Inventory

JNDI variables, CDI, Groovy mix project

Óbudai Egyetem, Java Enterprise Edition
Műszaki Informatika szak
Labor 13

Bedők Dávid
2016.12.05.
v0.5

Feladat

Egy egyszerű **Inventory** alkalmazás létrehozása, melyben az egyes tételeknek (`InventoryItem`) az alábbi adatát tároljuk memóriában: *referencia* (`String`), *megnevezés* (`String`), *típus* (enum: BOOK, DISK, MAGAZINE, stb.) és *érték* (`int`).

A feladat célja eddig nem érintett területek bemutatása (a már bemutatott területeket pedig leegyszerűsíti amennyire lehetséges).

- JEE **Interceptors**
- **CDI**
- JNDI-on keresztüli alkalmazás konfiguráció
- JSON annotációk

Részfeladat

Készítsünk egy weboldalt (http://localhost:8080/inventory/JSPList), mely egy táblázatban megjeleníti a memóriában megtalálható BOOK típusú tételeket.

- inv-webservice
  - `JSPListController` (WebServlet)
  - `list.jsp`
- inv-ejbservice
  - `InventoryFacade` (Local EJB)
    - `List<InventoryItem> getInventories(InventoryType type) throws AdaptorException;`
    - `InventoryFacadeImpl` (SLSB implementáció)
- inv-persistence
  - `InventoryHolder` (Local EJB)
    - `List<InventoryItem> list(InventoryType type);`
    - `InventoryHolderImpl` (SSB implementáció)

Naplózzuk a `getInventories()` hívásokat és mérjük a lefutási időt!

Naplózás továbbfejlesztése

Gyakori DEBUG naplózás technika a service metódusok elején, esetleg közvetlenül a visszatérésük előtt egy DEBUG logot készíteni (milyen paraméterekkel került a service meghívásra, illetve mi lett az eredménye mindennek). Jó volna, ha nem kellene minden metódust beburkolni ezzel. Valamilyen **AOP** (Aspect-Oriented Programming) megoldás volna itt kézenfekvő.

server.log

```
13:36:48,055 INFO
[hu.qwaevisz.inventory.ejbservice.interceptor.LoggedInterceptor]
(http-/127.0.0.1:8080-3) Entering:
hu.qwaevisz.inventory.ejbservice.facade.InventoryFacadeImpl.getInventories(BOOK)
13:36:48,057 INFO
[hu.qwaevisz.inventory.ejbservice.interceptor.LoggedInterceptor]
(http-/127.0.0.1:8080-3) Exiting:
hu.qwaevisz.inventory.ejbservice.facade.InventoryFacadeImpl.getInventories(BOOK)
- running time: 1 millisecond(s)
```

Interceptors

- JSR 318, Enterprise JavaBeans 3.1
- Eljárásokkal összefüggésben illetve életciklus elemek körül definiálhatóak.
- **Regisztráció**: Osztályként vagy metódusként is implementálhatóak (mi osztályként valósítjuk meg csak, a Separation of Concern végett (https://en.wikipedia.org/wiki/Separation_of_concerns))
- **Kötés**: A felhasználás helyén konfigurálható a `javax.interceptor.Interceptors` (s!) annotációval, XML konfigurációval illetve CDI segítségével is. Javasolt az utóbbi alkalmazása, de bemutatásra kerül a többi is.

Interceptor Metadata Annotation	Description
javax.interceptor.AroundInvoke	Designates the method as an interceptor method.
javax.interceptor.AroundTimeout	Designates the method as a timeout interceptor, for interposing on timeout methods for enterprise bean timers.
javax.annotation.PostConstruct	Designates the method as an interceptor method for post-construct lifecycle events.
javax.annotation.PreDestroy	Designates the method as an interceptor method for pre-destroy lifecycle events.

Regisztráció (önálló osztályként)

A `javax.interceptor.Interceptor` annotáció segítségével:

LoremInterceptor.java

```
@Interceptor
public class LoremInterceptor implements Serializable {
    [..]
}
```

Vagy a `beans.xml`-ben való felvétellel (`inv-ejbservice|src|main|resources`):

beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">

     <interceptors>
          <class>hu.qwaevisz.inventory.ejbservice.interceptor.LoggedInterceptor</class>
     </interceptors>

</beans>
```

Interceptor készítése

LoggedInterceptor.java

```
package hu.qwaevisz.inventory.ejbservice.interceptor;
[..]
@Interceptor
public class LoggedInterceptor implements Serializable {
    @AroundInvoke
    public Object logMethodInvocations( InvocationContext context) throws
Exception {
        StringBuilder info = new StringBuilder();
        info.append(context.getTarget().getClass().getName()).append(".").appen
d(context.getMethod().getName()).append("(");
        Object[] parameters = context.getParameters();
        [..]
        info.append(")");
        LOGGER.info("Entering: " + info.toString());
        long start = System.currentTimeMillis();
        Object result = context.proceed();
        long end = System.currentTimeMillis();
        LOGGER.info("Exiting: " + info.toString() + " - running time: " + (end
- start) + " millisecond(s)");
        return result;
    }
}
```

Bejárhatjuk a paramétereket, azok értékeit, típusait. Saját annotációk használata esetén Reflection API-val további vezérlést is beépíthetünk!

Egy service hívás köré szeretnénk ezt beágyazni (`@AroundInvoke` annotáció). A valós hívást mi kezdeményezzük (`context.proceed()`), és ennek megfelelően akár meg is akadályozhatjuk, vagy módosíthatjuk a kimenetét mielőtt visszatérünk.

Kötés (vagy-vagy)

Ezek helyett a CDI binding egy még kényelmesebb megoldás, a CDI megismerése után ezt fogjuk alkalmazni!

InventoryFacadeImpl.java

```
@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl implements InventoryFacade {
    @Override
    @Interceptors({ LoggedInterceptor.class })
    public void test() {
        [..]
    }
}
```

ejb-jar.xml

```
<interceptor-binding>

<target-name>hu.qwaevisz.inventory.ejbservice.facade.InventoryFacadeImpl</target
-name>

