili değişiklikler meydana geldiği zaman olay bilgilendirmeleri (*event notifications*) alır.

LDAP Paketi

javax.naming.ldap[api] paketi , halen daha üreysel olan javax.naming.directory[api] nin tam olarak kapsayamadığı LDAP v3 'e has belirli özellikler için sınıflar ve ara yüzler içerir. Aslında LDAP 'ı kullanan çoğu JNDI uygulaması javax.naming.directory paketini yeterli bulabilir ve javax.naming.ldap paketini kullanmayabilir. Bu paket öncelikle uzatılmış işlemler kontroller yada talep edilmemiş bildirimler kullanması gereken uygulamalar için anlamlıdır.

Uzatılmış İşlemler (Extended Operations)

Arama değiştirme gibi iyi-tanımlanmış işlemlerin yanı sıra LDAP v3 (RFC 2251) LDAP istemci ve sunucuları arasıda henüz-tanımlı (yet-to-be defined) işlemleri iletmek için bir yol belirtir. Bu işlemlere uzatılmış işlemler denir. Bir uzatılmış işlem bir standart organizasyonu olan Internet mühendislik çalışma kolu (Internet Engineering Task Force (IETF)) tarafından yada bir sağlayıcı tarafından tanımlanmalıdır. Bu paket Start TLS uzantısı(Start TLS extension.) için sınıflar tanımlar.

Kontroller (Controls)

LDAP v3 herhangi bir istem veya cevabın henüz tanımlanmış kontroller adındaki değiştiriciler tarafından artırılmasına izin verir. bir istemle beraber gönderilen kontrol'e istem kontrolü (*request control*) ve bir cevapla beraber gönderilen kontrole de cevap kontrolü (*response control*) denir. Bir kontrolün IETF gibi standart belirleyen bir organizasyon yada sağlayıcılar tarafından tanımlanması gerekir. İstem kontrolleri ve cevap kontrolleri birer çift gibi düşünülmemelidir. Gönderilen her istem kontrolünün bir cevap kontrolü olmasına gerek yoktur.

Talep Edilmemiş bildirimler (Unsolicited Notifications)

İstemci sunucu mimarisindeki normal istem/cevap mekanizmasına ek olarak LDAP v3 sunucu tarafından asenkron bir biçimde istemcilere her hangi bir istemin cevabı olmayan mesajlar göndermeyi de belirler. Bu tür mesajlara talep edilmemiş bildirimler denir.

LDAP içeriği

LdapContext[api] ara yüzü, istem kontrolleri gönderme veya cevap kontrolü alma gibi uzatılmış işlemleri yapabilmek için bir içerik sunar.

Servis sağlayıcı paketi

javax.naming.spi[api] paketi farklı isimlendirme ve dizinleme servisi üreticilerinin kendi gerçekleştirimlerini üretip JNDI kullanan uygulamalar için erişilebilir olmasını sağlamasına vesile olur.

Tak-Çalıştır Mimarisi (Plug-in Architecture)

javax.naming.spi paketi farklı gerçekleştirimlerin dinamik olarak eklenip kaldırılmasına olanak verir. bu gerçekleştirimler başlangıç içeriği ve ondan erişilebilen içerikleri de kapsar.

Java Nesne Desteği (Java object support)

javax.naming.spi paketi Context.lookup() [api] nin ve ilgili metotların uyarlamalarını Java programcısı için doğal ve sezgisel olan nesneler döndürmek için destekler. Mesela bir dizinden bir yazıcı arıyorsunuz, o halde büyük ihtimalle bir yazıcı nesnesinin dönmesini beklersiniz. Bu destek Object Factories (Factory Tasarım örüntsünün bir gerçekleştirimi) tarafından sağlanır.

Bu paket ayrıca tersini de yapmayı destekler. Context.bind() [api] ve ilgili metotların uyarlamaları Java nesnelerini kabul edip isimlendirme/dizinleme servisinin kabul edebileceği bir biçimde depolanmasına izin verir. Bu destek State Factories tarafından sağlanır.

