МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«Гомельский государственный технический университет имени П.О. Сухого»

Кафедра «Информационные технологии»

Курс лекций
старшего преподавателя Стефановского И.Л.
по дисциплине

# «СОВРЕМЕННЫЕ ТЕХНОЛОГИИ СЕРВЕРНОЙ РАЗРАБОТКИ»

для студентов специальности
6-05-0611-01 "Информационные системы и технологии"
дневной и заочной форм обучения

Гомель 2024

**Содержание**

# РАЗДЕЛ 1 ТЕХНОЛОГИИ СЕРВЕРНОЙ РАЗРАБОТКИ

## Лекция 1 Технологии серверной разработки информационных систем

**Информационная система (ИС)** – это вся инфраструктура предприятия, задействованная в процессе управления всеми информационнодокументальными потоками, включающая в себя следующие обязательные элементы:

- Информационная модель, представляющая собой совокупность правил и алгоритмов функционирования ИС. Информационная модель включает в себя все формы документов, структуру справочников и данных, и т.д.
- Регламент развития информационной модели и правила внесения в неё изменений.
- Кадровые ресурсы (департамент развития, привлекаемые консультанты), отвечающие за формирование и развитие информационной модели.
- Программное обеспечение, конфигурация которого соответствует требованиям информационной модели (программное обеспечение является основным движителем и, одновременно, механизмом управления ИС). Кроме того, всегда существуют требования к поставщику программного обеспечения, регламентирующие процедуру технической и пользовательской поддержки на протяжении всего жизненного цикла.
- Кадровые ресурсы, отвечающие за настройку и адаптацию программного обеспечения, и его соответствие утвержденной информационной модели.
- Регламент внесения изменений в настраиваемые структуры (специфические настройки, структуры баз данных и т.д.) и конфигурацию программного обеспечения и состав его функциональных модулей.
- Аппаратно-техническая база, соответствующая требованиям по эксплуатации программного обеспечения (компьютеры на рабочих местах, периферия, каналы телекоммуникаций, системное программного обеспечение и СУБД).
- Эксплуатационно-технические кадровые ресурсы, включая персонал по обслуживанию аппаратно-технической базы.
- Правила использования программного обеспечения и пользовательские инструкции, регламент обучения и сертификацию пользователей.

**Ресурсы корпораций включают***:*

1. материальные (материалы, готовая продукция, основные средства);
2. финансовые;
3. людские (персонал); 4. знания (ноу-хау);
5. КИС.

**Система управления любой компании включает** *три основные подсистемы:*

1. Планирование продаж и операций. Это общий план функционирования предприятия, устанавливающий объемы изготовления готовой продукции. Главным здесь является планирование спроса и оценка ресурсов, необходимых для удовлетворения спроса. Здесь же создается основной производственный план, определяющий, какие изделия, в каком количестве и в какие сроки нужно произвести.
2. Детальное планирование необходимых ресурсов (материалов, производственных мощностей, трудовых ресурсов и т.д.). Составленный план определяет время и объем заказов для всех материалов и комплектующих, необходимых для реализации основного производственного плана.
3. Управление исполнением планов в процессе производства и закупок (снабжения).

Все эти подсистемы реализуются на основе КИС.

**Корпоративные информационные системы (КИС)** - это интегрированные системы управления территориально распределенной корпорацией, основанные на углубленном анализе данных, широком использовании систем информационной поддержки принятия решений, электронных документообороте и делопроизводстве. КИС призваны объединить стратегию управления предприятием и передовые информационные технологии.

**Корпоративная информационная система** — это совокупность технических и программных средств предприятия, реализующих идеи и методы автоматизации.

Комплексная автоматизация бизнес процессов предприятия на базе современной аппаратной и программной поддержки может называться поразному. В настоящее время наряду с названием Корпоративные информационные системы (КИС) употребляются, например, следующие названия:

1. Автоматизированные системы управления (АСУ);
2. Интегрированные системы управления (ИСУ);
3. Интегрированные информационные системы (ИИС); 4. Информационные системы управления предприятием (ИСУП).

**Главная задача КИС** - эффективное управление всеми ресурсами предприятия (материально- техническими, финансовыми, технологическими и интеллектуальными) для получения максимальной прибыли и

удовлетворения материальных и профессиональных потребностей всех сотрудников предприятия.

**КИС по своему составу** - это совокупность различных программноаппаратных платформ, универсальных и специализированных приложений различных разработчиков, интегрированных в единую информационнооднородную систему, которая наилучшим образом решает в некотором роде уникальную задачу каждого конкретного предприятия. То есть, КИС - человеко-машинная система и инструмент поддержки интеллектуальной деятельности человека, которая под его воздействием должна:

- Накапливать определенный опыт и формализованные знания;
- Постоянно совершенствоваться и развиваться;
- Быстро адаптироваться к изменяющимся условиям внешней среды и новым потребностям предприятия.

Комплексная автоматизация предприятия подразумевает перевод в плоскость компьютерных технологий всех основных деловых процессов организации. И использование специальных программных средств, обеспечивающих информационную поддержку бизнес-процессов, в качестве основы КИС представляется наиболее оправданным и эффективным. Современные системы управления деловыми процессами позволяют интегрировать вокруг себя различное программное обеспечение, формируя единую информационную систему. Тем самым решаются проблемы координации деятельности сотрудников и подразделений, обеспечения их необходимой информацией и контроля исполнительской дисциплины, а руководство получает своевременный доступ к достоверным данным о ходе производственного процесса и имеет средства для оперативного принятия и воплощения в жизнь своих решений. И, что самое главное, полученный автоматизированный комплекс представляет собой гибкую открытую структуру, которую можно перестраивать на лету и дополнять новыми модулями или внешним программным обеспечением.

Под корпоративной информационной системой будем понимать информационную систему организации, отвечающую следующему минимальному перечню требований:

1. Функциональная полнота системы;
2. Надежная система защиты информации;
3. Наличие инструментальных средств адаптации и сопровождения системы;
4. Реализация удаленного доступа и работы в распределенных сетях;
5. Обеспечение обмена данными между разработанными информационными системами и др. программными продуктами, функционирующими в организации;
6. Возможность консолидации информации;

7. Наличие специальных средств анализа состояния системы в процессе эксплуатации.

**Функциональная полнота системы**

− выполнение международных стандартов управленческого учета MRP II, ERP, CSRP;

− автоматизация в рамках системы решения задач планирования, бюджетирования, прогнозирования, оперативного (управленческого) учета, бухгалтерского учета, статистического учета и финансовогоэкономического анализа;

− формирование и ведение учета одновременно по российским и международным стандартам;

− количество однократно учитываемых параметров деятельности организации от 200 до 1000, количество формируемых таблиц баз данных – от 800 до 3000.

**Система защиты информации**

− парольная система разграничения доступа к данным и реализуемым функциям управления;

− многоуровневая система защиты данных (средства авторизации вводимой и корректируемой информации, регистрация времени ввода и модификации данных).

**Инструментальные средства адаптации и сопровождения системы**

− изменение структуры и функций бизнес-процессов;

− изменение информационного пространства;

− изменение интерфейсов ввода, просмотра и корректировки информации;

− изменение организационного и функционального наполнения рабочего места пользователя; − генератор произвольных отчетов;

− генератор сложных хозяйственных операций; − генератор стандартных форм.

**Возможность консолидации информации**

− на уровне организации – объединение информации филиалов, холдингов, дочерних компаний и т.д.;

− на уровне отдельных задач – планирования, учета, контроля и т.д.; − на уровне временных периодов – для выполнения анализа финансово-экономических показателей за период, превышающий отчетный.

**Специальные средства анализа состояния системы в процессе эксплуатации**

− анализ архитектуры баз данных;

− анализ алгоритмов;

− анализ статистики количества обработанной информации;

− журнал выполненных операций;

− список работающих станций серверов; −

анализ внутрисистемной почты.