<interceptor-class>hu.qwaevisz.inventory.ejbservice.interceptor.LoggedIntercepto
r</interceptor-class>
    <method-name>test</method-name>
</interceptor-binding>
```

Contexts and Dependency Injection for the Java EE Platform (CDI)

- JSR 299, JSR 330, JSR 316
  - Expression Language (EL) integráció Java Server Faces illetve JSP lapokban való componens eléréshez
  - Osztályok, injektált elemek dekorációja
  - Interceptor-ok bekötése typesafe módon
    - ezt fogjuk használni, e miatt nem részleteztük az `Interceptors` annotációt és az XML konfigurációt az interceptor-ok kapcsán)
  - Eseménykezelés
  - Web conversion scope, mely kiegészíti a request/session/application Java Servlet specifikációt
  - Service Provider Interface (SPI), mely lehetővé teszi 3rd party keretrendszerek JEE 6 környezetbe ágyazását
- A CDI használatával a `String` kulcs alapján működő erőforrás lekérést **type-safe** módon tudjuk kezelni.

CDI

- A `@EJB` annotáció használata helyett a CDI `@Inject` annotációját is használhatjuk a továbbiakban.
  - Az `@EJB` annotációt csak ott lehet használni, ahol “managed” elemek vannak (pl. egy sima POJO-ban nem, de egy servlet-ben, session bean-ben igen, vagy bármely osztályban ami context-be kerül valamilyen módon).
  - A CDI használható szinte az összes Java osztályban, session bean-ekben, stb. (de nem: MDB-ben és Remote EJB-ben).
- Classpath-ra egy `beans.xml` elhelyezése szükséges azokban a module-okban, ahol használni szeretnénk (`src/main/resources/beans.xml`). A használat elkezdéséhez elegendő egy teljesen üres file is (http://www.cdi-spec.org/faq/)

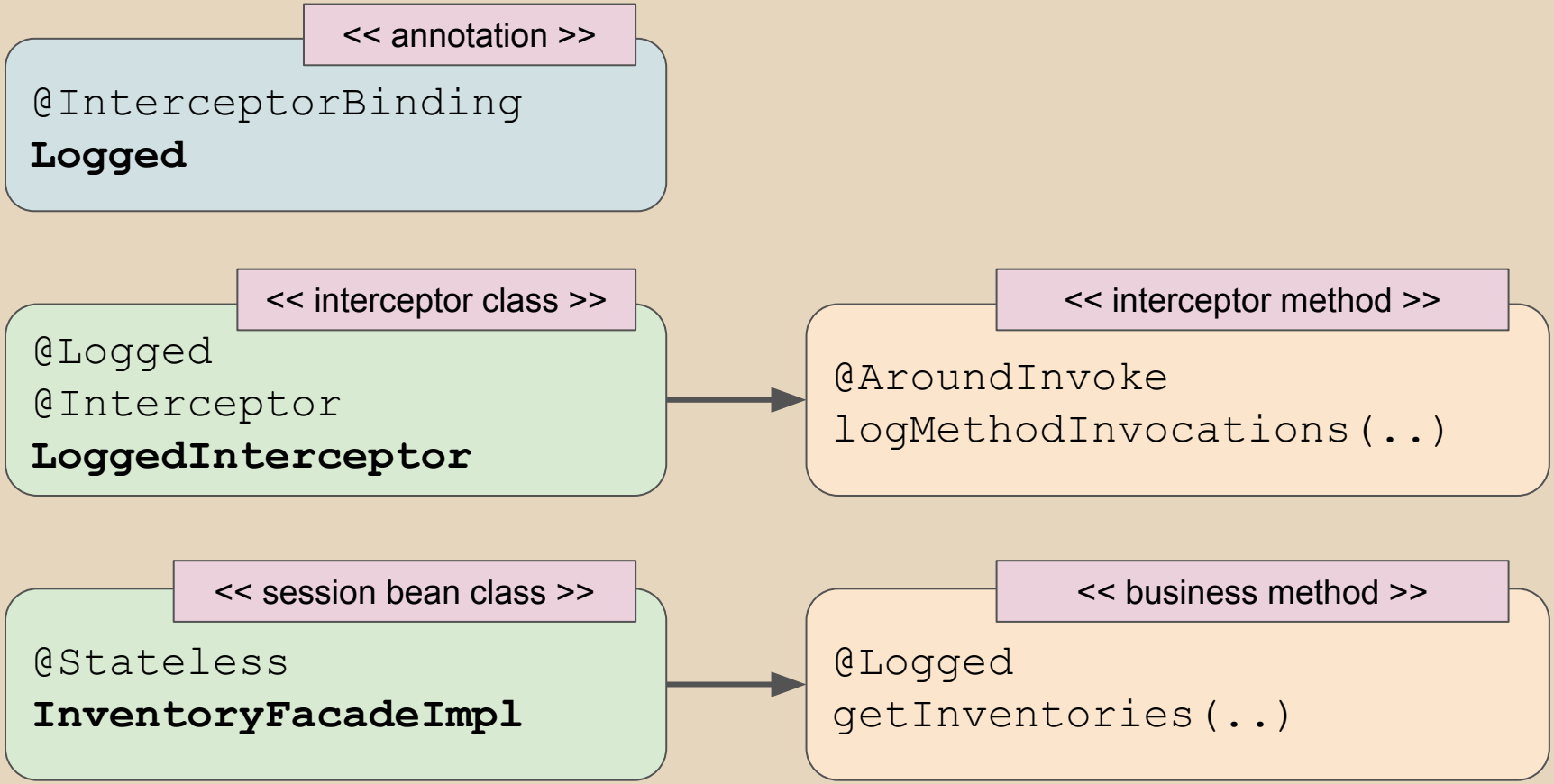
@InterceptorBinding

Interceptor-ok bekötésére használható annotáció. Az az annotáció rendelkezik a `@InterceptorBinding` annotációval, melyet használni szeretnénk interceptorok kötésére.

Ezt követően ez az annotáció megjelöli az Interceptor-t tartalmazó osztályt, és az Interceptor felhasználásának a helyét is!

Fontos: Az `@InterceptorBinding` **ANNOTÁCIÓK** annotálására használható!

Áttekintés


<< annotation >>
@InterceptorBinding
Logged
<< interceptor class >>
@Logged
@Interceptor
LoggedInterceptor
<< interceptor method >>
@AroundInvoke
logMethodInvocations(..)
<< session bean class >>
@Stateless
InventoryFacadeImpl
<< business method >>
@Logged
getInventories(..)

Naplózás InterceptorBinding annotációja

Logged.java