Çoklu İsimlendirme Sistemleri (Federasyonlar)

JNDI işlemleri uygulamalara ,uygulamaların birden fazla isim uzayına yayılabilen isimler sağlamasını destekler. Bir işlemi tamalama esnasında bir servis sağlayıcı başka bir servis sağlayıcı ile etkileşime geçme ihtiyacı hissedebilir. Mesela bir işlemi bir diğer servis sağlayıcıda devam etmesi için ona geçirebilir. Bu paket farklı servis sağlayıcıların bir JNDI işlemini bitirebilmek için beraber çalışmaları için destek sağlar.

GEREKLİ YAZILIM

Aşağıdakiler JNDI kullanmanız için size gerekebilecek yazılımlardır.

- Java platform yazılımı
- JNDI yazılımı
- Servis sağlayıcı yazılımı
- İsimlendirme ve dizinleme sunucu yazılımı

Java Platform Yazılımı

JNDI kullanan uygulamalar veya apletler yazmak için, v1.1.2 yada daha üst sürüm bir Java 2 Software Development Kit (SDK) yazılımına ihtiyacınız olacaktır.

JNDI yazılımı

Servis sağlayıcı yazılımlar istemci/sunucu paradigmasına dayalı sistemlerde daha çok istemci kısmına daha yakındırlar. JNDI mimarisini bir isimlendirme dizinleme sunucuna erişmek için o sunucunun servis sağlayıcısını bir istemci gibi kullanırız. Mesela LDAP dizinleme sunucusuna JNDI ile erişmek istiyorsak LDAP servis sağlayıcısını kullanmamız gerekir. Şu da bir gerçektir ki ; isimlendirme/ve dizinleme sunucuları sadece JNDI tabanlı yazılımlara hizmet vermez. JNDI gerekli sunucudan bağımsız hizmet almak için geliştirilmiş bir ara yüzdür.

İsimlendirme ve Dizinleme Sunucu Yazılımı

Bu dokümanda bir isimlendirme yazılımı olarak kendi Kütük sisteminizi kullanabilirsiniz. Ama mesela LDAP kullanmak istiyorsanız OpenLDAP gibi bedava sunucu yazılımlarından faydalanabilirsiniz. Daha sonraki kesinlerde LDAP sunucusunun bu dokümandaki örneklerle nasıl çalıştığı açıklanacaktır.

openLDAP yazılımını şu sunuculardan da alabilirsiniz :

- ldap://ldap.Bigfoot.com
- ldap://ldap.four11.com
- ldap://ldap.InfoSpace.com

ÖRNEKLER

Bu derste iki örneğimiz var.İlk örnek bir isimlendirme sisteminde bir ismi nasıl arayabileceğimizi (lookup()) anlatır. İkincisi ise bir dizinleme servisinde nasıl bir özniteliği okuyacağımızı anlatır. İkinci örnekte kullanılan dizinleme servisi LDAP 'tır. Örnek kütükleri bu kütüğün içinde bulunduğu dizinde (Sanırım) bulabilirsiniz.

İsimlendirme Örneği

Bu örnekte komut satırından ismi geçirilen bir nesnenin nasıl isimlendirme sisteminde bakılabileceğini (Lookup()) göreceğiz. Program kütük sistemi için bir servis sağlayıcı kullanır. Bu yüzden programa geçireceğiniz isim bir kütük adı olmalıdır.Bu noktada sevis sağlayıcı hakkında detay bilmenize gerek yoktur.

JNDI sınıflarını dahil etmek

Öncelikle Herhangi bir metin düzenleyici ile bir Java dosyası oluşturun ve ismini Lookup.java koyun. Sonra ya javax.naming[api] paketinin tamamını yada belirli sınıfları ve ara yüzleri kodunuza ekleyebilirsiniz. Aşağıdaki kod javax.naming paketindeki örneğimiz için
gerekli sınıfları dahil eder.

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
```

Bir başlangıç İçeriği oluşturmak

Programın main() metodunda bir başlangıç içeriği oluştur. Bir Kütük sistemi servis sağlayıcısı kullandığını InitialContext[api] yapıcısına (constructor) ortam değişkenleri (*environment properties)* parametrelerini kurarak göster. Aşağıdaki gibi:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.fscontext.RefFSContextFactory");

Context ctx = new InitialContext(env);
```

Bir Nesneye bakmak

Sonrasında Context.lookup()[api] metoduyla nesneyi ara. Aşağıdaki kod bir ad ismine bağlanmış bir nesne nesnesini arar.

```
Object nesne = ctx.lookup(ad);
```