Наиболее развитые корпоративные ИС (КИС) предназначены для автоматизации всех функций управления корпорацией: от научнотехнической и маркетинговой подготовки ее деятельности до реализации ее продукции и услуг. В настоящее время КИС имеют в основном экономическую и производственную направленность.

Общие вопросы проектирования и внедрения КИС

Успешное руководство бизнесом невозможно сегодня без постоянной, объективной и всесторонней информации. Для повышения эффективности и минимизации издержек управления (временных, ресурсных и финансовых), разрабатываются и применяются корпоративные информационные системы, помогающие осуществлять контроль бюджетных процессов, рабочего времени сотрудников, выполненных ими работ, хода реализации проектов, документооборота, и других управленческих функций. Доступ к подобного рода данным может быть осуществлён как в локальной сети, так и через Интернет. С помощью эффективной корпоративной информационной системы можно значительно упростить процессы контроля и управления на предприятии любого уровня. Разработка и реализация информационных систем – одно из основных направлений деятельности вашей специальности. Этот процесс начинается с анализа деятельности предприятия и заканчивается внедрением разработанной системы. **Все *этапы* этого процесса:**

1. Проведение предпроектного обследования
2. Формулирование целей и ограничений проекта, разработка стратегии реализации проекта
3. Инжиниринг и реинжиниринг бизнес-процессов Заказчика, консалтинг в различных областях
4. Выбор платформы, разработка системы, интеграция с используемым программным обеспечением
5. Поставка оборудования и программного обеспечения
6. Пусконаладочные работы по вводу системы в эксплуатацию
7. Сопровождение созданной системы в процессе эксплуатации, работы по ее дальнейшему развитию

Так же корпоративные информационные системы сегодня являются важнейшим инструментом внедрения новых методов управления и реструктуризации предприятия**.**

В последнее время интерес к корпоративным информационным системам (КИС) постоянно растет. Если вчера КИСы привлекали внимание

довольно узкого круга руководителей, то сейчас проблемы автоматизации деятельности компаний стали актуальными практически для всех. Обусловлено это не только положительной динамикой развития экономики, но и тем, что сегодня предприятия уже обладают значительным опытом использования программных продуктов различного класса.

Основная задача *проектирования и внедрения* корпоративных информационных систем, как результата системной интеграции, - комплексная деятельность по решению бизнес-задач средствами современных информационных технологий. Разработка проекта информационной системы ведется совместно с клиентом, что позволяет создать успешно работающую и удовлетворяющую все потребности заказчика корпоративную информационную систему.

Спектр бизнес-процессов, реализованных в различных КИС, может быть достаточно широк. Среди прочего это и управление продажами в различных формах, например, продажа в кредит или продажа с оплатой встречным обязательством, разнообразные бизнес-процессы, связанные с планированием, закупками, производством, хранением, персоналом, и многое-многое другое.

Информационная система может строиться с применением послойного принципа. Так, в отдельные слои можно выделить специализированное программное обеспечение (офисное, прикладное), непосредственно workflow, систему управления документами, программы поточного ввода документов, а также вспомогательное программное обеспечение для связи с внешним миром и обеспечения доступа к функционалу системы через коммуникационные средства (e-mail, Internet/intranet). Среди преимуществ такого подхода следует отметить возможность внесения изменений в отдельные программные компоненты, расположенные в одном слое, без необходимости коренных переделок на других слоях, обеспечить формальную спецификацию интерфейсов между слоями, поддерживающих независимое развитие информационных технологий и реализующих их программных средств. Причем применение открытых стандартов позволит безболезненно осуществлять переход с программных модулей одного производителя на программы другого (например, замена почтового сервера или СУД). Кроме того, послойный подход позволит повысить надежность и устойчивость к сбоям системы в целом.

***Преимущества внедрения корпоративных информационных систем:***

1. получение достоверной и оперативной информации о деятельности всех подразделений компании;
2. повышение эффективности управления компанией;
3. сокращение затрат рабочего времени на выполнение рабочих операций;
4. повышение общей результативности работы за счет более рациональной ее организации.

Самый важный вопрос. Давайте на секунду спросим себя: что даёт человеку нервная система? Конечно же, способность управлять собой, сопротивляться неблагоприятным внешним факторам и гибко реагировать на изменения окружающей среды. Если представить компанию в качестве живого организма, то КИС лучше всего подходит на роль его нервной системы, пронизывающей все органы, все частички корпоративного организма.

Повышение внутренней управляемости, гибкости и устойчивости к внешним воздействиям увеличивает эффективность компании, её конкурентоспособность, а, в конечном счёте - прибыльность. Вследствие внедрения КИС увеличиваются объёмы продаж, снижается себестоимость, уменьшаются складские запасы, сокращаются сроки выполнения заказов, улучшается взаимодействие с поставщиками. Но, несмотря на привлекательность приведённых утверждений, вопрос об окупаемости инвестиций в КИС не теряет свою актуальность. Соотношение выгоды от использования системы и ее стоимости является одним из наиболее важных факторов, оказывающих влияние на решение "покупать или не покупать". Любой инвестиционный проект, а внедрение КИС, несомненно, нужно рассматривать как инвестиционный проект, представляет собой своего рода "покупку" и, соответственно, требует оценки его стоимости и ожидаемой выгоды.

Прямую окупаемость КИС посчитать непросто, поскольку в результате внедрения оптимизируется внутренняя структура компании, снижаются трудноизмеримые транзакционные издержки. Сложно определить, например, в какой степени увеличение доходов компании явилось следствием работы КИС (читай - программной системы), а в какой - результатом настройки бизнес-процессов, то есть плодом управленческих технологий. Однако в некоторых аспектах деятельности компании оценка вполне реальна. В первую очередь это касается логистики, где внедрение КИС приводит к оптимизации материальных потоков и к снижению потребности в оборотных средствах. Постановка на базе КИС системы финансового контроллинга приводит к снижению накладных затрат компании, ликвидации убыточных подразделений и исключению из ассортимента нерентабельных продуктов.

Совсем трудно оценить эффект от ликвидации хаоса. Для того чтобы это сделать, нужно чётко представлять масштабы хаоса, что в силу самой природы беспорядка невозможно. Действительно, можете ли Вы сказать, сколько денег Ваша компания не зарабатывает (читай - теряет) из-за перекосов в ассортименте, или, скажем, из-за срыва сроков исполнения заказов? Какие ресурсы компании оказываются выведенными из оборота вследствие "посмертного" учёта и нестыковки данных в бухгалтерии, на складе и в цехах? А как оценить объём воровства и разбазаривания ресурсов?

В настоящее время для оценки эффективности IT-проектов применяется метод инвестиционного анализа Cost Benefit Analysis **(CBA)** Метод назван так, поскольку в основе лежит оценка и сравнение выгод от осуществления проекта, с затратами на его реализацию.

Глобальная цель внедрения КИС - повышение эффективности компании. Каждая компания определяет ключевые сферы, влияющие на ее эффективность, так называемые "критические факторы успеха" (Critical Success Factor -- CSF). Повышение эффективности происходит за счет реализации задач в каждой из ключевых областей. Поэтому в основе CBA лежат именно бизнес-цели компании, определенные на этапе стратегического планирования.

Но достигнуть цели можно несколькими путями, поэтому второй краеугольный камень CBA - сравнение альтернативных вариантов. При этом одним из возможных является вариант "без КИС", т. е. рассматривается развитие во времени текущей ситуации без внесения в нее каких-либо изменений. Сравнение альтернативных вариантов производится на основании измерения приносимых ими выгод и требуемых для этого затрат. Учитываются как количественные, так и качественные показатели. Анализу качественных показателей в последнее время уделяется особое внимание. Помимо соотношения выгод и затрат, альтернативные варианты также отличаются степенью риска и факторами, которые эти риски определяют. Поэтому анализ влияния таких факторов на соотношение выгод и затрат является еще одной сферой внимания CBA. Это о методах оценки конкретного случая.