```
package hu.qwaevisz.inventory.ejbservice.interceptor;

import java.lang.annotation.ElementType;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import javax.interceptor.InterceptorBinding;

@Inherited
@InterceptorBinding
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD })
public @interface Logged {

}
```

Interceptor módosítása

LoggedInterceptor.java

```
package hu.qwaevisz.inventory.ejbservice.interceptor;

[..]

@Logged
@Interceptor
public class LoggedInterceptor  implements Serializable {

    [..]

    @AroundInvoke
    public Object logMethodInvocations(InvocationContext context)  throws
Exception {
        [..]
    }

    [..]

}
```

Interceptor felhasználása

InventoryFacadeImpl.java

```
package hu.qwaevisz.inventory.ejbservice.facade;

[..]

@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl  implements InventoryFacade {

    @EJB
    private InventoryHolder inventoryHolder;

    @Logged
    @Override
    public List<InventoryItem> getInventories(InventoryType type)  throws
AdaptorException {
        return this.inventoryHolder.list(type);
    }

}
```

A `@Logged` annotációval minden olyan metódust megjelölhetünk, melynél szeretnénk a hívás elején és végén logolni, illetve a kettő között eltelt futási időt mérni.

Részfeladat

Készítsünk egy RESTful service-t (http://localhost:8080/inventory/api/search/IPS65), mely referencia alapján lekér egy Inventory elemet, de a facade rétegben az árát 20%-al csökkenti promóciós időszakban. Normál időszakban azonban az eredeti árat adja vissza.

Az ármódosítást egy `CostService` határozza meg, mely egy mezei interface legyen, egy `NormalCostServiceImpl` és egy `PromotionalCostServiceImpl` implementációval. Promóciós időszakban **CDI Qualifier**-ek segítségével inject-áljuk a megfelelő implementációt.

Qualifiers

A Qualifier egy annotáció, mely bean-ek annotálására használható. Egy annotáció attól lesz qualifier, hogy rendelkezik a `javax.inject.Qualifier` annotációval.

A Qualifier-ek segítségével **type-safe** módon lehet használni pl. egy EJB két különféle implementációját (a kért Qualifier-rel rendelkezőt lehet inject-álni), vagy mindehhez nem is kellenek session bean-ek.

Ha egy EJB-nek több implementációja van a bean nevét jelezni kell az annotációk használata során (pl. `@Stateless(name="lorem")` és `@EJB(beanName="lorem")`). Ez teszi ezt a megoldást nem type-safe módszerré.

Megjegyzés: Természetesen a String értékek helyet lehet egy konstanst használni.

Ha egy bean-nek nincs Qualifier-e, akkor automatikusan kap egy `@Default` qualifier-t.

Implementáció

- inv-webservice
  - `SearchRestService` (RESTful service /search)
    - `InventoryItem getInventory(@PathParam("reference") String reference) throws AdaptorException;`
    - `SearchRestServiceBean` implementáció
- inv-ejbservice
  - `InventoryFacade` (Local EJB)
    - `InventoryItem getInventory(String reference) throws AdaptorException;`
    - `InventoryFacadeImpl` (SLSB implementáció)
- inv-persistence
  - `InventoryHolder` (Local EJB)
    - `InventoryItem get(String reference);`
    - `InventoryHolderImpl` (SSB implementáció)
- inv-ejbservice
  - `InventoryFacadeImpl` (SLSB implementáció)
    - `CostService` (CDI managed interface)
      - `int getCost(int originalValue);`
      - `NormalCostServiceImpl` (CostService implementáció)
      - `PromotionalCostServiceImpl` (CostService implementáció)

CostService és qualifiers

CostService.java

```
package hu.qwaevisz.inventory.ejbservice.cost;
public interface CostService {
    int getCost(int originalValue);
}
```

Standard.java

```
import javax.inject.Qualifier;
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD,
ElementType.PARAMETER })
public @interface Standard {
}
```

Discount.java

```
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD,
ElementType.PARAMETER })
public @interface Discount {
}
```

CostService implementációk

NornalCostServiceImpl.java

```
package hu.qwaevisz.inventory.ejbservice.cost;
@Standard
public class NormalCostServiceImpl  implements CostService {
    @Override
    public int getCost(int originalValue) {
        return originalValue;
    }
}
```

PromotionalCostServiceImpl.java

```
package hu.qwaevisz.inventory.ejbservice.cost;
@Discount
public class PromotionalCostServiceImpl  implements CostService {

    private static final float DISCOUNT_PERCENT = 20;

    @Override
    public int getCost(int originalValue) {
        return Math.round(originalValue * (1 - (DISCOUNT_PERCENT / 100)));
    }
}
```

Qualifier alapú injection

InventoryFacadeImpl.java

```
package hu.qwaevisz.inventory.ejbservice.facade;

@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl  implements InventoryFacade {

    @Inject
    @Discount
    private CostService costService;

    @EJB
    private InventoryHolder inventoryHolder;

    @Logged
    @Override
    public InventoryItem getInventory(String reference)  throws AdaptorException
{
        InventoryItem inventory =  this.inventoryHolder.get(reference);
        inventory.setValue( this.costService.getCost(inventory.getValue()));
        return inventory;
    }

}
```

Részfeladat

Készítsünk egy RESTful service-t (http://localhost:8080/inventory/api/search/stub/IPS65) mely az Inventory elem referenciája alapján visszaad egy *Stub*-ot, melyben megjelenik a termék (`product`), mely az elem nevének és referenciájának a konkatenációja, illetve az ár (`price`), mely az értékéből és az aktuális pénznemből tevődik össze. Az árak az Inventory-ban EURO-ban értelmezettek.