NamingException Yakalamak

Bir başlangıç içeriği yaratmak ve lookup metodunu çağırmak bir NamingException[api] aykırı durumunu atabilir. Bu yüzden bu kod kesimini bir dene/yakala (try/catch) bloğunun içinde yazmamız gerekir. Mesela;

```
try {
    // bir başlangıç nesnesi oluştur
    Context ctx = new InitialContext(env);

    // Bir nesneye bak
    Object nesne  = ctx.lookup(ad);

    // Ekrana yazdır
    System.out.println(ad + " 'ın bağlı olduğu nesne : " + obj);

} catch (NamingException e) {
    System.err.println("Hata :" + ad + ": ismine bakarken :" + e);
}
```

Programı derlemek

Artık kaynak kütüğü Java derleyicisiyle derleyebiliriz. Fakat önce JNDI sınıflarımızın olduğundan emin olmalıyız. Java 2 SDK v1.3 ya da daha yeni bir versiyonu kullanıyorsanız zaten JNDI sınıflarınız vardır Fakat yoksa CLASSPATH değişkenini JNDI 'ın web sitesinden indirdiğiniz jndi.jar kütüğünün yolunu da içerecek şekilde genişletebilirsiniz.

Programı Çalıştırmak

Java SDK v1.3 kullanıyor olsanız bile bazı servis sağlayıcı sınıflarınız ilk başta yoktur. Kütük yönetimi servis sağlayıcısının sınıflarını (fscontext.jar ve providerutil.jar) CLASSPATH değişkeninde göstererek onları sonradan da ekleyebilirsiniz. Son olarak Lookup.class 'ınızın da bulunduğu dizini CLASSPATH 'e ekleyerek programı çalıştırabilirsiniz.

Bunun dışında programı komut satırından bulmak istediğiniz dosyanın yolunu arguman olarak vererek çalıştırabilirsiniz.

```
# java Lookup /tmp
/tmp is bound to: com.sun.jndi.fscontext.RefFSContext@1dae083f
```

gibi.

Dizinleme Örneği

Bu ornekbir dizin nesnesinden nasıl öznitelik alınacağını gösterir. Bir LDAP servis sağlayıcısı kullanır. Internet 'ten openldap gibi bdava LDAP servis sağlayıcıları edinebilirsiniz.

JNDI dizinleme sınıflarını dahil etmek

Getattr.java adında bir kütük açın ve javax.naming ve javax.naming.directory paketlerini ya da kullanacağınız belirli sınıf ve ara yüzleri dahil edin. Bunlar aşağıdaki gibidir;

```
import javax.naming.Context;
import javax.naming.directory.InitialDirContext;
import javax.naming.directory.DirContext;
import javax.naming.directory.Attributes;
import javax.naming.NamingException;
```

Bir Başlangıç İçeriği Oluşturma

main() metodunda isimlendirme örneğinde olduğu gibi bir ilk içerik oluşturabilirsiniz. Tek fark bu sefer InitialDirContext[api] sınıfını kullanacak olmanız;

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
```

```
    "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:389/o=JNDITutorial");

DirContext ctx = new InitialDirContext(env);
```

InitialDirContext constructor[api] sınıfına LDAP servis sağlayıcısını kullanacağınızı bildirdikten sonra artık sorgulara geçebilirsiniz. Burada LDAP sunucunuzun kullandığı geçit(port) farklı olabilir veya başka bir makinede olabilir. Buna göre parametreleri değiştirmeniz gerekebilir.

Bir Dizin Nesnesinin Özniteliklerini Alma

Artık getAttributes()[api] metodunu dizinin özniteliklerini almak için kullanabilirsiniz. Mesela ;

```
Attributes attrs = ctx.getAttributes("cn = Ted Geisel, ou=People");
```

Burada "cn=Ted Geisel, ou=People" nesnenin bağlı olduğu isimdir.

İstenen Özniteliğe Ulaşma

Attributes.get()[api] metodu ile istediğiniz özniteliği alabilirsiniz.Mesela;

```
attrs.get("sn").get();
```

NamingException Yakalamak

Tüm bu metotlar NamingException aykırı durumu fırlatabilirler. Bu yüzden bir dene/yakala (try/catch) bloğu içinde yazılması gerekir. Mesela;