**Если же говорить о статистических данных, характеризующих эффективность внедрения КИС, могу привести следующие цифры:**

- Снижение транспортно-заготовительных расходов на 60%;
- Сокращение производственного цикла по заказным изделиям на 50%;
- Сокращение количества задержек с отгрузкой готовой продукции на 45%;
- Уменьшение уровня неснижаемых остатков на складах на 40%;
- Снижение производственного брака на 35%;
- Уменьшение административно-управленческих расходов на 30%;
- Сокращение производственного цикла по базовым изделиям на 30%;
- Уменьшение складских площадей на 25%;
- Увеличение оборачиваемости средств в расчётах на 30%;
- Увеличение оборачиваемости ТМЗ на 65%;
- Увеличение количества поставок точно в срок на 80%.

Эта статистика собрана на примере западных компаний, где качество управления и так достаточно высокое. Как Вы считаете, на российской почве эффект будет больше или меньше?

Принципы построения КИС

Концепция построения КИС предусматривает наличие типовых компонентов:

1. Ядро системы, обеспечивающее комплексную автоматизацию совокупности бизнес-приложений, содержит полный набор функциональных модулей для автоматизации задач управления;
2. Система автоматизации документооборота в рамках корпорации;
3. Вспомогательные инструментальные системы обработки информации (экспертные системы, системы подготовки и принятия решений и др.) на базе хранилищ данных КИС;
4. Программно-технические средства системы безопасности КИС;
5. Сервисные коммуникационные приложения (электронная почта, программное обеспечение удаленного доступа);
6. Компоненты интернет/интранет для доступа к разнородным базам данных и информационным ресурсам, сервисным услугам;
7. Офисные программы - текстовый редактор, электронные таблицы, СУБД настольного класса и др.
8. Системы специального назначения - системы автоматизированного проектирования (САПР), автоматизированные системы управления технологическими процессами (АСУТП), банковские системы и др.

Ядром каждой производственной системы являются воплощенные в ней рекомендации по управлению производством. На данный момент существует несколько сводов таких рекомендаций. Они представляют собой описание общих правил, по которым должны производиться планирование и контроль различных стадий деятельности корпорации. Далее рассмотрены некоторые из существующих технологий управления.

К основным принципам построения КИС относятся:

1. Принцип интеграции, заключающийся в том, что обрабатываемые данные вводятся в систему только один раз и затем многократно используются для решения возможно большего числа задач; принцип однократного хранения информации;
2. Принцип системности, заключающийся в обработке данных в раз личных разрезах, чтобы получить информацию, необходимую для принятия решений на всех уровнях и во всех функциональных под системах и подразделениях корпорации; внимание не только к под системам, но и к связям между ними; эволюционный аспект – все стадии эволюции продукта, в фундаменте КИС должна лежать способность к развитию;

3. Принцип комплексности, подразумевающий автоматизацию процедур преобразования данных на всех стадиях продвижения продуктов корпорации.

Этапы проектирования КИС:

1. Анализ

Обследование и создание моделей деятельности организации, анализ (моделей) существующих КИС, анализ моделей и формирование требований к КИС, разработка плана создания КИС.

2. Проектирование

Концептуальное проектирование, разработка архитектуры КИС, проектирование общей модели данных, формирование требований к приложениям.

3. Разработка

Разработка, прототипирование и тестирование приложений, разработка интеграционных тестов, разработка пользовательской документации.

4. Интеграция и тестирование

Интеграция и тестирование приложений в составе системы, оптимизация приложений и баз данных, подготовка эксплуатационной документации, тестирование системы.

5. Внедрение

Обучение пользователей, развертывание системы на месте эксплуатации, инсталляция баз данных, эксплуатация.

6. Сопровождение

Регистрация, диагностика и локализация ошибок, внесение изменений и тестирование, управление режимами работы ИС.

Предлагаемая платформа позиционируется как основа для построения бизнес-систем для мобильных пользователей на базе платформы J2ME, а также расширения функциональности существующих систем для работы с мобильными клиентами. Основой данной платформы является защищенный протокол, обеспечивающий регистрацию пользователя в системе, двустороннюю («строгую») аутентификацию на базе ИОК, и конфиденциальность передаваемой информации. Таким образом, данная платформа ориентирована на использование в системах электронной коммерции, платежных системах, системах мобильного банкинга и других системах, где требуется обеспечение аутентичности пользователя и конфиденциальности данных.

Требования, предъявляемые к данной платформе:

− обеспечение различных способов аутентификации клиента и сервера, в том числе двухсторонней «строгой» аутентификации на базе отечественных криптографических алгоритмов и цифровых сертификатов X.509;

− наличие подсистемы многофакторной аутентификации (в том числе с использованием биометрических данных) на базе мобильного устройства;

− обеспечение конфиденциальности передаваемой информации с помощью отечественных алгоритмов шифрования;

− обеспечение удаленной регистрации пользователей и жизненного цикла ключа;

− независимость от оператора сотовой сети; − функционирование в общедоступных сетях связи.

Функционирование систем на базе данной платформы возможно в мобильных сетях общего пользования (GSM 2G/2.5G) с распространенной технологией пакетной передачи данных (GPRS/EDGE). При этом затраты на развертывание данной системы минимальны, так как не требуется доступ к оборудованию оператора сотовой связи. Система предназначена для функционирования в открытых IP-сетях.

В качестве клиента данной платформы выступает мобильный J2MEтерминал, Web-клиент или же их совокупность, при этом мобильный терминал используется в качестве дополнительного фактора аутентификации при работе с Web-клиентом. Предоставляется возможность применения различных способов аутентификации, что расширяет спектр предоставляемых сервисов для пользователей бизнес системы в зависимости от используемого способа:

1) для мобильного и Web-клиента: пользователь аутентифицируется с помощью пары логин/пароль, аутентификация сервера не предусмотрена, данные передаются по открытому каналу;

2) для мобильного клиента: двусторонняя «строгая» аутентификация как клиента так и сервера на основе цифровых сертификатов X.509 и отечественного алгоритма ЭЦП ГОСТ Р 34.10-94 (по аналогии с SSL версии 1.0 RFC 2246 [1]), канал шифруется на установленном сессионном ключе, канал шифруется на установленном сессионном ключе;

3) для Web-клиента: двусторонняя аутентификация по аналогии с п.2 осуществляется при помощи подписанного Java-апплета. Ключевой контейнер располагается в файловой системе отчуждаемого носителя, канал шифруется на установленном сессионном ключе;

4) для Web-клиента: пользователь аутентифицируется с помощью пары логин/пароль и одноразового пароля, полученного при помощи мобильного терминала (см. п.2), канал шифруется на установленном сессионном ключе;

5) для мобильного и Web-клиента: использование биометрических данных на основе цифрового подчерка пользователя. Применение биометрии для данных целей является темой будущих исследований и не рассматривается в данной работе.

В качестве мобильного терминала используется мобильное устройство J2ME с поддержкой TCP сокет соединения (большинство мобильных телефонов, имеющих профиль мобильного устройства MIDP1.0/1.1). Для хранения ключевого контейнера используется локальное хранилище RMS или (если поддерживается устройством) JSR-72 FileConnection API. Ключевой контейнер защищен буквенно-цифровым паролем, чувствительным к регистру букв и может содержать специальные символы. Ввод пароля пользователем необходим каждый раз при обращении к секретному ключу.

Web-клиентом является Web-браузер с поддержкой Java (любой современный браузер).

Серверная часть платформы представляет собой комплекс J2EEкомпонент под управлением сервера приложений JBoss (или другого, например WebSphere). Компонентами системы являются (представлено на рис. 1):

- центр сертификации;
- центр регистрации;
- сервер БД;
- Web-клиент пользователя;
- консоль администратора;
- XML-RPC сервер;
- сетевой шлюз;
- компонент бизнес-логики.