```
{"price":"19825 HUF","product":"IPS65-Ipsum"}
```

- Legyen lehetőség a *Stub* kulcsainak programozott konfigurációjára, melyhez ne kelljen a forráskód mezőneveit változtatni (ahogy korábban XML-nél, itt is annotációval vezéreljük a JSON kimenetet).
- Az aktuális pénznem (példában **HUF**) egy indítási konfigurációs paraméter legyen, melyet JNDI-ban tároljunk el "`java:global/currency`" kulccsal, ahogyan az EUR2HUF váltószám is ("`java:global/exchangeRate`" kulccsal).

JNDI paraméter konfigurációja

standalone.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<server xmlns="urn:jboss:domain:1.7">
    <profile>
        ...
        <subsystem xmlns="urn:jboss:domain:naming:1.4">
            <remote-naming/>
            <bindings>
                <simple name="java:global/currency" value="HUF"/>
                <simple name="java:global/exchangeRate" value="305" type="int" />
            </bindings>
        </subsystem>
        ...
    </profile>
    ...
</server>
```

Az egyszerű értékek mellett (`simple`) van lehetőség objektum referenciát is (`object-factory`) és JNDI alias-t is definiálni (`lookup`).

A változók nevei nem kezdődhetnek akárhogyan. A container már meglévő hierarchiájába építi be ezeket, saját elnevezési szabályai szerint. A `java:global` olyan változókat definiál, melyek scope-ja a teljes alkalmazás szerverre érvényes.

Implementáció

- inv-webservice
  - `SearchRestService` (RESTful service)
    - `@Produces(MediaType.APPLICATION_JSON) InventoryItemStub getInventoryStub(@PathParam("reference") String reference) throws AdaptorException;`
    - `SearchRestServiceBean` impementation
- inv-ejbservice
  - call `InventoryHolder` EJB (inv-persistence)
  - call `InventoryItemConverter` POJO
    - `InventoryItem` átalaktása `InventoryItemStub`-ra
    - CDI segítségével a converter injectálása
    - JNDI erőforrások kikérése

Stub

InventoryItemStub.java

```
package hu.qwaevisz.inventory.ejbservice.domain;
import org.codehaus.jackson.annotate.JsonProperty;
public class InventoryItemStub {

    private String label;
    private String price;

    public InventoryItemStub(String label, String price) { [..] }

    @JsonProperty("product")
    public String getLabel() {
        return this.label;
    }

    public void setLabel(String label) {  this.label = label; }

    @JsonProperty("price")
    public String getPrice() {
        return this.price;
    }

    public void setPrice(String price) {  this.price = price; }
}
```

A példa kedvéért szándékosan más nevet használunk a `label` helyett.

A `@JsonProperty` annotációt a getter metódusokra helyezhetjük el, és ~ekvivalens a működése az `@XmlElement(name = "...")` annotációval.

Json függőség hozzáadása
inv-ejbservice

inv-ejbservice

```
dependencies {
    [..]
    compile group: 'org.codehaus.jackson', name: 'jackson-core-asl',
version: jacksonVersion
    compile group: 'org.codehaus.jackson', name: 'jackson-mapper-asl',
version: jacksonVersion
    [..]
}
```

build.gradle

[root]

```
[..]
ext {
    [..]
    jacksonVersion = '1.9.9'
    [..]
}
[..]
```

build.gradle

Converter

InventoryItemConverter.java

```
package hu.qwaevisz.inventory.ejbservice.converter;

import hu.qwaevisz.inventory.ejbservice.domain.InventoryItemStub;
import hu.qwaevisz.inventory.persistence.domain.InventoryItem;

public interface InventoryItemConverter {

    InventoryItemStub to(InventoryItem item);

}
```

Ahhoz hogy CDI-al ennek implementációját injectáljuk, nincs szükségünk semmilyen annotációra. Ha egy implementációja létezik csak, az megkapja automatikusan a `@Default` qualifier-t, és a `@Inject` qualifier nélküli használata a `@Default` qualifier-el rendelkező implementációt fogja használni.

Converter POJO impementáció

InventoryItemConverterImpl.java

```
package hu.qwaevisz.inventory.ejbservice.converter;

import javax.annotation.Resource;
import hu.qwaevisz.inventory.ejbservice.domain.InventoryItemStub;
import hu.qwaevisz.inventory.persistence.domain.InventoryItem;

public class InventoryItemConverterImpl  implements InventoryItemConverter {

    @Resource(lookup = "java:global/currency")
    private String currency;

    @Resource(lookup = "java:global/exchangeRate")
    private int exchangeRate;

    @Override
    public InventoryItemStub to(InventoryItem item) {
        String label = item.getReference() + "-" + item.getName();
        String price = item.getValue() *  this.exchangeRate +" "+  this.currency;
        return new InventoryItemStub(label, price);
    }
}
```

A `@Resource` annotáció a megadott JNDI név alapján kikéri az elemet a JNDI-ból. Az `InitialContext`-et ez esetben a container biztosítja.

Business method

InventoryFacadeImpl.java

```
package hu.qwaevisz.inventory.ejbservice.facade;
[..]
@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl  implements InventoryFacade {

    @EJB
    private InventoryHolder inventoryHolder;

    @Inject
    private InventoryItemConverter converter;

    @Override
    public InventoryItemStub getInventoryStub(String reference)  throws
AdaptorException {
        return this.converter.to(this.inventoryHolder.get(reference));
    }

}
```

Részfeladat

Amikor egy Inventory tételt lekérdeznek a rendszerből, értesíteni szükséges az éles vagy a teszt (fake) rendszer felhasználóját a lekérésről.

JEE **eseménykezeléssel** oldjuk meg a feladatot (ez is a CDI témaköre)!

