# Лабораторная работа 1 - RMI

Обратимся к определению:

RMI-технология позволяет выполнять прозрачный запуск методов объектов Java, которые расположены на удаленных машинах, таким образом, как если бы они располагались на локальных машинах. RMI – позволяет строить приложения, работа которых распределена между несколькими машинами в архитектуре клиент-сервер.

Короче говоря, RMI есть технология позволяющая выносить работу на плечи сервера, разгружая при этом клиента. Реализуется следующим образом:

1. Создается интерфейс, который включает в себя набор описаний методов, которые можно вызывать удаленно.
2. На стороне сервера создается класс и впоследствии объект этого класса, реализующего интерфейс, описанный выше. Объект класса регистрируется под псевдонимом на RMI-сервере, который является промежуточным звеном между клиентом и сервером. На RMI-сервере хранятся данные о зарегистрированных объектах с методами для удаленного вызова.
3. На клиенте с RMI-сервера забирается «болванка» объекта с методами для удаленного вызова. Ссылка этой «болванки» является ссылкой интерфейса, описанной в первом пункте. Таким образом клиент имеет данные о методах, которые можно вызвать. Это болванка не есть копия объекта на сервере, она скрывает в себе вызовы на сервере методов соответствующего объекта, хотя с виду может показаться, что идет работа с обычным объектом.
4. Используя болванку делаем вызов удаленных методов и получаем результат – PROFIT!

Попробуйте! Это легко. (ну, когда посидишь над этим)

# Лабораторная работа 2 – Запуск TomCat

Tomcat – контейнер сервлетов с открытым исходным кодом, реализубщий спецификацию сервлетов и спецификацию JavaServer Pages (JSP) и JavaServer Faces (JSF), написанный на языке Java.

Томкат позволяет запускать веб-приложения, являясь для них контейнером.

Работает Томкат примерно так. На сервер поступает запрос от клиента. Его обрабатывает Томкат, определяя какому приложению предназначался запрос, и специальным образом обработав его, отправляет далее по пункту назначения. Соответственно, веб приложение, обработав запрос, формирует ответ, и ответ также проходит через руки Томката, который, проделав некоторые манипуляции, отправляет результат клиенту.

Приложение в лабораторной содержит две страницы JSP. Первая содержит кнопку, которая перенаправляет нас на вторую страницу, где отображается время, прошедшее с начала работ приложения.

Это все чудо работает благодаря Томкату и технологии JSP. На первой странице создается бин типа «Григорианский календарь» и по нажатию кнопки передается запрос на сервер вместе с временем в миллисекундах. Это счастье отправляется на сервлет, в методе doPost() из объекта реквеста достается параметр, содержащий вермя в милисекундах, обрабатывается, делается расчет прошедшего времени, в реквест кладется атрибут с нужными данными, и реквест перенаправляется на страницу, которая отобразит результат. И весь зал рукоплещет!

# Лабораторная работа 3 – JPA, паттерн «Команда»

Дело в том, что мы уже сдали курсовой, где все это применяли, что [как бы намекает](http://lurkmore.to/%D0%9A%D0%B0%D0%BA_%D0%B1%D1%8B_%D0%B3%D0%BE%D0%B2%D0%BE%D1%80%D0%B8%D1%82_%D0%BD%D0%B0%D0%BC). Но не будет лишним рассказать про это все.

Java Persistence AP предоставляет стандартный механизм для ORM, EntityManager API для создания, обновления и удаления объектов и язык запросов JPA-QL или JPQL для извлечения сущностей, запросы которого могут быть проверены на корректность на этапе компиляции и могут быть динамически сформированы на этапе выполнения приложения.

Слизано у Блинова, но он неверной информации не скажет. В общем, данная технология помогает соотносить классы сущностей с соответствующими таблицами в базе данных, а также предоставляет API для манипулирования сущностями. API позволяет сохранять, восстанавливать, обновлять или удалять объекты, освобождая программиста от написания JDBC и SQL кода. Также предоставляет язык запросов для восстановления объектов из базы данных. Это один из наиболее важных элементов, так как некорректные SQL утверждения могут замедлить базу данных. Этот подход также освобождает приложение от внутренних SQL, разбросанных по всему приложению. Тоже слизано, но из песни слов не выкинешь.

Короче говоря, крутая штука, облегчающая работу программиста, - при минимуме усилий создается управление над базой данных с совершенно любой структурой связей между сущностями базы.

Далее, в лабораторной используется паттерн команда. Его смысл в разделении выполнении определенной последовательности команд и непосредственного содержания данных команд.

Стоит пояснить. Мы имеем определенный контракт – интерфейс с описанием некоторых методов. Имеем код, использующий ссылки на данный интерфейс и работающий в рамках своей философии. Под ссылкой на интерфейс может скрываться абсолютно что угодно. «Что угодно», реализующее интерфейс, оговоренный ранее.