Шлюз обеспечивает взаимодействие с мобильными клиентами по защищенному протоколу. Далее информационный поток пользователей инкапсулируется в XML-RPC запросы поверх HTTPS. Таким образом, обеспечивается интеграция с другими системами на базе J2EE-платформы, так как нет необходимости встраивания собственных сокет-серверов в J2EE сервер приложений.

Центр регистрации обеспечивает управление учетными данными пользователей (регистрационная информация пользователя, статус сертификата пользователя, история отозванных сертификатов и т.д.) и взаимодействием с сервером БД (поддерживается широкий спектр БД).

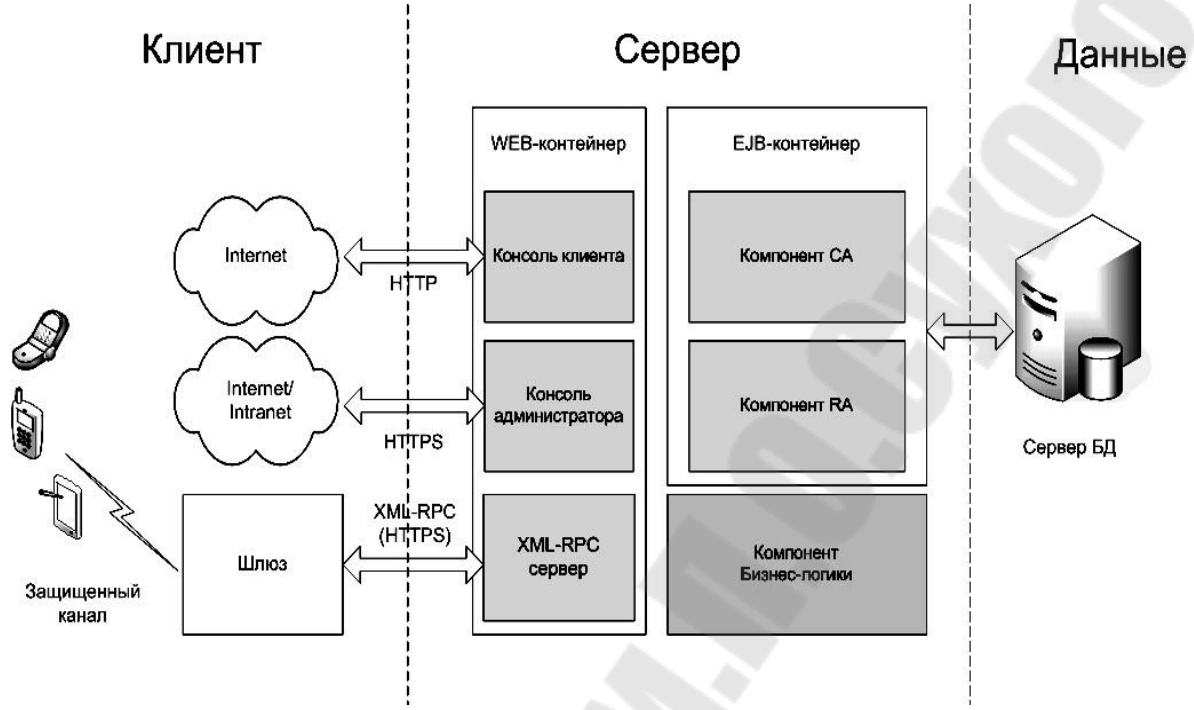


Рис. 1. Архитектура системы

Центр сертификации обеспечивает издание сертификатов пользователей и сертификата аутентификации сервера. В качестве центра сертификации может выступать сторонний сертифицированный ЦС, взаимодействие с которым может быть обеспечено по одному из доступных протоколов.

С помощью Web-клиента пользователь системы получает доступ в систему и в зависимости от типа аутентификации, получает соответствующие права доступа: информацию о текущем статусе сертификата в случае ввода пользовательского пароля без установки защищенного соединения и полный доступ к системе в случае использования одного из способов «строгой» аутентификации.

С помощью консоли администратора производится управление учетными записями пользователей (добавление новых пользователей, изменение учетных данных пользователей, установка требуемого типа аутентификации для каждого пользователя и т.п.), изменение статусов сертификатов или их отзыв, а также общее администрирование компонентов системы.

Компонент бизнес-логики представляет собой собственно интерфейс (например, WSDL как стандартный интерфейс взаимодействия сервисов SOA-платформ, RPC или программный интерфейс) взаимодействия с бизнессистемой, для которой обеспечивается безопасность работы с мобильными пользователями.

Регистрация пользователя происходит в два этапа. На первом этапе пользователь получает одноразовый пароль в конверте, с помощью которого будет происходить первичная аутентификация пользователя на сервере. На

втором этапе происходит генерация ключевой пары пользователем и создание запроса на сертификат.

После получения запроса, сервер получает регистрационные данные пользователя из базы данных и происходит издание сертификата пользователя. Клиенту сертификата отправляется блок данных состоящий из корневого сертификата Центра сертификации, сертификата аутентификации сервера и сертификата аутентификации клиента. Данный блок зашифрован с помощью ключа, созданного на базе одноразового пароля. Таким образом, происходит первичная аутентификация сервера на стороне клиента. После получения данного блока данных и сохранения сертификатов в безопасном хранилище, одноразовый пароль уничтожается.

После прохождения процедуры регистрации, пользователь готов для работы в системе. Перед соединением с сервером пользователь вводит пароль доступа к секретному ключу. Далее, клиентское приложение формирует блок аутентификации пользователя, состоящий из собственного сертификата аутентификации и сообщения Diffie-Hellman. Данный блок подписывается на секретном ключе клиента по алгоритму ЭЦП ГОСТ Р 34.10-94 и отравляется серверу. После завершения процесса аутентификации сервер и клиент согласовывают сессионный ключ в соответствии с RFC 4357 [2]. Шифрование передаваемой информации осуществляется с помощью
ГОСТ 28147-89 в режиме OFB. В случае разрыва соединения, процесс аутентификации пользователя повторяется.

Выбор соответствующих криптографических алгоритмов и алгоритмов ЭЦП связан с оценкой скорости выполнения криптографических операций на мобильных устройствах на базе платформы J2ME. Так, процесс генерации ключевой пары ГОСТ Р 34.10–94 занимает в среднем 30–45 секунд, а процесс подписания и проверки подписи в среднем 3-5 секунд. Соответствующие показатели для ГОСТ Р 34.10–2001 составляют 20–30 минут и 7–8 минут соответственно, что неприемлемо для использования в мобильных сервисах, которые призваны обеспечивать быстрый повсеместный доступ к сервису.

К особенностям разработанной системы относятся:

1. Поддержка всех мобильных устройств, имеющих профиль MIDP1.0/2.0 с возможностью создания TCP соединения.
2. Функционирование в стандартных сетях связи.
3. Независимость от оператора сотовой связи.
4. Интегрируемость в существующие J2EE бизнес-системы.
5. Гибкость развертывания системы в зависимости от требований безопасности бизнес-системы.
6. Поддержка различных способов аутентификации и разграничение на их основе предоставляемых бизнес-системой сервисов.
7. Поддержка методов многофакторной аутентификации, в том числе используя биометрические данные пользователя.

8. Модульная архитектура, возможность наращивания функционала системы.

Таким образом, данная платформа может быть интегрирована в существующие корпоративные информационные системы (КИС) на базе SOA-решений Java для обеспечения защищенного информационного обмена с мобильными пользователями, обеспечивать гибкость предоставляемых сервисов и ресурсов в зависимости от прав доступа и типа аутентификации пользователя, а также быть основой для построения новых КИС.

## Лекция 2 Доступ к БД из Java.

JDBC – это стандартное Java API для работы с реляционными СУБД. В реальности оказывается, что некоторые объектные СУБД и иногда даже совсем не СУБД предоставляют JDBC интерфейс для работы с данными.

В этом документе будут приведены самые базовые сведения об использовании JDBC.