Hogy az éles vagy a teszt rendszer felhasználójának üzenünk, szintén CDI által injectált implementációja legyen egy `Client`-nek, azonban most a két különböző qualifier annotáció helyett egy enum-al különböztessük meg a felhasználókat (hiszen több tétel esetén nagyon zavaró és egyben nem clean volna rengetek annotációt létrehozni!)

Client

Client.java

```
package hu.qwaevisz.inventory.ejbservice.domain;
public class Client {

    private final String reference;
    private final String name;

    public Client(String reference, String name) {
        this.reference = reference;
        this.name = name;
    }

    public String getReference() { return this.reference; }

    public String getName() { return this.name; }

    @Override
    public String toString() {
        return "Client [reference=" + this.reference + ", name=" +
this.name + "]";
    }
}
```

Egy Qualifier sok helyett

ClientFlag.java

```
package hu.qwaevisz.inventory.ejbservice.qualifier;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import javax.inject.Qualifier;

import com.ericsson.inventory.ejbservice.domain.ClientType;

@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD })
public @interface ClientFlag {

    ClientType value();

}
```

A *Qualifier* `value()` "attribútuma" tér vissza a megkülönböztetést hordozó *enum* egy példányával.
Így elérjük hogy 1 *Qualifier* és 1 *Enum* segítségével N különböző implementációt tudunk kezelni (N>2 esetén érdemes haszálni).

ClientType.java

```
package hu.qwaevisz.inventory.ejbservice.domain;
public enum ClientType {
    FAKE,
    LIVE
}
```

Kliensek annotációk nélkül

ClientHolder.java

```
package hu.qwaevisz.inventory.ejbservice.client;
import hu.qwaevisz.inventory.ejbservice.domain.Client;
public interface ClientHolder {
    Client getCurrent();
}
```

LiveClientHolder.java

```
package hu.qwaevisz.inventory.ejbservice.client;
import hu.qwaevisz.inventory.ejbservice.domain.Client;
@ClientFlag(ClientType.LIVE)
public class LiveClientHolder implements ClientHolder {
    @Override
    public Client getCurrent() {
        return new Client("MCF012", "Matthew C. Flores");
    }
}
```

FakeClientHolder.java

```
[..]
@ClientFlag(ClientType.FAKE)
public class FakeClientHolder implements ClientHolder {
    @Override
    public Client getCurrent() {
        return new Client("MCF012", "Matthew C. Flores");
    }
}
```

Eseménykezelés

- Esemény
  - POJO
  - `NotifierEvent`
- Eseménykezelő(k)
  - `NotifierEventHandler`
    - `@Observes NotifierEvent` (CDI annotáció)
  - Akármennyi eseménykezelő feliratkozhat egy eseményre
- Feliratkozás, regisztrálás, eseménykküldés
  - `@Inject private Event<NotifierEvent> notifier;`
  - `this.notifier.fire(new NotifierEvent(..));`

Esemény

NotifierEvent.java

```
package hu.qwaevisz.inventory.ejbservice.event;
[..]
public class NotifierEvent implements Serializable {

    public Client client;
    public String message;

    public NotifierEvent() {
    }

    public NotifierEvent(Client client, String message) {
        this.client = client;
        this.message = message;
    }

    @Override
    public String toString() {
        return "NotifierEvent [client=" + this.client + ", message=" +
this.message + "]";
    }
}
```

Eseménykezelő
egy lehetséges implementáció

NotifierEventHandler.java

```
package hu.qwaevisz.inventory.ejbservice.event;

import java.io.Serializable;
import javax.enterprise.event.Observes;
import org.apache.log4j.Logger;

public class NotifierEventHandler  implements Serializable {

    private static final Logger LOGGER =
Logger.getLogger(NotifierEventHandler. class);

    public void handle(@Observes NotifierEvent event) {
        LOGGER.info(event.client.getName()+" performs an event:
"+event.message);
    }

}
```

Business method
getInventory() módosítása

InventoryFacadeImpl.java

```
public class InventoryFacadeImpl implements InventoryFacade {

     @Inject
     @Discount
     private CostService costService;

     @Inject
     @ClientFlag(ClientType.LIVE)
     private ClientHolder clientHolder;

     @Inject
     private Event<NotifierEvent> notifier;

     @EJB
     private InventoryHolder inventoryHolder;

     @Logged
     @Override
     public InventoryItem getInventory(String reference) throws AdaptorException {
          InventoryItem inventory = this.inventoryHolder.get(reference);
          inventory.setValue(this.costService.getCost(inventory.getValue()));
          Client client = this.clientHolder.getCurrent();
          this.notifier.fire(new NotifierEvent(client, "Get " + inventory.getName() + " (ref: "
+ inventory.getReference() + ") inventory item."));
          return inventory;
     }

}
```

Paraméteres osztály inject-álása?

Mi van akkor, ha a beszúrandó példány létrehozásához kellenek paraméterek (ctor-ának vannak argumentumai)?

Ekkor nem tehetjük a Qualifier-t az osztályra, hiszen nem tudja létrehozni azt a container!

Ilyenkor használhatjuk a

`javax.enterprise.inject.Produces`

annotációt!

Feladat: legyen egy olyan “Custom” kliens is, mely nevét JNDI-ból kapja, és referenciája nevének első három karaktere legyen!

Custom Client holder

CustomClientHolder.java

```
package hu.qwaevisz.inventory.ejbservice.client;

import hu.qwaevisz.inventory.ejbservice.domain.Client;

public class CustomClientHolder  implements ClientHolder {

    private final Client client;

    public CustomClientHolder(String clientName) {
        String reference = clientName.substring(0, 3).toUpperCase();
        this.client = new Client(reference, clientName);
    }

    @Override
    public Client getCurrent() {
        return this.client;
    }

}
```

Nem tehetjük az osztályra a **`@ClientFlag(ClientType.CUSTOM)`** annotációt, mert nem tudja a container a clientName értékét!

