ИСПОЛНЯТОР

Итоговый проект в УНЦ «Инфоком»

Дмитрий Солдатов, Алексей Скобеев

2011

Часть 1
Концепция

Что это такое?

ИСПОЛНЯТОР – это система распределения задач. Предназаначена для решения организационных вопросов.

Все пользователи условно разделены на **менеджеров** и **исполнителей**. Первые – ставят задачи, вторые – исполняют.

В результате, менеджерам проще контролировать работу подчинённых, а подчинённые имеют возможность обсуждать задачи и уточнять подробности.

Так что же это такое?

- **Веб-приложение** – клиент получает доступ через веб-браузер
- **Java Enterprise приложение** – работает на специальном сервере (сервере приложений)
- Для хранения данных используется **база данных**
- Возможен экспорт данных в **XML**-файл и последующий их импорт

Как это должно работать?

- Менеджер добавляет задачу, назначает исполнителей
- В комментариях идёт обсуждение, менеджер может корректировать сроки, описание задачи
- Исполнитель приступает к выполнению
- Исполнитель сообщает о завершении задачи
- Менеджер присваивает задаче соответствующий статус

Сущности предметной области

- **Пользователь**
  имя, пароль, e-mail, роль в системе

- **Задача**

  название, описание, исполнители, сроки выполнения, статус, приоритет, комментарии, история изменений

Всего – 8 таблиц

Роли – разделение прав

- **Администратор**
  управляет аккаунтами, правами и другой информацией в системе
- **Менеджер**
  создаёт задачи и контролирует их выполнение
- **Исполнитель**
  получает задачи и общается с менеджером
- **Гость**
  человек, не имеющий аккаунта в системе

Статус – текущее состояние задачи

- **Не начато**
- **Запланировано**
  менеджер зарезервировал задачу для определённого исполнителя; это сигнал для других менеджеров
- **На выполнении**
- **Приостановлено**
- **Выполнено**
- **Отменено**

Приоритет – степень важности на данный момент

С течением времени приоритет может меняться

- Низкий
- Нормальный
- Высокий
- Критический

Экспорт и импорт

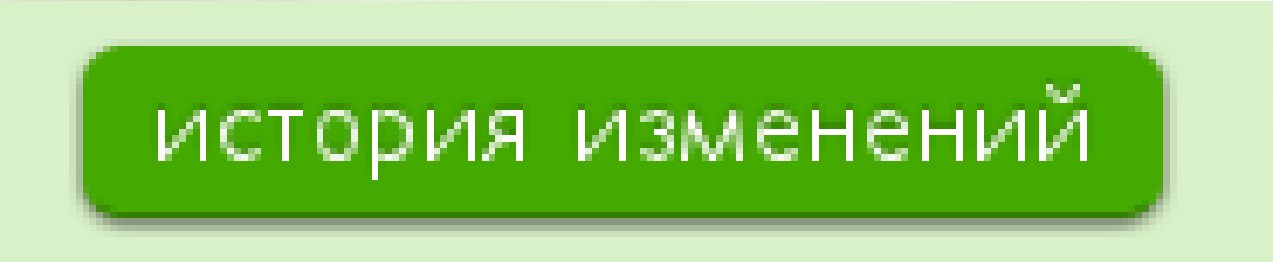
Функция экспорта связанных данных сохраняет объекты, которые связаны с выбранными задачами.

Если в файле есть связанные данные, при импорте по ним можно проверить корректность данных.

История изменений

Все изменения, вносимые в задачи после создания, фиксируются в истории.

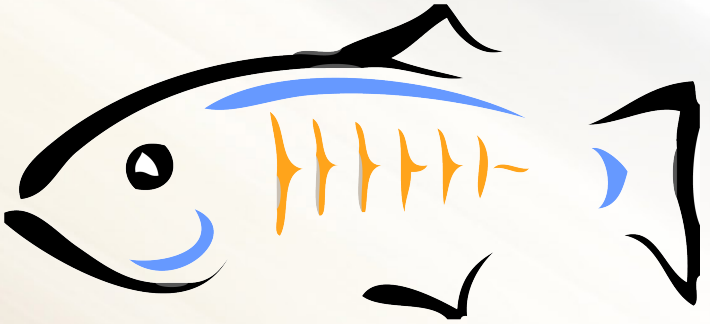
Для каждой задачи можно посмотреть хронологию изменений