### JDBC Драйверы

Как правило, все поставщики СУБД распространяют библиотеки для работы со своими СУБД для разных платформ. Такие библиотеки в Java называются JDBC драйверами.

Раньше, когда Java ещё не занимала достойного места на рынке, считалось, что будут популярны разные виды драйверов, в т.ч. реализованные как вызовы более старого интерфейса ODBC (JDBC-ODBC Bridge). Сегодня практически любой драйвер представляет собой JAR файл с классами, реализующими интерфейсы из пакетов java.sql и javax.sql.

Для того, чтобы начать работать с СУБД в classpath необходимо добавить JDBC драйвер для соответствующей СУБД.

### Основные классы JDBC:

- DriverManager – Основная задача данного класса – получение соединения с СУБД по JDBC URL
- Connection – Соединение с СУБД.
- Statement – Объекты этого класса можно порождать в контексте соединения и использовать для выполнения запросов.
- PreparedStatement – То же, что и Statement, но запрос разбирается только один раз, а потом в него подставляются параметры и он исполняется столько раз, сколько нужно.

- CallableStatement – То же самое, что и PreparedStatement, но для исполнения хранимых процедур и функций СУБД
- ResultSet – Класс, представляющий собой курсор базы данных. Другими словами – это итератор по результатам выполненного запроса. Как правило, конкретная реализация ResultSet подкачивает строки выборки небольшими буферами для оптимизации.

- Blob – Один из типов значений колонки в выборке, который соответствует Binary Large Object типу конкретной СУБД. Из него можно читать как из файла с помощью InputStream.
- Clob – То же самое что и Blob, но для данных типа Character Large Object.
- DataSource – Данный интерфейс очень важен в JavaEE. Обычно он представляет собой пул соединений с СУБД, которые можно повторно использовать по мере необходимости и после чего возвращать обратно в пул без закрытия сетевого соединения

**Основные приёмы работы**

Получение соединения

Способы получения соединения с СУБД в Java SE и в Java EE сильно отличаются. В этом документе будет описан только обычный способ, характерный для Java SE.

Обычно в комплекте с драйвером идёт документация, в которой подробно всё описано, но чаще всего пишут что-то такое:

```
public class LoadDriver {    public static
void main(String[] args) {
      try {
            // The newInstance() call is a work around for some
            // broken Java implementations
            Class.forName("com.mysql.jdbc.Driver").newInstance();
      } catch (Exception ex) {
            // handle the error
      }
  }
}
```

Сперва необходимо загрузить главный класс драйвера. В нашем случае – com.mysql.jdbc.Driver.

Потом используем DriverManager:

```
import java.sql.Connection; import
java.sql.DriverManager;
import java.sql.SQLException; try
{
  Connection                                conn                                =
DriverManager.getConnection("jdbc:mysql://localhost/test");
  // Do something with the Connection
} catch (SQLException ex) {
  // handle any errors
  System.out.println("SQLException: " + ex.getMessage());
  System.out.println("SQLState: "     + ex.getSQLState());
  System.out.println("VendorError: "  + ex.getErrorCode());
}
```

Судя по всему, при загрузке класса драйвера срабатывает блок статической инициализации, который регистрирует в DriverManager'e себя и что он обрабатывает соединения, URL которых имеет вид jdbc:mysql://*

С соединениями примерно такая же беда, как с Input/Output потоками. Их необходимо закрывать, иначе очень быстро наступит предельно допустимое количество открытых соединений и к ней нельзя будет подключиться.

Итак, соединение получено. Теперь нужно научиться делать самые простые вещи. Выборку данных (как правило это SQL оператор SELECT) и их модификацию (INSERT, UPDATE и DELETE).

В JDBC операции чтения и модификации выполняются немного поразному, т. к. при чтении нужно возвращать сами данные (ResultSet), а при модификации можно обойтись количеством строк, которые были изменены в хоте выполнения запроса.

Читаем из базы (Select):

```
Connection conn = null;
try {
   conn = ds.getConnection();
   PreparedStatement ps = conn.prepareStatement("SELECT * FROM
SOME_TABLE WHERE id=? AND name=? AND time=?");
   ps.setInteger(1,                  10);
ps.setString(2,                "Mike");
ps.setLong(3,
System.currentTimeMillis());

   ResultSet rs = rs;
while (rs.next()) {
     // Извлекаем значения колонок текущей строки выборки
```

```
        String ctxName = rs.getString("col1");
    int ctxId      = rs.getInteger("col2");
        int pending    = rs.getLong("col3");
      }
    } finally {
      JdbcDaoHelper.safeClose(conn, log);
    }
```

Обычно стоит написать утилитный метод, который закрывает соединение и игнорирует исключения (JdbcDaoHelper). Очень важно закрывать соединение при любых обстоятельствах.

Пишем в базу (Update, Delete и Insert):

```
Connection conn = null;
try {
conn = connectionPool.getConnection();
PreparedStatement ps = conn.prepareStatement(
"delete from SOME_TABLE where id=?"
);
ps.setInt(1, someId); ps.execute();
} finally {
JdbcDaoHelper.safeClose(conn, log);
}
```

Вот ещё один пример:

```
public int uploadFile(InputStream is, int fileId) throws IOException,
SQLException
{
    Connection conn = null;
    try {
      conn = ds.getConnection();
      PreparedStatement ps = conn.prepareStatement("insert into
FILE_BODIES (id, file_id, data) values (null, ?, ?)");
      ps.setInt(1, fileId);
    ps.setBlob(2, is);
    ps.execute();          if (
    !rs.next() ) {
    throw new
    SQLException("Hmm..
```

```
No keys were
generated!");
        }
        return rs.getInt(1);
    } finally {
        JdbcDaoHelper.safeClose(conn, log);
    }
}
```

Это функция, которая загружает в СУБД файл как Blob-значение. В этом примере стоит обратить внимание на следующее. Во-первых, СУБД сама генерирует значение первичного ключа. В соответствующей схеме данных в таблице FILE_BODIES задан автоматически инкрементирующийся первичный ключ.

Мы просим СУБД вернуть сгенерированные значения ключа (мы собираемся вернуть идентификатор вставленной записи для возможных дальнейших действий).

Работа с UPDATE по сути дела ничем не отличается от DELETE, разве что синтаксисом SQL запроса.

**Именование колонок**

ResultSet позволяет извлекать данные из колонок по индексу и по имени (в примерах мы использовали только извлечение по имени). Индекс начинается с 1 и соответствует порядку колонок в результирующей выборке или в таблице, если в запросе используется *.

Предпочтительно использовать именно доступ по имени и не использовать * в запросах в качестве списка колонок. Дело в том, что порядок колонок и их индекс может меняться в ходе эволюции приложения, и это будет приводить к отказам в коде, который полагается на эти индексы.

# РАЗДЕЛ 2 РАЗРАБОТКА РАСПРЕДЕЛЕННЫХ СИСТЕМ НА ОСНОВЕ ТЕХНОЛОГИИ JAVA

## Лекция 3 Клиентские и серверные приложения.

Эта лекция посвящена описанию пакета java.net. Java поддерживает протокол TCP/IP, во-первых, расширяя свой интерфейс потоков вводавывода, описанного в предыдущей лекции, и во вторых, добавляя возможности, необходимые для построения объектов ввода-вывода при работе в сети.

**InetAddress**

Java поддерживает адреса абонентов, принятые в Internet, с помощью класса InetAddress. Для адресации в Internet используются служебные функции, работающие с обычными, легко запоминающимися символическими именами, эти функции преобразуют символические имена в
32-битные адреса.

**Фабричные методы**

В классе InetAddress нет доступных пользователю конструкторов. Для создания объектов этого класса нужно воспользоваться одним из его фабричных методов. Фабричные методы — это обычные статические методы, которые возвращают ссылку на объект класса, которому они принадлежат. В данном случае, у класса InetAddress есть три метода, которые можно использовать для создания представителей. Это методы getLocalHost, getByName и getAllByName.