Custom Client holder Factory

ClientType.java

```
package hu.qwaevisz.inventory.ejbservice.client;

import javax.enterprise.inject.Produces;
[..]

@ApplicationScoped
public class CustomClientFactory {

    @Resource(lookup = "java:global/customClientName")
    private String customClientName;


    @Produces
    @ClientFlag(ClientType.CUSTOM)
    public ClientHolder getCustomClient() {
        return new CustomClientHolder(this.customClientName);
    }

}
```

A `@Produces` annotációval ellátott elemeket a container összegyűjti, és egy adott típusú erőforrás kérésekor használja őket (legtöbbször csak egyszer futnak le).

standalone.xml-be:

```
<simple name="java:global/customClientName"
value="Dolores M. Putto" />
```

Ne keverjük össze a `javax.enterprise.inject.Produces` és a `javax.ws.rs.Produces` annotációkat!

Felhasználás:

```
@Inject
@ClientFlag(ClientType.CUSTOM)
private ClientHolder clientHolder;
```

Részfeladat

Az Inventory rendszerünk egy rendkívül hasznos új funkciója, hogy **véletlen szám**ot kérésre vissza tud adni. Mindez azért remek, mert így okot ad arra, hogy véletlen számot injectáljunk CDI segítségével. A feladat valójában arról szól, hogy miként injectálunk úgy egy erőforrást, hogy az minden ismételt elővétel során újból létrejöjjön (a létrehozási szakasz dinamikus paramétereket használ, pl. véletlen számot ez esetben).

Véletlen számok

Request:

http://localhost:8080/inventory/api/search/numbers/5

Response:

```
[91,64,25,77,66]
```

A véletlen számokat 0 és egy MAX érték között szeretnénk definiálni. A MAX egy nem dinamikus de erőforrásként injectált érték legyen (mert pl. valamilyen konfigurációból jön később, most csak egy konstans lesz, értéke legyen 100 és JNDI-ból olvassuk ki).

Maximum number resource

MaxNumber.java

```
package hu.qwaevisz.inventory.ejbservice.qualifier;
[..]
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD, ElementType.PARAMETER })
public @interface MaxNumber {
}
```

MaxNumberFactory.java

```
package hu.qwaevisz.inventory.ejbservice.service;
[..]
@ApplicationScoped
public class MaxNumberFactory {

	@Resource(lookup = "java:global/maxNumber")
	private int maxValue;

	@Produces
	@MaxNumber
	public int getMaxNumber() {
		return this.maxValue;
	}

}
```

Fontos! Ahhoz hogy ez az érték frissüljön az alkalmazásban, az alkalmazás szervert újra kell indítani! Konfigurációs paraméterre pl. a JMX lehet megoldás.

JNDI via JBoss CLI
java:global/maxNumber

```
> jboss-cli --connect
[/] /subsystem=naming/binding=java\:global\/maxNumber:read-resource
{
    "outcome" => "success",
    "result" => {
        "binding-type" => "simple",
        "cache" => undefined,
        "class" => undefined,
        "environment" => undefined,
        "lookup" => undefined,
        "module" => undefined,
        "type" => "int",
        "value" => "100"
    }
}
[/] /subsystem=naming/binding=java\:global\/maxNumber:write-attribute(name=value,value=200)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

A JNDI névből a `:` és a `/` karaktert escape-elni kell (`\:` és `\/`).
Az olvasó operation neve `read-resource`, míg az író művelet neve `write-attribute`.

Fontos! A fenti módszerrel ellenőrizhető, hogy a JNDI paramétereket az alkalmazáson belül nem tudjuk "átírni", reload-olni. Akkor sem, ha InitialContext-ből kérjük ki. A szerver újraindítása szükséges.

A JNDI fát a webconsole-on (http://localhost:9990/console/App.html#naming) is tudjuk böngészni (Runtime | JNDI view).

Random number resource

Random.java

```
package hu.qwaevisz.inventory.ejbservice.qualifier;
[..]
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ ElementType.TYPE, ElementType.METHOD, ElementType.FIELD, ElementType.PARAMETER })
public @interface Random {
}
```

MaxNumberFactory.java

```
@ApplicationScoped
public class RandomNumberFactory {
     private final java.util.Random random = new java.util.Random(System.currentTimeMillis());

     @Inject
     @MaxNumber
     private int maxNumber;

     @Logged
     @Produces
     @Random
     public int next() {
          return this.random.nextInt(this.maxNumber);
     }
}
```

A `@MaxNumber` egyszer lesz injectálva, hiába írjuk át a JNDI-ban az értékét, nem fog érvényesülni. Ellentétben majd a `@Random` értékével, aminél nagyon rossz lenne ha nem inicializálódni ismét!

Business method

InventoryFacadeImpl.java

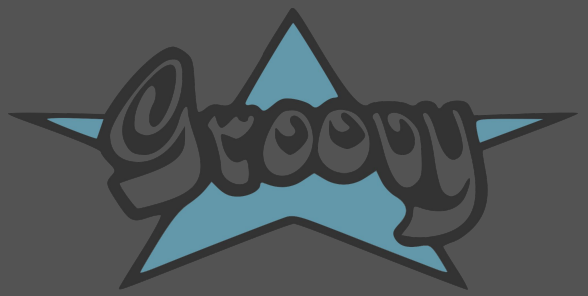
```
package hu.qwaevisz.inventory.ejbservice.facade;
[..]
@Stateless(mappedName = "ejb/inventoryFacade")
public class InventoryFacadeImpl  implements InventoryFacade {

    @Inject
    @Random
    private Instance<Integer> randomNumber;


    @Logged
    @Override
    public List<Integer> getRandomNumbers( int quantity)  throws AdaptorException
{
        List<Integer> result =  new ArrayList<>();
        for (int i = 0; i < quantity; i++) {
            result.add(this.randomNumber.get());
        }
        return result;
    }