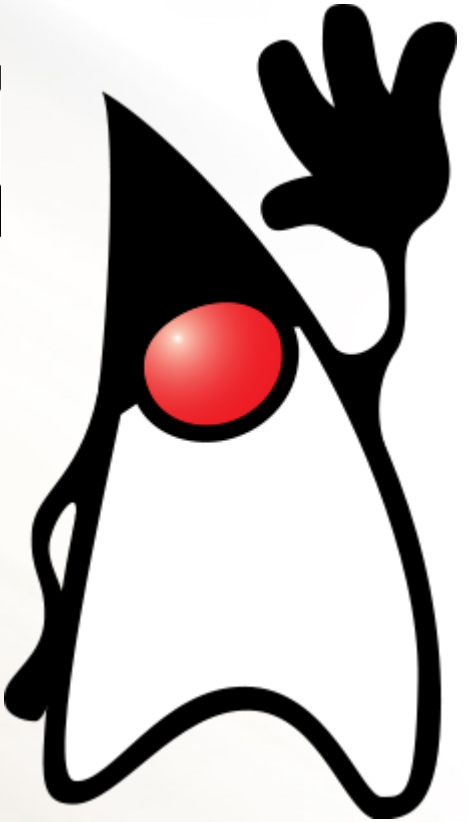


Часть 2
Технические подробности

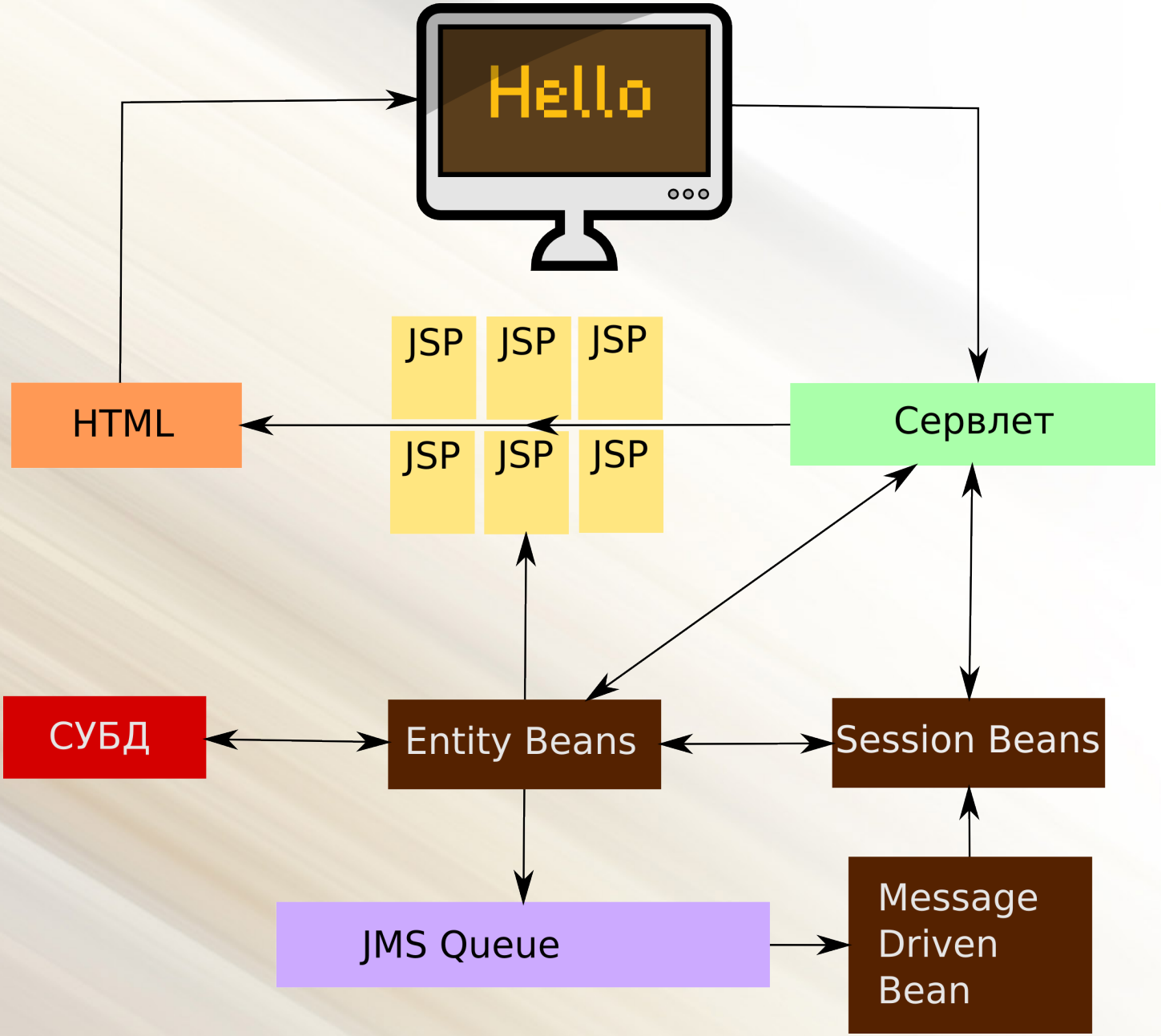
Платформа

- J2EE 1.4
- Oracle XE 10g
- GlassFish 3.1
- NetBeans 7.0
- HTML 5
- CSS 3





Как всё устроено?