В приведенном ниже примере выводятся адреса и имена локальной машины, локального почтового узла и WWW-узла компании, в которой работает автор.

```
InetAddress Address = InetAddress.getLocalHost();
System.out.println(Address);
Address = InetAddress.getByName("mailhost");
System.out.println(Address);
InetAddress SW[] = InetAddress.getAllByNarne(“www.starwave.com");
System.out.println(SW);
```

У класса InetAddress также есть несколько нестатических методов, которые можно использовать с объектами, возвращаемыми только что названными фабричными методами:

- getHostName() возвращает строку, содержащую символическое имя узла, соответствующее хранящемуся в данном объекте адресу Internet.
- getAddress() возвращает байтовый массив из четырех элементов, в котором в порядке, используемом в сети, записан адрес Internet, хранящийся в данном объекте.
- toString() возвращает строку, в которой записано имя узла и его адрес.

**Дейтаграммы**

Дейтаграммы, или пакеты протокола UDP (User Datagram Protocol) — это пакеты информации, пересылаемые в сети по принципу “fire-and-forget” (выстрелил и забыл). Если вам надо добиться оптимальной производительности, и вы в состоянии минимизировать затраты на проверку целостности информации, пакеты UDP могут оказаться весьма полезными.

UDP не предусматривает проверок и подтверждений при передаче информации. При передаче пакета UDP по какому-либо адресу нет никакой гарантии того, что он будет принят, и даже того, что по этому адресу вообще есть кому принимать такие пакеты. Аналогично, когда вы получаете дейтаграмму, у вас нет никаких гарантий, что она не была повреждена в пути или что ее отправитель все еще ждет от вас подтверждения ее получения.

Java реализует дейтаграммы на базе протокола UDP, используя для этого два класса. Объекты класса DatagramPacket представляют собой контейнеры с данными, а DatagramSocket — это механизм, используемый при передаче и получении объектов DatagramPacket.

Сокеты “для клиентов”

TCP/IP-сокеты используются для реализации надежных двунаправленных, ориентированных на работу с потоками соединений точка-точка между узлами Internet. Сокеты можно использовать для соединения системы вводавывода Java с программами, которые могут выполняться либо на локальной машине, либо на любом другом узле Internet. В отличие от класса DatagramSocket, объекты класса Socket реализуют высоконадежные устойчивые соединения между клиентом и сервером.

В пакете java.net классы Socket и ServerSocket сильно отличаются друг от друга. Первое отличие в том, что ServerSocket ждет, пока клиент не установит с ним соединение, в то время, как обычный Socket трактует недоступность чего-либо, с чем он хочет соединиться, как ошибку. Одновременно с созданием объекта Socket устанавливается соединение между узлами Internet. Для создания сокетов вы можете использовать два конструктора:

- Socket(String host, int port) устанавливает соединение между локальной машиной и указанным портом узла Internet, имя которого было передано конструктору. Этот конструктор может возбуждать исключения UnknownHostException и IOException.
- Socket(InetAddress address, int port) выполняет ту же работу, что и первый конструктор, но узел, с которым требуется установить соединение, задается не строкой, а объектом InetAddress. Этот конструктор может возбуждать только IOException.

Из объекта Socket в любое время можно извлечь информацию об адресе Internet и номере порта, с которым он соединен. Для этого служат следующие методы:

• getInetAddressQ возвращает объект InetAddress, связанный с данным объектом Socket.

• getPort() возвращает номер порта на удаленном узле, с которым установлено соединение.

• getLocalPort() возвращает номер локального порта, к которому присоединен данный объект.

После того, как объект Socket создан, им можно воспользоваться для того, чтобы получить доступ к связанным с ним входному и выходному потокам. Эти потоки используются для приема и передачи данных точно так же, как и обычные потоки ввода-вывода, которые мы видели в предыдущей лекции: • getInputStream() возвращает InputStream, связанный с данным объектом. • getOutputStream() возвращает OutputStream, связанный с данным объектом. • close() закрывает входной и выходной потоки объекта Socket.

Приведенный ниже очень простой пример открывает соединение с портом 880 сервера “timehost” и выводит полученные от него данные.

```
import java.net.*; import java.io.*; class TimeHost {
public static void main(String args[]) throws Exception {
int c;
Socket s = new Socket("timehost.starwave.com",880);
InputStream in = s.getInputStream(); while ((c =
in.read()) != -1) {
System.out.print( (char) c);
}
s.close();
} }
```

Сокеты “для серверов”

Как уже упоминалось ранее, Java поддерживает сокеты серверов. Для создания серверов Internet надо использовать объекты класса ServerSocket. Когда вы создаете объект ServerSocket, он регистрирует себя в системе, говоря о том, что он готов обслуживать соединения клиентов. У этого класса есть один дополнительный метод accept(), вызов которого блокирует подпроцесс до тех пор, пока какой-нибудь клиент не установит соединение по соответствующему порту. После того, как соединение установлено, метод accept() возвращает вызвавшему его подпроцессу обычный объект Socket.

Два конструктора класса ServerSocket позволяют задать, по какому порту вы хотите соединяться с клиентами, и (необязательный параметр) как долго вы готовы ждать, пока этот порт не освободится.

• ServerSocket(int port) создает сокет сервера для заданного порта.

• ServerSocket(int port, int count) создает сокет сервера для заданного порта. Если этот порт занят, метод будет ждать его освобождения максимум count миллисекунд.

URL

URL (Uniform Resource Locators — однородные указатели ресурсов) — являются наиболее фундаментальным компонентом “Всемирной паутины”. Класс URL предоставляет простой и лаконичный программный интерфейс для доступа к информации в Internet с помощью URL.

У класса URL из библиотеки Java - четыре конструктора. В наиболее часто используемой форме конструктора URL адрес ресурса задается в строке, идентичной той, которую вы используете при работе с браузером:

*URL(String spec)*

Две следующих разновидности конструкторов позволяют задать URL, указав его отдельные компоненты:

*URL(String protocol, String host, int port, String file)*

*URL(String protocol, String host, String file)*

Четвертая, и последняя форма конструктора позволяет использовать существующий URL в качестве ссылочного контекста, и создать на основе этого контекста новый URL. *URL(URL context, String spec)*

В приведенном ниже примере создается URL, адресующий www-страницу (поставьте туда свой адрес), после чего программа печатает свойства этого объекта.

```
import java.net.URL; class myURL { public static void
main(String args[]) throws Exception { URL hp = new
URL("http://coop.chuvashia.edu");
System.out. println("Protocol: " + hp.getProtocol());
System.out.printin("Port: " +        hp.getPort());
System.out.println("Host: " +        hp.getHost());
System.out.println("File: " + hp.getFile());
System.out.println("Ext: " + hp.toExternaLForm());
} }
```

Для того, чтобы извлечь реальную информацию, адресуемую данным URL, необходимо на основе URL создать объект URLConnection, воспользовавшись для этого методом openConnection().

**URLConnection**

URLConnection — объект, который мы используем либо для проверки свойств удаленного ресурса, адресуемого URL, либо для получения его содержимого. В приведенном ниже примере мы создаем URLConnection с помощью метода openConnection, вызванного с объектом URL. После этого мы используем созданный объект для получения содержимого и свойств документа.

```
import java.net.*; import
java.io.*; class
localURL {
public static void main(String args[]) throws Exception {
int c;
```

```
URL hp = new URL("http", "127.0.0.1", 80, "/");
URLConnection hpCon = hp.openConnection();
System.out.println("Date: " + hpCon.getDate());
System.out.println("Type: " + hpCon.getContentType());
System.out.println("Exp: " + hpCon.getExpiration());
System.out.println( "Last M: " + hpCon.getLastModified());
System.out.println("Length: " + hpCon.getContentLength());
if (hpCon.getContentLength() > 0) {
System.out.println("=== Content ===");
InputStream input = hpCon.getInputStream();
int i=hpCon.getContentLength(); while (((c =
input. read()) != -1) && (--i > 0)) {
System.out.print((char) c);
}
input.close();
} else
{
System.out.println("No Content Available");
} }
}
```

Эта программа устанавливает HTTP-соединение с локальным узлом по порту 80 (у вас на машине должен быть установлен Web-сервер) и запрашивает документ по умолчанию, обычно это - index.html. После этого программа выводит значения заголовка, запрашивает и выводит содержимое документа.