}
```

A generikus `Instance<T>` típus esetén a `T` helyére írjuk az injektálandó dinamikus erőforrás típusát, és a a `T get()` metódus meghívásakor az inicializálás mindig ismételtelten meg fog történni.

Groovy-Java mixed project

A Java bár fejlődik, nem olyan hibrid nyelv, mint pl. a C#. Viszont a bytecode nem Java specifikus, annak szintjén a lehetőségeink kibővülnek. Ha már a build tool miatt (*gradle*) a Grovvy nyelvvel barátkozunk, kézenfekvő a Groovy-t a production kód szintjére is behozni. Így pl. *lambda* kifejezéseket tudunk készíteni egy Java 7-es környezetben is!

Groovy



http://www.groovy-lang.org/

Latest: **2.4.7**

JVM Required: 1.7+

Apache Groovy (from 2.4.4)

- A multi-faceted language for the Java platform
- Optionally typed and dynamic language
- Static-typing and static compilation capabilities

Install: unzip

(System) Environment Variable:

GROOVY_HOME - c:\apps\groovy-2.4.7\

Path kiegészítése: %GROOVY_HOME%\bin

```
>groovy --version
Groovy Version: 2.4.7
JVM: 1.8.0_102 Vendor:
Oracle Corporation OS:
Windows 7
```

Groovy - JBoss module

[JBOSS-HOME]\modules\org\codehaus\groovy\main\

- FROM: groovy-2.4.7\embeddable\groovy-all-2.4.7.jar
- module.xml

module.xml

```
<module xmlns="urn:jboss:module:1.1" name=" org.codehaus.groovy">
    <resources>
        <resource-root path=" groovy-all-2.4.5.jar"/>
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.servlet.api"/>
        <module name="org.apache.commons.logging"/>
    </dependencies>
</module>
```

`NoClassDefFoundError` hibák esetén tipikusan ClassLoader hiba áll fenn. Ilyenkor a függőségek beállítása hiányozhat.

standalone.xml

```
..
<subsystem xmlns="urn:jboss:domain:ee:1.2">
    <global-modules>
        ...
        <module name="org.codehaus.groovy" slot="main"/>
    </global-modules>
</subsystem>
```

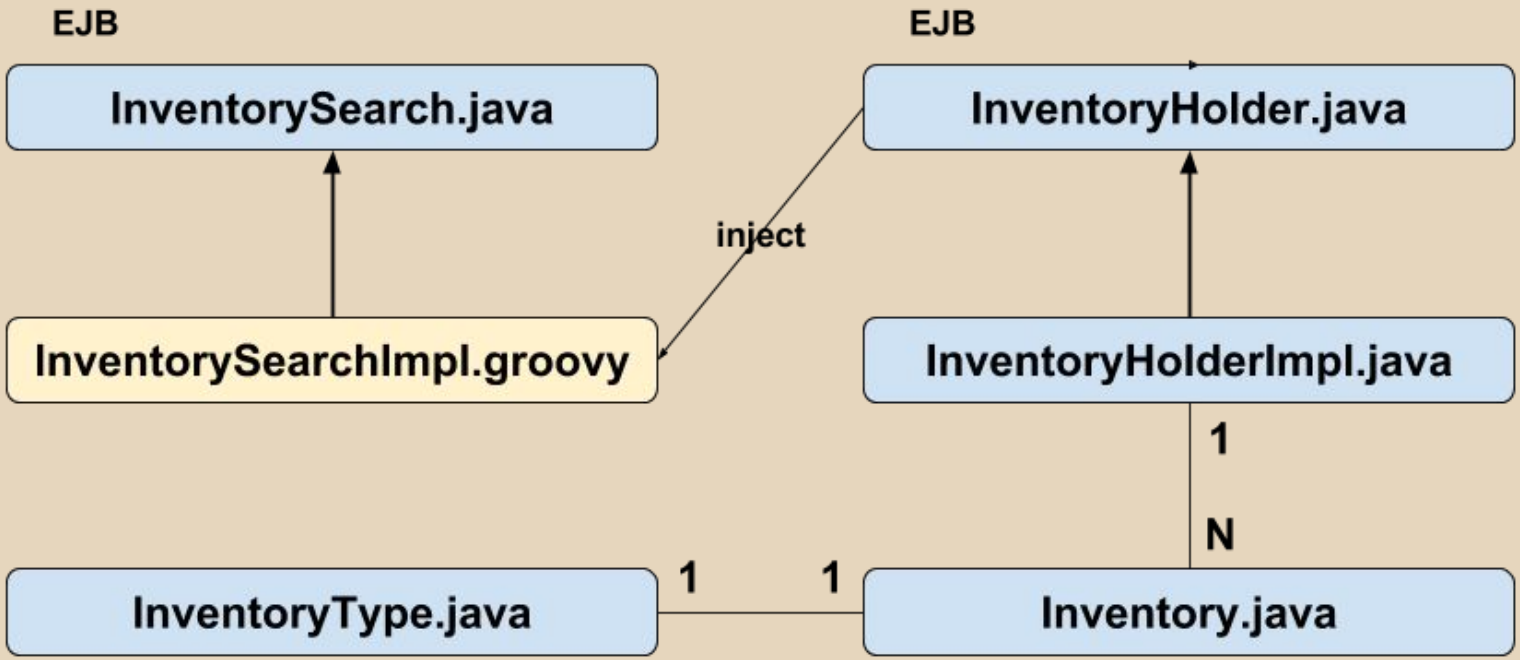
Részfeladat

Készítsünk egy RESTful metódust (pl.: http://localhost:8080/inventory/api/search/list/BOOK/Lorem), mely visszaadja azokat az Inventory elemeket, melyek egy megadott típusnak (*BOOK*) és név töredéknek (*Lorem*) megfelelnek (tipikus kereső művelet).

A persistence rétegben a Java interface implementációja **Groovy nyelven** készüljön el!

Persistence réteg

Az új Stateless EJB interface oldala Java-ban legyen, míg az implementáció teljes egészében Groovy-ban!



Groovy mixed project build file

inv-persistence

```
apply plugin: 'groovy'

sourceSets.main.java.srcDirs = []
sourceSets.main.groovy.srcDirs = ['src/main/java', 'src/main/groovy']

dependencies {
    compile group: 'javax', name: 'javaee-api', version: jeeVersion
    compile group: 'org.codehaus.groovy', name: 'groovy' , version:
groovyVersion
}
```

Source set beállítások: A java kódot önmagában ne fordítsa le (mivel kell neki a Groovy, így elbukna), a groovy-t viszont a Java-val együtt (mixed mód). A Groovy nyelvben van helye a Java-nak, így semmilyen problémát nem okoz számára az egyébként teljesen Java nyelven írt osztályok lefordítása.

build.gradle