Работа с базой данных

- Автоматическая генерация ID: sequence + триггер
- Работа через Connection Pool и связанный с ним DataSource
- Использование bind variables и классов `OracleConnection`, `OraclePreparedStatement`

Пример работы с БД

```
DataSource ds = (DataSource) new
InitialContext().lookup("jdbc/IspolnyatorDataSource");

OracleConnection conn = (OracleConnection)
ds.getConnection().unwrap(oracle.jdbc.OracleConnection.class);

OraclePreparedStatement statement = (OraclePreparedStatement)
conn.prepareStatement("SELECT comment_id FROM comments WHERE comment_id
= :commentid");

statement.setLongAtName("commentid", Long.parseLong(aKey.toString()));

ResultSet rs = statement.executeQuery();

if (!rs.next()) {

    throw new ObjectNotFoundException();

}

statement.close();

conn.close();
```

Entity Beans

- Предоставляют доступ к данным из БД
- Синхронизация происходит по усмотрению сервера приложений
- Всегда имеют возможность поиска по первичному ключу
- Возможен поиск по другим критериям

Home-интерфейс entity компонента

```
public interface ExecutorLocalHome extends EJBLocalHome {

    public entityBeans.ExecutorLocal
findByPrimaryKey(java.lang.Long key) throws FinderException,
EJBException;

    public java.util.Collection findByTaskId(java.lang.Long key)
throws FinderException, EJBException;

    public java.util.Collection findByUserId(java.lang.Long key)
throws FinderException, EJBException;

    public entityBeans.ExecutorLocal
findByUserIdAndTaskId(java.lang.Long uid, java.lang.Long tid)
throws FinderException, EJBException;

    public entityBeans.ExecutorLocal create(java.lang.Long
userIdArg, java.lang.Long taskIdArg) throws CreateException,
EJBException;

}
```

Session Beans

- Выполняют импорт/экспорт и сохранение истории изменений
- Являются компонентами без состояния
- Используют Entity-компоненты для работы с БД

```
public interface TransportBeanLocal extends EJBLocalObject {

   public String exportToXML(ArrayList tasks, boolean
needLinkedData);

   public void importFromXML(String filename, boolean
checkLinkedData);

  }
```