```
try {

    // Bir başlangıç içeriği oluştur
    DirContext ctx = new InitialDirContext(env);

    // Nesnenin tüm özniteliklerini iste
    Attributes attrs = ctx.getAttributes("cn=Ted Geisel, ou=People");

    // Soyad özniteliği olan ("sn") öznieliğini bu ve ayzdır
    System.out.println("sn: " + attrs.get("sn").get());

} catch (NamingException e) {
    System.err.println("Problem getting attribute:" + e);
}
```

Programı derleme ve çalıştırma

İsimlendirme örneğinde olduğu gibi JNDI sınıflarını tanıtmanız gerekir. Ayrıca servis sağlayıcı sınıflarına da (ldap.jar ve providerutil.jar). erişeceksiniz. Eğer Java 2 SDK v1.3 kullanıyorsanız bunlar zaten eklenmiştir. Ornek çıktı aşağıdaki gibidir;

```
# java Getattr
sn: Geisel
```

Burada dikkat edilmesi gereken

```
env.put(Context.PROVIDER_URL, "ldap://localhost:389/o=JNDITutorial");
```

satırıdır. LDAP sunucusunu doğru tanıttığınızdan ve o=JNDITutorial isim uzayının tanıtılmış olduğundan emin olun.

Paketler ve ClassPath

JNDI 'ı uygulamalarınızda kullanmak istiyorsanız ilk önce onun derlenmesini ve çalışma ortamını hazırlamalısınız.

JNDI Sınıflarını Dahil etmek

Aşağıdaki paketler JNDI paketleridir.

- `javax.naming` api
- `javax.naming.directory` api
- `javax.naming.event` api
- `javax.naming.ldap` api
- `javax.naming.spi` api

JNDI ile ilgili tüm uygulamalarınızda aşağıdaki paketleri dahil etmeniz yeterlidir.

```
import javax.naming.*;
import javax.naming.directory.*;
```

yada sadece kullandığınız belirli sınıf ve ara yüzleri de dahil edebilirsiniz.

Derleme Ortamı

Java 2 SDK v1.3 ünüz varsa zaten JNDI sınıflarınız var demektir. Am Java sdk nın daha eski sürümlerini kullanıyorsanız JNDI Web sitesinden gerekli JNDI sınıflarını indirebilirsiniz.

Java 2 SDK, v1.2 kullanıyorsanız JNDI sınıflarını java uzantıları olarak *JAVA_HOME*/jre/lib/ext dizinine kopyalayabilirsiniz. JAVA_HOME burada SDK nın kurulu olduğu dizindir. Yada JDK 1.1 kullanıyorsanız jndi.jar kütüğünü herhangi kalıcı bir dizine kopyalayıp CLASSPATH ortam değişkenini o dizini de gösterecek şekilde genişletebilirsiniz.

Çalışma Ortamı

JNDI kullanan uygulamaları çalıştırmak için JNDI sınıflarına ve uygulamanızın kullandığı servis sağlayıcının sınıflarına erişebilmeniz gerekir. Java 2 Runtime Environment (JRE) v1.3 zaten JNDI sınıfları ve servis sağlayıcılar olan LDAP ,COS adlandırma RMI kayıtçısı sınıflarını içerir. Eğer başka bir servis sağlayıcı kullanmak istiyorsanız onun arşiv dosyalarını JAVA_HOME/jre/lib/ext dizinine kopyalamanız gerekir. JAVA_HOME burada JRE nin kurulu olduğu dizindir.

Mesela bu dokümanda LDAP ve kütük sistemi servis sağlayıcılarını kullanacaksınız. Bunu sağlamak için fscontext.jar 'ı ve providerutil.jar 'ı CLASSPATH değişkeninde göstermeniz gerekir.

Naming Aykırı Durumları

JNDI paketindeki metotların neredeyse tümü yapması gereken işi yapamadığını göstermek için NamingException aykırı durumu fırlatır. Genelde bu tip metotlarınetrafında dene/yakala (try/catch) blokları bulursunuz.

