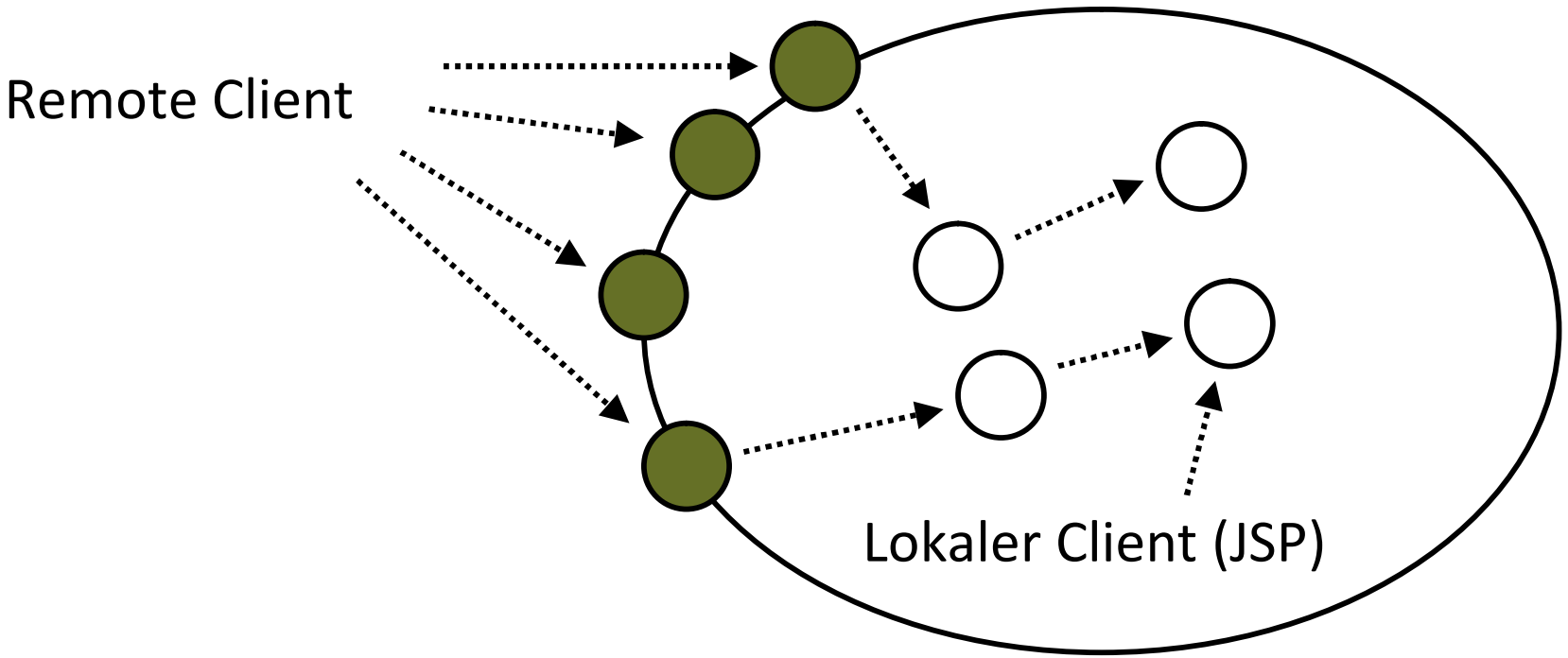
Lokale Beans

Der lokale und verteilte Fall



Remote Session/Entity-Bean

Lokale Session/Entity-Bean

Lokale Beans

- Die bisher vorgestellten EJBs waren immer in der Lage auf einem anderen Rechner abzulaufen.
  - Die Kommunikation läuft über RMI, vielleicht auch über CORBAs IIOP.
- Der Nachteil ist: RMI ist für EJBs, die sich innerhalb eines Servers unterhalten, unnützer Overhead.
  - Sieht man sich Web-Applikationen an, in dem ein JSP/Servlet eine EJB-Methode in dem gleichen Java EE-Container aufruft, ist die remote-Fähigkeit unnötig.
- Der EJB-Standard sieht daher seit EJB 2.0 **lokale EJB**s vor.
  - Lokale Beans können Entity-Beans oder Session-Beans sein.
- Ein Bean kann aber beides sein, was eine tolle Flexibilität ergibt.