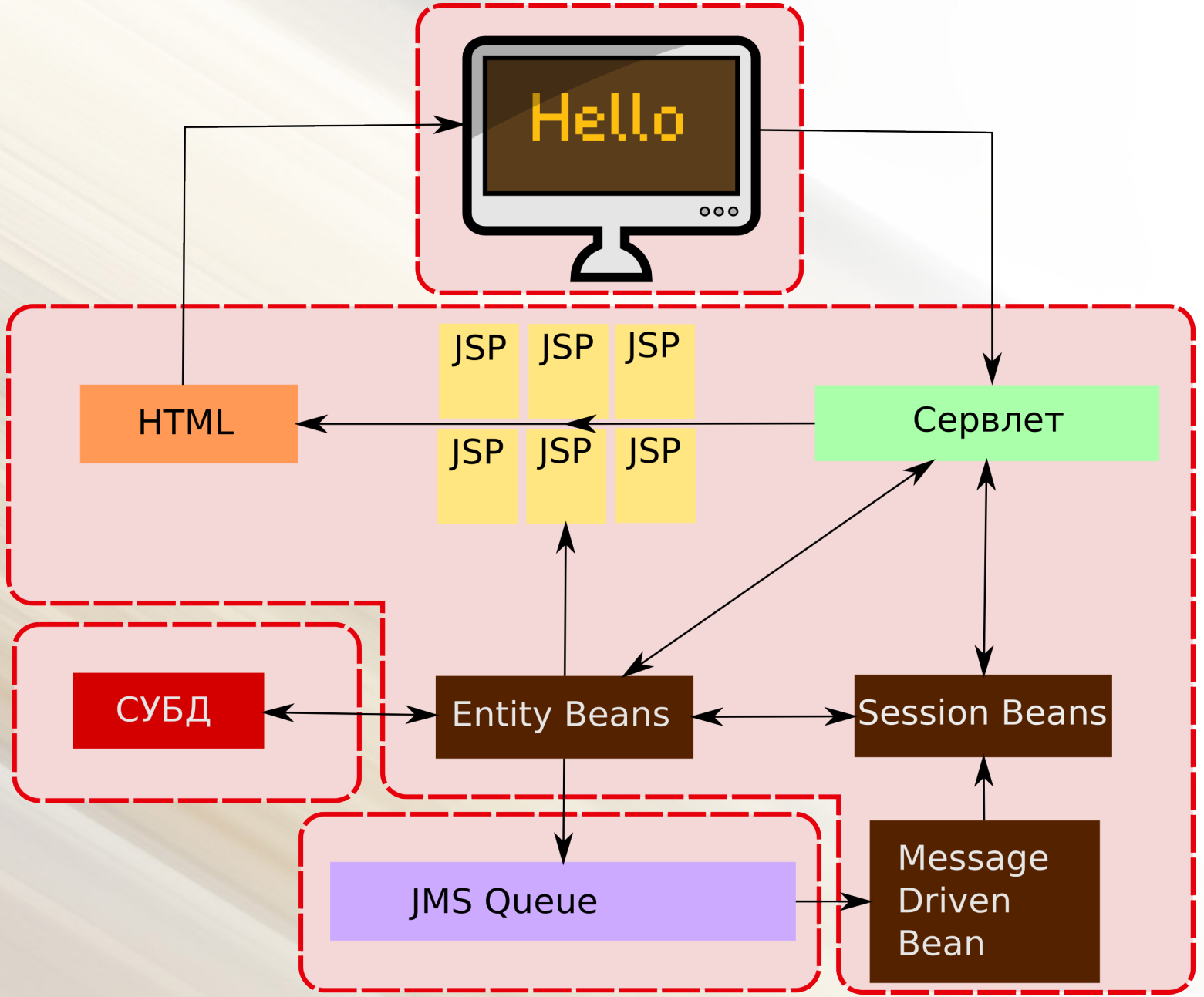
Message Driven Beans

- Получают сообщения JMS от Entity-компонентов
- Используют сессионные компоненты для обработки сообщений
- Могут вместе со службой JMS находиться на отдельном сервере

Обработка сообщения

```
public void onMessage(Message aMessage) throws
EJBException {
        ObjectMessage m = (ObjectMessage)
aMessage;
        HistoryMessagemsgobj = (HistoryMessage)
m.getObject();
        HistoryBeanLocal hb = null;
        historyBean = historyHome.create();
        historyBean.addRecord(msgobj);
    }
```

Возможное распределение по компьютерам



Сервлет

- Принимает запросы, адресованные приложению
- Выполняет необходимую подготовку
- Передаёт управление(forwarding) нужному JSP
- Если обработанный URL не требует отдельной страницы, выполняется перенаправление (redirecting) на один из существующих URL приложения

Сервлет (продолжение)

- Реализует разграничение прав пользователей
- Выполняет подготовительные действия (сохранение объектов в сессии, определение полного адреса приложения ...)
- Обрабатывает возможные ошибки

```
private void handleException(HttpServletRequest request,
HttpServletResponse response, Exception ex){

        request.getSession(true).setAttribute("error", ex);

        RequestDispatcher dispatcher =
request.getRequestDispatcher("/error.jsp");

        dispatcher.forward(request, response);

    }
```

JSP страницы

- Заготовки страниц, которые сервлет заполняет данными и возвращает пользователю
- Общие части выделены в отдельные JSP и подключаются тэгом **<jsp:include>**
- Используют рекомендации **HTML 5**
- Для удобства пользователя использованы процедуры **JavaScript**
- От наличия **прав доступа** зависит наличие/доступность некоторых ссылок

Что же получается?

- Entity-компоненты представляют данные (Model)
- JSP страницы — способ отображения (View)
- Сервлет реализует большую часть бизнес-логики и даёт возможность работать с данными с помощью имеющихся средств отображения (Controller)

Да это же MVC!

Простор для совершенствования

- Оповещения (e-mail, jabber, Skype, …)
- Возможность редактирования наборов ролей, статусов, приоритетов
- Отчёты, версии для печати
- Авторизация через LDAP
- Механизм взаимодействия пользователей (назначение исполнителя, сдача задания)
- Улучшения интерфейса (AJAX)

Спасибо за внимание!