```
>gradle clean build
```

Kereső szolgáltatás

InventorySearchImpl.groovy

```
package hu.qwaevisz.inventory.persistence.service

import hu.qwaevisz.inventory.persistence.domain.InventoryItem
import hu.qwaevisz.inventory.persistence.domain.InventoryType

import javax.ejb.EJB
import javax.ejb.Stateless

@Stateless
class InventorySearchImpl implements InventorySearch {

    @EJB
    private InventoryHolder holder

    @Override
    List<InventoryItem> list(InventoryType type, String nameTerm) {
        holder.getAll().findAll { it -> it.type == type &&
it.name.startsWith(nameTerm) }
    }
}
```

Ez **groovy** kód, de sok helyen él Java-s elemekkel, mely a Groovy-ban teljesen megengedett és szintaktikailag helyes!

Lamda expression. Ellenőrizd a létrejövő class állományokat egy Java 7-es környezetben!

Tesztelés

GET http://localhost:8080/inventory/api/search/list/BOOK/Lorem


```
[{
    "reference": "LOR42",
    "name": "Lorem10",
    "type": "BOOK",
    "value": 42
},
{
    "reference": "LOR78",
    "name": "Lorem20",
    "type": "BOOK",
    "value": 78
},
{
    "reference": "LOR34",
    "name": "Lorem30",
    "type": "BOOK",
    "value": 34
}]
```

Részfeladat

Készítsük el a korábban elkészült JSP lapot Groovy nyelven **Groovy servlet**-ként és GSP (**Groovy Server Pages**) lapként is.

Mindegyik megoldás (a JSP-t is beleértve) valójában nem más, mint egy Java Servlet-nek az elkészítése egy kicsit célnak megfelelőbb eszközzel (a Java-hoz viszonyítva).

Groovy Servlet és Groovy Server Pages

web.xml

```
<web-app ...>
    <servlet>
        <servlet-name>GroovyServlet</servlet-name>
        <servlet-class>groovy.servlet.GroovyServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>GroovyTemplate</servlet-name>
        <servlet-class>groovy.servlet.TemplateServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>GroovyServlet</servlet-name>
        <url-pattern>*.groovy</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>GroovyTemplate</servlet-name>
        <url-pattern>*.gsp</url-pattern>
    </servlet-mapping>
</web-app>
```

Compile time nincs szükség a weblayer esetén a **groovy.jar**-ra, *runtime* azonben a container-nek biztosítania kell.

Java Server Pages

list.jsp

```
<c:choose>
    <c:when test="${requestScope.inventories.isEmpty()}">
        <span>Inventory list is <strong>empty</strong>.</span>
    </c:when>
    <c:otherwise>
        <table class="inventorytable">
            <thead>
                <tr>
                    <th>Reference</th>
                    <th>Name</th>
                    <th>Type</th>
                    <th>Value</th>
                </tr>
            </thead>
            <tbody>
                <c:forEach items="${requestScope.inventories}" var="item">
                    <tr>
                        <td><c:out value="${item.reference}" /></td>
                        <td><c:out value="${item.name}" /></td>
                        <td><c:out value="${item.type}" /></td>
                        <td><c:out value="${item.value}" /></td>
                    </tr>
                </c:forEach>
            </tbody>
        </table>
    </c:otherwise>
</c:choose>
```

A JSP lap mögött van egy `JSPListController` Servlet, mely a request attribútumai közé `inventories` kulccsal felvesz egy `List<Inventory>` példányt, majd a kérést **forward**-olja a **list.jsp** felé.

A JSP lapból a jsp-api a háttérben Java servlet-et készít.

Groovy Server Pages

list.gsp

```
<% if ( request.getAttribute('inventories').isEmpty() ) { %>
     <span>Inventory list is <strong>empty</strong>.</span>
<% } else { %>
    <table class="inventorytable">
        <thead>
            <tr>
                <th>Reference</th>
                <th>Name</th>
                <th>Type</th>
                <th>Value</th>
            </tr>
        </thead>
        <tbody>
          <% request.getAttribute('inventories').each { %>
                <tr>
                    <td>${it.reference}</td>
                    <td>${it.name}</td>
                    <td>${it.type}</td>
                    <td>${it.value}</td>
                </tr>
            <% } %>
        </tbody>
    </table>
<% } %>
```

A GSP lap mögött van egy `GSPListController` Servlet, mely a request attribútumai közé `inventories` kulccsal felvesz egy `List<Inventory>` példányt, majd a kérést **forward**-olja a **list.gsp** felé.

A GSP lapból a Groovy a háttérben TemplateServlet-et készít, mely egy JSP template-hez hasonlít leginkább. Ebből lesz ezt követően Java Servlet.

Groovy Servlet

list.groovy

```
if ( request.getAttribute('inventories').isEmpty() ) {
     span('Inventory list is empty.')
} else {
     table(class: 'inventorytable') {
          thead {
               tr {
                    th('Reference')
                    th('Name')
                    th('Type')
                    th('Value')
               }
          }
          tbody {
               request.getAttribute('inventories').each { item ->
                    tr {
                         td(item.reference)
                         td(item.name)
                         td(item.type)
                         td(item.value)
                    }
               }
          }
     }
}
```

A Groovy script mögött van egy `GroovyListController` Servlet, mely a request attribútumai közé `inventories` kulccsal felvesz egy `List<Inventory>` példányt, majd a kérést **forward**-olja a **list.groovy** script felé.

Ez egy *hagyományos* **Groovy script**, ami itt megjelenik, az lesz az output. Vannak implicit változók, melyek elérhetőek (pl. request, response, context, session, headers, out, html, json). A **html** egy `MarkupBuilder` osztály példánya (lásd példa).

A Groovy Script a Groovy Servlet GET HTTP methodjának tartalma lesz.