**Web-сервисы**

Web-сервисы - новое слово в технологии распределенных систем. Спецификация Open Net Environment (ONE) корпорации Sun Microsystems и инициатива .Net корпорации Microsoft обеспечивают инфраструктуры для написания и развертывания Web -сервисов. В настоящий момент имеется несколько определений Web-сервиса. Web -сервисом может быть любое приложение, имеющее доступ к Web, например, Web -страница с динамическим содержимым. В более узком смысле Web -сервис - это приложение, которое предоставляет открытый интерфейс, пригодный для использования другими приложениями в Web.

Спецификация ONE Sun требует, чтобы Web -сервисы были доступны через HTTP и другие Web -протоколы, чтобы дать возможность обмениваться информацией посредством XML-сообщений и чтобы их можно было найти через специальные сервисы - сервисы поиска. Для доступа к Web -сервисам разработан специальный протокол - Simple Object Access Protocol (SOAP),который представляет средства взаимодействия на базе XML для

многих Web -сервисов. Web -сервисы особенно привлекательны тем, что могут обеспечить высокую степень совместимости между различными системами. Огромный потенциал Web -сервисов определяется не технологией, примененной для их создания. HTTP, XML и другие протоколы, используемые Web -сервисами, не новы. Функциональная совместимость и масштабируемость Web -сервисов подразумевает, что разработчики могут быстро создавать большие приложения и более крупные Web -сервисы из меньших Web -сервисов. Спецификация Sun Open Net Environment описывает архитектуру для созданияинтеллектуальных Webсервисов.Интеллектуальные Web -сервисы задействуют общее операционное окружение. Совместно используя контекст, интеллектуальныеWeb -сервисы могут выполнять стандартную аутентификацию для финансовых транзакций, предоставлять рекомендации и указания в зависимости от географического местоположения компаний, участвующих в электронном бизнесе. Согласно определению W3C, " WSDL - формат XML для описания сетевых сервисов как набора конечных операций, работающих при помощи сообщений, содержащих документно-ориентированную или процедурноориентированную информацию". Документ WSDL полностью описывает интерфейс Web -сервиса с внешним миром. Он предоставляет информацию об услугах, которые можно получить, воспользовавшись методами сервиса, и способах обращения к этим методам. Таким образом, в случае если сигнатура метода Web -сервиса точно не известна (например, она изменилась со временем), у целевого Web -сервиса может быть запрошено WSDL -описание - файл, в котором эта информация будет содержаться.

Следующим слоем технологии является сервис Universal Description, Discovery and Integration (UDDI).Эта технология предполагает ведение реестра Web -сервисов. Подключившись к этому реестру, потребитель сможет найти Web -сервисы, которые наилучшим образом подходят для решения его задач. Технология UDDI дает возможность поиска и публикации нужного сервиса, причем эти операции могут быть выполнены как человеком, так и другим Web -сервисом или специальной программойклиентом. UDDI, в свою очередь, также представляет собой Web -сервис.

Таким образом, Web -сервисы являются еще одной реализацией системного программного обеспечения промежуточного слоя. Отличительной чертой этой технологии является ее независимость от используемого программного и аппаратного обеспечения, а также использование широко применяемых открытых стандартов (таких как XML ) и стандартных коммуникационных протоколов.

В настоящее время Web -сервисы являются очень активно продвигаемой технологией и позиционируются как средство решения целого ряда задач.

Следует отметить, что с их применением могут строиться и так называемые "стандартные" приложения, где в качестве Web -сервиса оформляется серверная часть.

**Простой протокол доступа к объектам (SOAP)**

Базовым протоколом, обеспечивающим взаимодействие в среде Web сервисов, является протокол SOAP.

Протокол SOAP разработали корпорации IBM, Lotus Development Corporation, Microsoft, Develop-Mentor и Userland Software. Этот протокол основан на HTTP-XML. Он позволяет приложениям взаимодействовать между собой через Internet, используя для этого XML -документы, называемые сообщениями SOAP. Протокол SOAP совместим с любой объектной моделью, поскольку он включает только те функции и методы, которые абсолютно необходимы для формирования коммуникационной инфраструктуры. Таким образом, SOAP является независимым от платформы и конкретных приложений, а для его реализации может применяться любой язык программирования. SOAP поддерживает практически любой транспортный протокол. SOAP также поддерживает любые методы кодирования данных, которые позволяют приложениям, основанным на SOAP, посылать в сообщениях SOAP информацию практически любого типа (например, изображения, объекты, документы и т.д.).

Сообщение SOAP содержит конверт, который описывает содержимое, предполагаемого получателя сообщения и требования к обработке сообщения. Необязательный элемент **header** (заголовок) сообщения SOAP содержит инструкции по обработке для приложений, которые принимают сообщение. Заголовок также может содержать информацию о маршрутизации. С помощью заголовка **header** поверх SOAP могут надстраиваться более сложные протоколы. Записи в заголовке могут модульно расширять сообщение для таких задач, как аутентификация, управление транзакциями и проведение платежей. Тело SOAP -сообщения содержит специфичные для приложения данные, предназначенные для предполагаемого получателя сообщения.

**Исходный код Web-сервиса**

После того, как указанные настройки будут закончены, можно приступить непосредственно к примеру.

Файл с исходным кодом Web -сервиса располагается в директории src, называется Hello.java и имеет следующий вид:

```
1  // Hello.java
2  package helloservice.endpoint;
3
4  import javax.jws.WebMethod;
5  import javax.jws.WebService;
6
7      @WebService()
```

```
8    public class Hello {
9    private String message = new String("Hello, ");
10
11       @WebMethod()
12       public String sayHello(String name) {
13       return message + name + ".";
14       }
15       }
```

Первое, что обращает на себя внимание, - удивительная краткость написанного кода. Но не стоит обольщаться: дело в том, что технология разработки Web -сервисов, которую мы будем использовать, просто скрывает от разработчика большую часть работы по реализации Web -сервиса. Фактически, все, что должен сделать разработчик, - реализовать код самих вызываемых методов; абсолютно всю работу по реализации механизмов, позволяющих вызывать эти методы удаленно, берет на себя используемая нами технология.

Рассматриваемый класс **Hello** удовлетворяет всем указанным ограничениям. Кроме того, он объявляет единственный метод, аннотированный как **WebMethod** (строка 11), который принимает параметр типа **String** и возвращает его же с присоединенной в начале константной строкой.

Собственно, на этом разработка Web -сервиса заканчивается. Следующее, что необходимо сделать, - откомпилировать его, пропустить через утилиту **wsgen** для генерации вспомогательных классов, создать **warfile**,содержащий в себе откомпилированное приложение и необходимые ресурсы, и затем разместить и зарегистрировать его на сервере приложений. В комплекте с примерами, поставляемыми в пакете The Java Web Services Tutorial,поставляются также скрипты для их компиляции. Эти скрипты предназначены для специального инструментального средства компиляции, которое называется ant (исполняющая часть **ant** устанавливается вместе с Sun Java System Application Server).

Разработчики примеров для пакета The Java Web Services Tutorial постарались на славу, и теперь для компиляции и установки приложения необходимо выполнить лишь несколько простых команд. Мы воспользуемся этим обстоятельством, а затем подробно рассмотрим, что стоит за каждой из этих простых команд и какие действия при этом выполняются.

**Компиляция и инсталляция на сервере приложений**

Итак, первое, что предстоит сделать, - откомпилировать приложение. Для компиляции в настройках сборки определена специальная цель (target) build.