```
try {
    Context ctx = new InitialContext();
    Object obj = ctx.lookup("somename");
} catch (NamingException e) {
    // Handle the error
    System.err.println(e);
}
```

Aykırı Durum Sınıf Hiyerarşisi

NamingException aykırı durumunun bir alt sınıfını özellikle ele almak istiyorsanız onu ayrıca yakalarsınız. Mesela:

```
try {
    Context ctx = new InitialContext();
    Object obj = ctx.lookup("somename");
} catch (AuthenticationException e) {
    // attempt to reacquire the authentication information
    ...
} catch (NamingException e) {
    // Handle the error
    System.err.println(e);
}
```

Numaralandırmalar (Enumerations)

Context.list() api , DirContext.search() api gibi metotlar NamingEnumeration api sınıfı döndürür. Eğer bir aykırı durum oluştuğunda hala dönen değerleri almak istiyorsanız NamingEnumeration.hasMore() api hatayı göstermek için bir NamingException yada alt sınıflarından birini fırlatır.

javax.naming[api] Paketindeki Aykırı durumlar

NamingException[api]
- CannotProceedException[api]
- CommunicationException[api]
- ConfigurationException[api]
- ContextNotEmptyException[api]
- InsufficientResourcesException[api]
- InterruptedNamingException[api]
- InvalidNameException[api]
- LimitExceededException[api]
  - SizeLimitExceededException[api]
  - TimeLimitExceededException[api]

LinkException[api]
- LinkLoopException[api]
- MalformedLinkException[api]

NameAlreadyBoundException[api]

NameNotFoundException[api]

NamingSecurityException[api]
- AuthenticationException[api]
- AuthenticationNotSupportedException[api]
- NoPermissionException[api]

NoInitialContextException[api]

NotContextException[api]

OperationNotSupportedException[api]

PartialResultException[api]

ReferralException[api]

ServiceUnavailableException[api]

javax.naming.directory[api] paketindeki aykırı durumlar

NamingException[api]
- AttributeInUseException[api]
- AttributeModificationException[api]
- InvalidAttributeIdentifierException[api]
- InvalidAttributesException[api]
- InvalidAttributeValueException[api]

InvalidSearchControlsException[api]
InvalidSearchFilterException[api]
NoSuchAttributeException[api]
SchemaViolationException[api]

javax.naming.ldap[api] **paketindekiaykırı durumlar**

NamingException[api]
ReferralException[api]
LdapReferralException[api]

Başlangıç İçeriği (Inıtıal Context)

Bir İsimlendirme/Dizinleme servisinde herhangi bir işlem yapmadan önce bir başlangıç içeriği elde etmeniz gerekir. Tüm metotlar bu içeriğe göreli çalışırlar. Bir başlangıç içeriği almak için

1. erişmek istediğiniz servis sağlayıcısının ve ulaşmak istediğiniz servisin belirlenmesi
2. başlangıç içeriğinin ihtiyaç duyabileceği herhangi bir yapılanmayı(configuration) belirtilmesi
3. InitialContext[api] yapılandırıcısının çalıştırılması

Servis sağlayıcısını seçmek

Servis sağlayıcısını ortam değişkenlerinin (environment properties) bir kümesini (bir Hashtable) oluşturup servis sağlayıcısının ismini de o kümeye katarak belirleyebilirsiniz.

Mesela Sun Microsystems 'in LDAP servis sağlayıcısını seçmek için kodunuz aşağıdakine benzer biçimde olmalıdır;

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.ldap.LdapCtxFactory");
```

Eğer bir kütük sistemi servis sağlayıcısını seçmek istiyorsanız kod bu sefer aşağıdakine benzeyecektir;

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.fscontext.RefFSContextFactory");
```

ayrıca sistem özelliklerini (system properties) kullanarak ta serivs sağlayıcısını seçebilirsiniz. Bunun için jndiprovider.properties adlı özel bir dosyayı sisteminize tanıtmanız gerekir.

Başlangıç İçeriğine Gereken Bilgiyi Sağlamak

Farklı dizinleme servisleri istemciyle diyaloga geçmeden önce farklı bilgilere ihtiyaç duyabilir. Mesela LDAP servis sağlayıcısı LDAP sunucusunun yerini ve hatta kullanıcı adı ,şifre bilgilerini isteyebilir. Böyle bir durumda kodun devamı şuna benzer;

```
env.put(Context.PROVIDER_URL, "ldap://ldap.wiz.com:389");
env.put(Context.SECURITY_PRINCIPAL, "joeuser");
env.put(Context.SECURITY_CREDENTIALS, "joepassword");
```

Mesela bu dokümanda örneklenen kütük sistemi servis sağlayıcısı PROVIDER_URL bilgisini parametre olarak ister. Bunun Daha önce çalıştırdığımız Setup.java programına verdiğimiz parametre ile aynı olması gerekir.