Lokale Beans in XDoclet

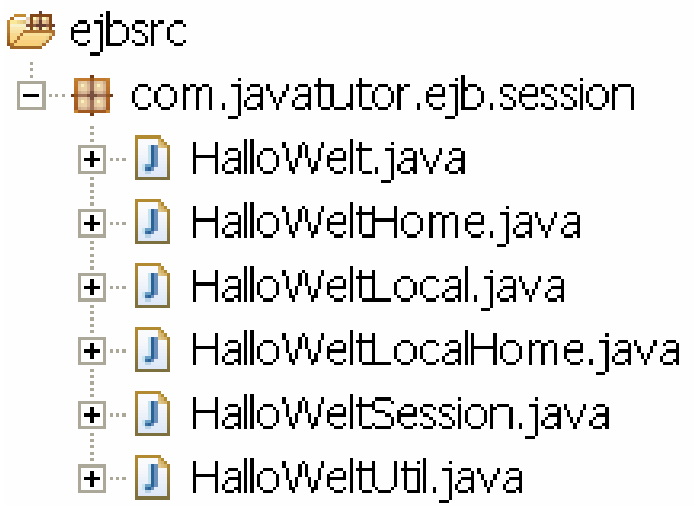
- Interessanterweise erstellt XDoclet für alle unsere Session Beans und Entiy-Beans automatisch Schnittstellen für den entfernten und lokalen Zugriff.
- Der Grund liegt darin, wie die Bean definiert wird:

```
/**
 * @ejb.bean name="HalloWelt"
 *           jndi-name="HalloWelt"
 *           type="Stateless"
 */
```

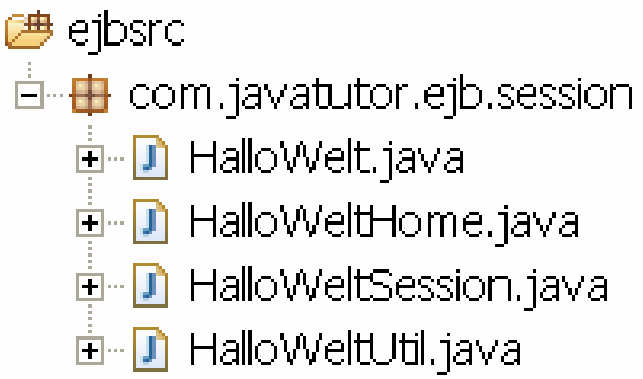
- Hier ist nicht ausdrücklich view-type="remote" gesetzt, so dass XDoclet für EJB 2.0 die Standardeinstellung »both« hat, also lokale und entferne Schnittstellen erzeugt.

Generierte Dateien bei view-type

```
/**
 * @ejb.bean
 *  name="HalloWelt"
 *  jndi-name="HalloWelt"
 *  type="Stateless"
 */
```




```
/**
 * @ejb.bean
 *  name="HalloWelt"
 *  jndi-name="HalloWelt"
 *  type="Stateless"
 *  view-type="remote"
 */
```



Entfernte und lokale Methoden

- Damit eine Methode lokal/remote oder beides ist, muss eine Angabe in XDoclet gemacht werden.
- Mit der folgenden Angabe ist hallo() remote und lokal.

```
/**
 * @ejb.interface-method
 */
public String hallo( String name )
{
  return "Hallo " + name + "!";
}
```

- Ist die Methode remote, steht an der Methode:

```
view-type="remote"
```


EJBLocalObject und EJBLocalHome

- Die Schnittstelle für den Client sind nun nicht mehr
  - javax.ejb.EJBObject (extends java.rmi.Remote) und
  - javax.ejb.EJBHome (extends java.rmi.Remote)

  da dies Java RMI Schnittstellen sind.
- Sie werden nun für lokale Beans ergänzt zu
  - javax.ejb.EJBLocalObject und
  - javax.ejb.EJBLocalHome.

Ein Blick auf die Klassen (1)

Generiertes remote Component Interface:

```
public interface HalloWelt extends EJBObject
{
  public String hallo( String name )
                    throws java.rmi.RemoteException;
}
```

Generiertes lokales Component Interface:

```
public interface HalloWeltLocal extends EJBLocalObject
{
  public String hallo( String name );
}
```

Ein Blick auf die Klassen (2)

Generiertes remote Home Interface:

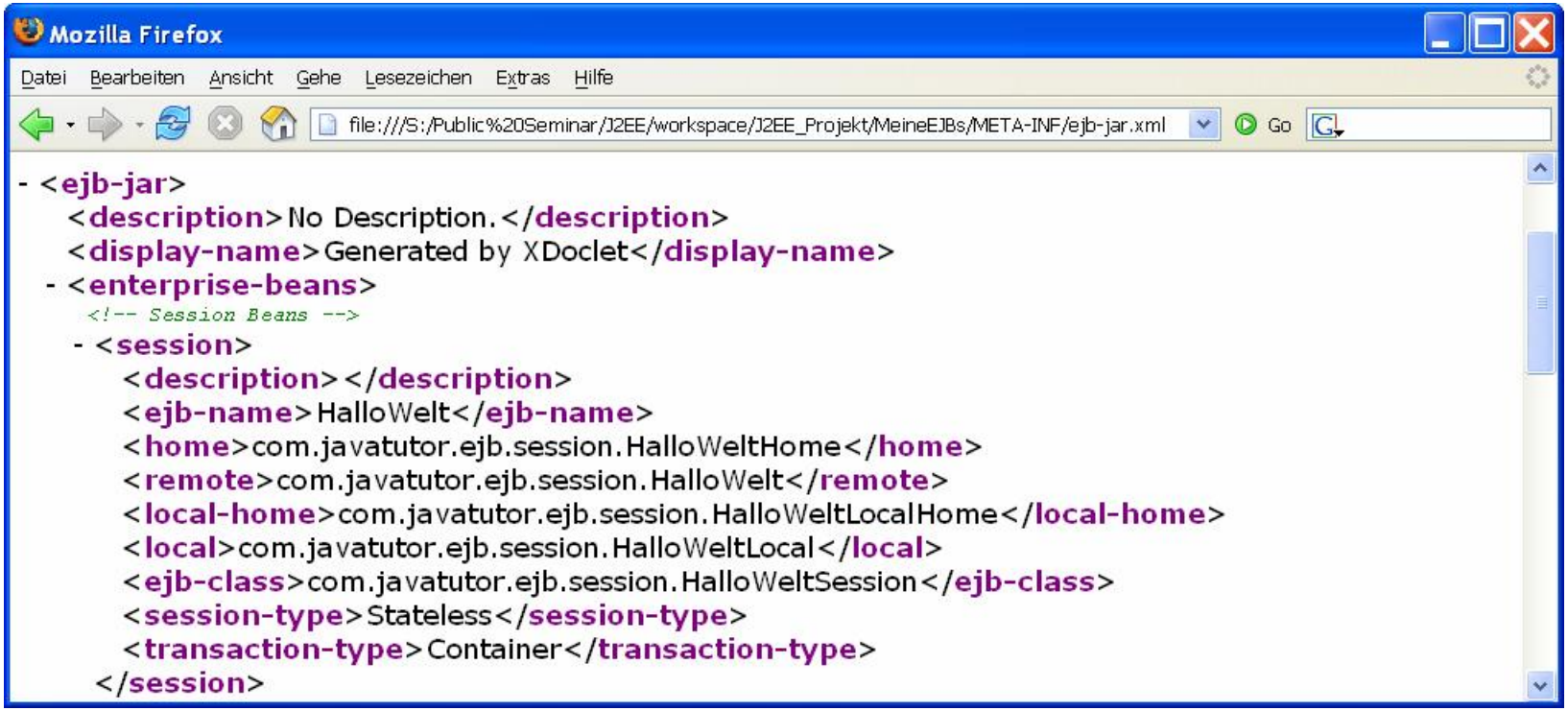
```
public interface HalloWeltHome extends EJBHome
{
  public HalloWelt create() throws
                        CreateException, RemoteException;
}
```

Generiertes lokales Home Interface

```
public interface HalloWeltLocalHome extends EJBLocalHome
{
  public HalloWeltLocal create() throws CreateException;
}
```

Angaben im Deployment-Deskriptor

- Ob eine Bean lokal ist oder nicht, bestimmen die Elemente <local-home> und <local> im Deskriptor.



- **jndi-name** und **local-jndi-name** müssen unterschiedlich sein!

Zusammenfassung von XDoclet

- XDoclet ermöglich an zwei Stellen die Angaben eines „view-type“:
  - Genau einmal an der Klasse über das Tag @ejb.bean und
  - an jeder Methode über @ejb.interface-method.
- Ist da Klassen-Tag @ejb.bean genutzt, so gilt die Einstellung automatisch für alle Methoden der EJB.
  - @ejb.interface-method kann diese Einstellung überschreiben.
  - Standardmäßig nimmt das dann (unsichtbare) @ejb.interface-method die Belegung aus @ejb.bean an.
- Während es für @ejb.interface-method keine Standard-belegung gibt (es wird @ejb.bean übernommen), ist der View-Typ von @ejb.bean ab EJB 2.0 auf „both“ vorbelegt.