То есть если мы имеем несколько задач для выполнения, то можно подставить соответствующую реализацию интерфейса в код и все. Кроме того, паттерн помогает избежать использования команды типа «switсh-case» при выборе команды, упрощая тем самым код программы. Также программа становится легко расширяемой, ибо для добавления новой команды не требуется переписывать и/или редактировать старый код, а просто создать новый класс, реализующий интерфейс, и зарегистрировать этот класс каким-то образом.

Кульная штука, когда попробуешь.

В примерах отдельно рассматриваются JPA и паттерн. С ДЖПА выполняются обычные действия, типа создать сущность, обновить сущность, удалить сущность. Также приведен пример реализации, где используется паттерн «Команда». От клиента посылаются определенные команды, которые распознаются на сервере. Далее создаются соответствующие экземпляры класса команды и выполняются его методы (в примере всего один метод на команде).

# Лабораторная работа 4 – MDB

В вопросах было «принцип асинхронной обработки запроса».

Начиная с версии 7.0 в Tomcat в полной мере реализована поддержка Servlet 3.0, а вместе с ней поддержка асинхронных сервлетов.

Если вкратце, то суть ее в том, что клиент открывает долгоживущее HTTP-соединение, которое хранится на сервере до того момента, пока сервер не будет готов отослать ответ обратно. Посылка ответа инициируется сервером. Такой подход позволяет избавиться от многократного опроса сервера множеством клиентов с целью получить как можно более свежую информацию. Например, такой сценарий имеет место в онлайн аукционах, разного рода службах информирования об изменении курса акций, чатах. Суть в том, что момент обновления информации неизвестен, но все хотят получить ее как можно быстрее после опубликования, поэтому начинают делать запросы на сервер с большой частотой в надежде не пропустить обновление информации на сервисе. Короче, такая вполне себе DoS-атака. При использовании асинхронных сервлетов соединение открывается один раз, и потом сервер сам отошлет ответ, когда будет что отсылать. Естественно, когда с сервера придет ответ, нужно установить соединение заново.

В пример мы использовали JMS, он как раз работает асинхронно. Интересный факт, но говорят, что он тяжеловесен и неудобен в использовании.

При добавлении сообщения в базу данных используется JMS. Так как он работает асинхронно, то порой возникает ситуация, что при добавлении сообщения и переходу к списку всех сообщений, новое в списке не появляется.

# Лабораторная работа 5 - EJB + JSF

Как сконфигурирован JPA?

Основной конфигурационной единицей JPA является Persistence Unit – в нем описывается тип провайдера, который предоставляет “услуги” JPA, а также список классов, за которые он “отвечает” – JPA-сущностей. В блоке <properties> описываются параметры доступа к базе данных, её тип, настройки логирования и тд.

После выхода спецификации WebServlet 3.0 часть описания может быть расположена непосредственно в классах в виде атрибутов методов, классов, полей.

Чем отличается JSF от JSP?

JavaServer Faces(JSF)— это фреймворк для веб-приложений на Java. Он служит для облегчения разработки пользовательских интерфейсов для Java EE приложений. Подход JSF основывается на использовании компонентов. Состояние компонентов пользовательского интерфейса сохраняется, когда пользователь запрашивает новую страницу и затем восстанавливается, если запрос повторяется. MVC: Model-entity или др классы; View – jsf страница в xhtml формате; Сontroller–managed bean, который интерпретирует данные, введённые пользователем, и информирует модель о необходимости соответствующей реакции.

Технология Java Server Pages (JSP) создана, чтобы облегчить создание

страниц с динамическим содержанием. В то время как сервлеты наилучшим образом подходят для выполнения контролирующей функции приложения в виде обработки запросов и определения содержания и вида ответа, страницы JSP выполняют функцию отображения

результатов работы приложения в виде текстовых документов типа HTML, XML, WML и некоторых других. JSP поддерживают как JavaScript, так и HTML-теги. JavaScript обычно используется, чтобы добавить функциональные возможности на уровне HTML-страницы.

Что такое dependency injection и inversion of control?

Inversion of Control (инверсия управления) — это некий абстрактный принцип, набор рекомендаций для написания слабо связанного кода. Суть которого в том, что каждый компонент системы должен быть как можно более изолированным от других, не полагаясь в своей работе на детали конкретной реализации других компонентов.

Dependency Injection (внедрение зависимостей) — это одна из реализаций этого принципа (помимо этого есть еще Factory Method, Service Locator).

В чем отличие stateless от statfull beans?

Stateful может хранить какое-либо состояние между запросами, соответственно контейнер имеет пул таких бинов и для каждой сессии выделяет отдельный экземпляр. Stateless никакого состояния не хранит, а следовательно может использоваться один и тот же экземпляр