```
env.put(Context.PROVIDER_URL, "file:/tmp/tutorial/");
```

yada

```
env.put(Context.PROVIDER_URL, "file:c:\\birdizin\\");
```

LDAP ı kullanan örnekler ise yerel makine de bir sunucunun bulunduğunu ve 389 uncu geçitten (port) iletişim kurulduğunu ve kökün ayırt edici isminin o=JNDITutorial olduğunu ayrıca hiç kimlik denetimi yapılmadığını farzeder. Aşağıdaki kodu ortam değişkenlerini günlemek için kullanır;

```
env.put(Context.PROVIDER_URL, "ldap://localhost:389/o=JNDITutorial");
```

Eğer başka bir şekilde çalışıyorsanız çalıştığınız ortama uygun değişkenler kullanmak zorundasınız.

Başlangıç İçeriğini Oluşturmak

Artık başlangıç içeriğini oluşturmaya hazırız. bunu yapmak için InitialContext constructor[api] metoduna ortam değişkenlerini geçiririz;

```
Context ctx = new InitialContext(env);
```

Artık bir Context[api] nesnesine referansımız var. Bundan sonra isimlendirme servisine erişmeye başlayabiliriz.

Dizinleme işlemlerini yapmak için ise InitialDirContext[api] ara yüzünü kullanırız. Bunu yapmak için onun yapılandırıcılarından (constructors[api]) birini kullanırız.

```
DirContext ctx = new InitialDirContext(env);
```

Bu ifade bize kullanmamız için bir DirContext[api] nesnesine referans döndürür.

İsimler

Context[api] ve DirContext[api] ara yüzleri her metodun aşırı yüklenmiş(overloaded) halini de içerir. Bunlardan birincisi java.lang.String isim kabul eden bir diğeri ise Name[api] sınıfı isim kabul edendir. Her bir aşırı yüklenen metot birbirinin dengidir. Fakat isimlerin değişebildiği göze alındığında nesnelerin isimlerini değiştirilebilir tutmanındaha mantıklı olduğu uygulamalarda Name[api] sınıfı daha etkili kullanılabilir.

İsimlendirme işlemleri

JNDI 'ı okuma ve isim uzayını günleme gibi isimlendirme işlemleri için de kullanabilirsiniz Anlatılan işlemler :

- .bir nesneyi aramak
- bir içeriğin içindekilere bakmak
- bir bağlantıyı ekleme silme günleme
- bir nesneyi yeniden adlandırma
- alt içerikler yaratma ve silme

Bu örnekler bir kütük sistemi servis sağlayıcısı kullandığından bu dokümanla aynı dizindeki Setup.java kütüğünü çalıştırıp gerekli ön hazırlığı yapabilirsiniz.

```
// Ortam değişkenlerini kütük sistemi hizmeti verecek şekilde ayarlamak
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.fscontext.RefFSContextFactory");
env.put(Context.PROVIDER_URL, "file:/tmp/tutorial");
Context ctx = new InitialContext(env);
```

Bir nesneye bakmak

Context.lookup()[api] bir nesneyi aramak için kullanılır. Mesela report.txt nesnesini aramak için;

```
Object obj = ctx.lookup("report.txt");
```

Kod kesimi işletilir. Lookup metodundan dönen servis sağlayıcıya göre değişir. Ama büyük ihtimalle ilk başta atanan nesne döner. Bu sefer Java.io.File nesnesinin dönmesi beklenir.

```
import java.io.File;
...
File f = (File)ctx.lookup("report.txt");

/****************************************/

class Lookup {
    public static void main(String[] args) {

      // Set up the environment for creating the initial context
      Hashtable env = new Hashtable(11);
```

```
        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.fscontext.RefFSContextFactory");
        env.put(Context.PROVIDER_URL, "file:c:\\1\\");

        try {
            // Create the initial context
            Context ctx = new InitialContext(env);

            // Perform lookup and cast to target type
            File f = (File) ctx.lookup("report.txt");

            System.out.println(f);