Praktisches Beispiel

JNDI-Namen für lokalen Fall

- XDoclet vergibt für den Standard **view-type** (also **both**) automatische einen lokalen Namen, der auf **Local** endet.
- Um diesen von Hand zu setzen, nutzt man **local-jndi-name**:

```
/**
 * @ejb.bean name="HalloWelt"
 *           jndi-name="HalloWelt"
 *           local-jndi-name="HalloWeltLocal"
 *           type="Stateless"
 */
public class HalloWeltBean implements SessionBean
 ...
```

Programmcode für den Zugriff

- Eine EJB kann nun auf eine lokale Bean direkt ohne remote-Aufrufe zugreifen (aber immer noch über das lästige JNDI):

```
Context ctx = new InitialContext();
HalloWeltLocalHome halloHome = (HalloWeltLocalHome)
   ctx.lookup( HalloWeltLocalHome.JNDI_NAME );
HalloWeltLocal hallo = halloHome.create();
return "'" + hallo.hallo(name) + "'";
```

- XDoclet erzeugt eine Util-Klasse mit einer praktischen Funktion, die gleich das LocalHome liefert:

```
HalloWeltLocalHome halloHome =
                        HalloWeltUtil.getLocalHome();
HalloWeltLocal hallo = halloHome.create();
```


Unterschiede in remote und lokal

Ortstransparenz

- Bei remote-EJBs kann der Client Funktionen aufrufen, die nicht zwingend auf seinem Rechner liegen.
  - Das nennt man **Ortstransparenz**.
  - Der Client kann Funktionen aufrufen, die entweder auf seinem Rechner liegen oder irgendwo anders.
- Eine lokale Schnittstelle muss auf dem Rechner platziert sein, wo auch die Bean auf sie zugreift.
  - Das grenzt die Flexibilität ein.
  - Das ist aber viel schneller!

Semantik bei Übergabe/Rückgabe

- Bei einem Remote-Interface werden alle Parameter oder Rückgabewerte als Wertekopie übergeben.
  - Bei Referenzen heißt das, das ein serialisiertes Objekt übergeben wird (falls der Parameter nicht instanceof Remote war).
  - Das konnte natürlich beliebige groß werden, man denke an einen serialisieren XML-Baum.
- Bei lokalen Aufrufen wird keine Kopie mehr übergeben, sondern ein Verweis auf das direkte Objekt, weil die Objekte in einem Adressraum liegen.
  - Änderungen werden plötzlich möglich und können zu einem Sicherheitsproblem werden.
  - Doch große Objekte können schnell weitergeben werden.

Keine RemoteException

- Die Methoden, die eine lokale Schnittstelle definieren, dürfen nun nicht mehr RemoteException auslösen.
  - Es gibt ja keine Transportschwierigkeiten mehr.
- Allerdings gab es einige Ausnahmen, die von java.rmi.RemoteException abgeleitet waren.
  - Diese sind jetzt auch nicht mehr erlaubt, da sich ja instanceof RemoteException sind.

java.rmi

Class RemoteException

java.lang.Object
  └ java.lang.Throwable
    └ java.lang.Exception
      └ java.io.IOException
        └ **java.rmi.RemoteException**

All Implemented Interfaces:
Serializable

Direct Known Subclasses:
AccessException, ActivateFailedException, ActivityCompletedException, ActivityRequiredException, ConnectException, ConnectIOException, ExportException, InvalidActivityException, InvalidTransactionException, MarshalException, NoSuchObjectException, ServerError, ServerException, ServerRuntimeException, SkeletonMismatchException, SkeletonNotFoundException, StubNotFoundException, TransactionRequiredException, TransactionRolledbackException, UnexpectedException, UnknownHostException, UnmarshalException

Umgewandelte Exceptions

- javax.transaction.TransactionRequiredException
  - javax.ejb.TransactionRequiredLocalException
- javax.ejb.TransactionRolledBackException
  - javax.ejb.TransactionRolledbackLocalException
- java.rmi.NoSuchObjectException
  - javax.ejb.NoSuchObjectLocalException

Methodenvergleich

- Während EJBHome
  - Metadaten bereitstellen kann (getEJBMetaData()),
  - Handels kennt (getHomeHandle()) und
  - zwei Methoden zum Löschen vorschreibt (remove(Handle handle), remove(Object primaryKey))

  hat man bei EJBLocalHome nur eine Methode:
  - void remove(Object primaryKey)
- remove() entfernt das lokale EJB-Objekt, sofern es sich um eine lokale Entity-Bean handelt.
- Die »fehlenden« Methoden aus EJBHome werden lokal nicht benötigt.
- Ein EJBLocalObject deklariert keine getHandle()-Methode, da ein serialisierbarer Handle lokal nicht nötig/möglich ist.