            // Close the context when we're done
            ctx.close();
        } catch (NamingException e) {
            System.out.println("Lookup failed: " + e);
        }
    }
}
```

Burada daha önce Setup.java kütüğünü çalıştırdığımızı varsayıyoruz. Ben parametre olarak 'c:\1\' vermiştim. Burada da env.put(Context.PROVIDER_URL, "file:c:\\1\\"); oldu.
Siz kendi parametrenize göre Context.PROVIDER_URL değişkenini günleyin.

İçeriği listeleme

Her defasında bir nesne almak yerine Context.lookup()[api] ile tüm içeriği bir kerede listeleyebilirsiniz. Biri bağlantıları diğeri sınıf-isim ikillerini döndüren iki lookup metodu vardır.

Context.list Metodu

Context.list()[api] metodu NameClassPair[api] sınıfının bir numaralandırmasını döndürür. Her NameClassPair bir nesne ismi ve bir de sınıf isminden oluşur. Aşağıdaki kod 'awt' dizininin içeriğini listeler.(o dizinde bulunan alt dizinler)

```
NamingEnumeration list = ctx.list("awt");

while (list.hasMore()) {
    NameClassPair nc = (NameClassPair)list.next();
    System.out.println(nc);
}
```

Başka bir örnek :

```
import javax.naming.*;
import java.util.Hashtable;

class List {
    public static void main(String[] args) {
```

```
        // Set up the environment for creating the initial context
        Hashtable env = new Hashtable(11);
        env.put(Context.INITIAL_CONTEXT_FACTORY,
            "com.sun.jndi.fscontext.RefFSContextFactory");
        env.put(Context.PROVIDER_URL, "file:c:\\1\\");

        try {
            // Create the initial context
            Context ctx = new InitialContext(env);

            // Get listing of context
            NamingEnumeration list = ctx.list("awt");

            // Go through each item in list
            while (list.hasMore()) {
              NameClassPair nc = (NameClassPair)list.next();
              System.out.println(nc);
            }

            // Close the context when we're done
            ctx.close();
        } catch (NamingException e) {
            System.out.println("List failed: " + e);
        }
    }
}
```

Örneğin çıktısı şu şekilde olacaktır

```
accessibility: javax.naming.Context
color: javax.naming.Context
datatransfer: javax.naming.Context
dnd: javax.naming.Context
event: javax.naming.Context
font: javax.naming.Context
geom: javax.naming.Context
im: javax.naming.Context
image: javax.naming.Context
peer: javax.naming.Context
print: javax.naming.Context
swing: javax.naming.Context
```

Context.listBindings Metodu

Context.listBindings()[api] metodu Binding[api] nin bir numaralandırmasını döndürür. Binding NameClassPair sınıfının bir altsınıfıdır. Ve sınıfın kendisini de içerir. Aşağıdaki kod 'awt' altındaki tüm bağlantıları ve nesneleri döker.

```
NamingEnumeration bindings = ctx.listBindings("awt");

while (bindings.hasMore()) {
    Binding bd = (Binding)bindings.next();
    System.out.println(bd.getName() + ": " + bd.getObject());
}
```

örnek:

```
import javax.naming.*;
import java.util.Hashtable;
class ListBindings {
    public static void main(String[] args) {

      // Set up the environment for creating the initial context
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL, "file:c:\\1\\");

      try {
          // Create the initial context
          Context ctx = new InitialContext(env);

          // Get listing of context
          NamingEnumeration bindings = ctx.listBindings("awt");

          // Go through each item in list
          while (bindings.hasMore()) {
            Binding bd = (Binding)bindings.next();
            System.out.println(bd.getName() + ": " + bd.getObject());
          }

          // Close the context when we're done
          ctx.close();
      } catch (NamingException e) {
          System.out.println("List Bindings failed: " + e);
      }
    }
}
```

Örneğin çıktısı aşağıdaki gibi olmalıdır:

```
accessibility: com.sun.jndi.fscontext.RefFSContext@a83b8a
color: com.sun.jndi.fscontext.RefFSContext@dd20f6
datatransfer: com.sun.jndi.fscontext.RefFSContext@1d1acd3
dnd: com.sun.jndi.fscontext.RefFSContext@a981ca
event: com.sun.jndi.fscontext.RefFSContext@8814e9
font: com.sun.jndi.fscontext.RefFSContext@1503a3
geom: com.sun.jndi.fscontext.RefFSContext@1a1c887
im: com.sun.jndi.fscontext.RefFSContext@743399
image: com.sun.jndi.fscontext.RefFSContext@e7b241
peer: com.sun.jndi.fscontext.RefFSContext@167d940
print: com.sun.jndi.fscontext.RefFSContext@e83912
swing:  com.sun.jndi.fscontext.RefFSContext@1fae3c6
```

Bir NamingEnumeration nesnesini sonlandırma

NamingEnumeration[api] üç yolla sonlanabilir.

- NamingEnumeration.hasMore()[api] false döndürürse normal olarak

- NamingEnumeration.close()[api] ile işimiz bittiği zaman biz kapatabiliriz.
- hasMore() yada next()[api] bir NamingException[api] fırlatır.

Bir bağlantıyı Ekleme silme günleme

Context.bind()[api] nesnesi bir bağlantıyı eklemek için kullanılır. Mesela ;

```
// bağlanacak nesneyi yarat
Fruit fruit = new Fruit("orange");

// bağla
ctx.bind("favorite", fruit);
```

Eğer iki kere eklemeye kalkarsanız bir NameAlreadyBoundException[api] aykırı durumu alırsınız ki bu da öyle bir isim var demektir. Birden fazla olan denemelerinizde yaniden bağla anlamında rebind()[api] 'ı kullanın.

```
// bağlanacak nesneyi yarat
Fruit fruit = new Fruit("lemon");

// bağla
ctx.rebind("favorite", fruit);
```

gibi.

unbind()[api] sınıfı ise bağlı bir nesneyi koparmaya yarar.

```
// Kopar
ctx.unbind("favorite");
```

gibi.

Bir nesneyi yeniden adlandırma

Context.rename()[api] metodu bu işi sizin için yapar. Mesela

```
// bir reporun adını değiştirme
ctx.rename("report.txt", "old_report.txt");
```

Mesela:

```
class Rename {
    public static void main(String[] args) {

      // Set up the environment for creating the initial context
      Hashtable env = new Hashtable(11);
      env.put(Context.INITIAL_CONTEXT_FACTORY,
          "com.sun.jndi.fscontext.RefFSContextFactory");
      env.put(Context.PROVIDER_URL, "file:c:\\1\\");

      try {
          // Create the initial context
```

```
            Context ctx = new InitialContext(env);

            // Rename to old_report.txt
            ctx.rename("report.txt", "old_report.txt");

            // Check that it is there using new name
            Object obj = ctx.lookup("old_report.txt");
            System.out.println(obj);

            // Rename back to report.txt
            ctx.rename("old_report.txt", "report.txt");

            // Check that it is there with original name
            obj = ctx.lookup("report.txt");
            System.out.println(obj);

            // Close the context when we're done
            ctx.close();
        } catch (NamingException e) {
            System.out.println("Rename failed: " + e);
        }
    }
}
```

gibi.

Alt İçerikler Oluşturma ve Silme

createSubcontext()[api] oluşturmak istediğiniz içeriğin ismini alarak içeriği sizin için oluşturur.

```
// içerik oluşturma
Context result = ctx.createSubcontext("new");
```

Silmek te aynı derecede kolaydır:

```
// içerik silme
ctx.destroySubcontext("new");
```

Dizinleme İşlemleri

JNDI 'ı aşağıdaki gibi dizin işlemlerini gerçekleştirmek için kullanabilirsiniz.

- Nesnenin özniteliklerini okumak
- Nesnenin özniteliklerini değiştirmek
- Dizini aramak
- Melez isimlendirme ve dizinleme işlemleri yapmak

Kurulum

Dizinleme servislerinden yararlanmak için kullanacağınız servis sağlayıcısını ve isim uzayını (namespace) belirlemek gerekir başka bir servis sağlayıcısı veya isim uzayında uygulamalar farklı davranabilir.

Örnek bir başlangıç içeriği aşağıdaki gibidir :

```
//başlangıç değerlerinin kurulması
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY,
    "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://localhost:389/o=JNDITutorial");
DirContext ctx = new InitialDirContext(env);
```

KAYNAKLAR

Sun ‘ın resmi sitesi (www.sun.com)
Berkeley üniversitesi çevrimiçi dokümanları
Jndi resmi sitesi (http://java.sun.com/products/jndi/)