Набрав в командной строке команду **asant build** (**asant** - вызов командного файла, запускающего **ant, build** - имя цели, которую он должен выполнить), получим следующий вывод:

Buildfile: build.xml

javaee-home-test:
init:
prep
are:
[echo] Creating the required directories....
[mkdir] Created dir: H:\Java\jwstutorial20_new\examples\
jaxws\helloservice\build

compile-service:
[echo] Compiling the server-side source code ... [javac]
Compiling 1 source file to H:\Java\
jwstutorial20_new\examples\jaxws\helloservice\build
[wsgen] command line: wsimport -classpath
H:\Java\AppServer\lib\activation.jar;
H:\Java\AppServer\lib\admin-cli.jar;
H:\Java\AppServer\lib\appserv-admin.jar;
H:\Java\AppServer\lib\appserv-cmp.jar;
H:\Java\AppServer\lib\appserv-deployment-client.jar;
H:\Java\AppServer\lib\appserv-ext.jar; H:\Java\AppServer\lib\appserv-jstl.jar;
H:\Java\AppServer\lib\appserv-jwsacc.jar;
H:\Java\AppServer\lib\appserv-launch.jar;
H:\Java\AppServer\lib\appserv-rt.jar;
H:\Java\AppServer\lib\appserv-tags.jar;
H:\Java\AppServer\lib\appserv-upgrade.jar;
H:\Java\AppServer\lib\appserv-ws.jar;
H:\Java\AppServer\lib\com-sun-commons-launcher.jar;
H:\Java\AppServer\lib\com-sun-commons-logging.jar;
H:\Java\AppServer\lib\dbschema.jar;
H:\Java\AppServer\lib\j2ee-svc.jar;
H:\Java\AppServer\lib\j2ee.jar;
H:\Java\AppServer\lib\javaee.jar;
H:\Java\AppServer\lib\jhall.jar;
H:\Java\AppServer\lib\jmxremote_optional.jar;
H:\Java\AppServer\lib\jsf-impl.jar;
H:\Java\AppServer\lib\mail.jar;
H:\Java\AppServer\lib\sun-appserv-ant.jar;

```
H:\Java\AppServer\lib\toplink-essentials-agent.jar;
H:\Java\AppServer\lib\toplink-essentials.jar;
H:\Java\AppServer\jdk\lib\tools.jar;
H:\Java\jwstutorial20_new\examples\jaxws\helloservice\build -d
H:\Java\jwstutorial20_new\examples\jaxws\helloservice\ build -keep -s
H:\Java\jwstutorial20_new\examples\jaxws\helloservice\ build
-verbose helloservice.endpoint.Hello [wsgen] Note:      ap round: 1
[wsgen] [ProcessedMethods Class: helloservice.endpoint.Hello]
[wsgen] [should process method: sayHello hasWebMethods: true ]
[wsgen] [endpointReferencesInterface: false]
[wsgen] [declaring class has WebSevice: true]
[wsgen] [returning: true]
[wsgen] [WrapperGen - method: sayHello(java.lang.String)]  [wsgen]
[method.getDeclaringType():
helloservice.endpoint.Hello]
[wsgen] [requestWrapper: helloservice.endpoint.jaxws.SayHello]
[wsgen] [ProcessedMethods Class: java.lang.Object]
[wsgen] helloservice\endpoint\jaxws\SayHello.java
[wsgen] helloservice\endpoint\jaxws\SayHelloResponse.java
[wsgen] Note:      ap round: 2
[wsgen] [completing model for endpoint:
helloservice.endpoint.Hello]
[wsgen] [ProcessedMethods Class: helloservice.endpoint.Hello]
[wsgen] [should process method: sayHello hasWebMethods: true ]
[wsgen] [endpointReferencesInterface: false]
[wsgen] [declaring class has WebSevice: true]
[wsgen] [returning: true]
[wsgen] [WebServiceReferenceCollector - method:
sayHello(java.lang.String)]
[wsgen] [ProcessedMethods Class: java.lang.Object]
build-service: build:
BUILD SUCCESSFUL
Total time: 9 seconds
```

Поскольку мы намеренно включили опцию вывода отладочной информации для **ant**,вывод получился довольно обширный.

Первое, что делается для компиляции программы, - создается специальная директория build,в которую будут помещены откомпилированные модули. Она создается втекущей директории. Затем вызывается компилятор javac, который компилирует наш класс **Hello.java**,а результат компиляции кладет в директорию build.Поскольку класс **Hello** определен в пакете **helloservice.endpoint**,в директории build будет создана

соответствующая система каталогов и файл **Hello.class** будет помещен в каталог ./build/ helloservice/endpoint.

Следующим шагом вызывается утилита **wsgen**,которая формирует вспомогательные классы. По умолчанию исходные коды этих классов после компиляции уничтожаются, однако, выставив опцию **keep=true** (эта и другие опции могут быть установлены в файле **build.properties**),исходные коды можно сохранить. Помещаются они в пакет **jaxws** того же пакета, которому принадлежит и класс. Соответственно, для нашего примера исходные файлы (а затем и откомпилированные классы) будут располагаться в директории ./build/ helloservice/endpoint/jaxws. После того как утилита wsgen отработала, мы имеем откомпилированный пакет **helloservice.endpoint.jaxws**,содержащий необходимые вспомогательные классы. На этом шаге компиляция нашего Web -сервиса закончена. Следующим этапом необходимо подготовить модуль развертывания. В нашем случае это делается с помощью команды:

asant create-war

Вывод получаем следующий:

Buildfile: build.xml

prepare-assemble:
[echo] Creating the assemble directory....
[mkdir] Created dir: H:\Java\jwstutorial20_new\examples\
jaxws\helloservice\assemble
[mkdir] Created dir: H:\Java\jwstutorial20_new\examples\
jaxws\helloservice\assemble\war

create-war:
[echo] Creating the WAR ...
[war] Building war: H:\Java\jwstutorial20_new\examples\
jaxws\helloservice\assemble\war\hello-jaxws.war
BUILD SUCCESSFUL
Total time: 3 seconds

Создается отдельный каталог assemble,в нем создается каталог war, в котором формируется файл **hello-jaxws.war**.Этот файл представляет собой архив, в который помещены откомпилированные файлы нашего приложения и некоторые вспомогательные файлы. Теперь у нас полностью готов модуль развертывания, который мы можем инсталлировать в сервере приложений. Инсталляция может быть выполнена командой: asant deploy

Результат выполнения команды следующий:

Buildfile: build.xml deploy: admin_command_common:

[echo] Doing admin task deploy assemble/war/hello-jaxws.war
[sun-appserv-admin] Executing: deploy --port 4848 --host localhost --passwordfile "H:\Java\jwstutorial20_new\examples\common\admin-password.txt" --user admin assemble/war/ hello-jaxws.war
[sun-appserv-admin] Command deploy executed successfully.

BUILD SUCCESSFUL
Total time: 43 seconds

В процессе инсталляции, кроме прочего, был сгенерирован WSDL -файл, описывающий установленный Web -сервис. Как уже говорилось, этот файл содержит полное описание Web -сервиса, включая названия его методов, а также количество и типы передаваемых и возвращаемых параметров. Этот файл является важной составляющей частью технологии, поскольку он позволяет строить приложения, осуществляющие динамические вызовы методов Web -сервисов. Кроме того, этот файл может быть использован для автоматической генерации вспомогательных классов (классов- proxy ) для обращения к Web -сервису.

**Тестирование Web-сервиса**

Итак, наш Web -сервис успешно инсталлирован. Осталось только убедиться в том, что он действительно работает. Чуть позже мы напишем специальное приложение-клиент, которое будет обращаться к нашему Web сервису, а пока воспользуемся средствами, предоставляемыми нам Sun Java System Application Server.Дело в том, что этот сервер приложений способен самостоятельно динамически выстроить среду для вызова методов инсталлированных в нем Web -сервисов. Всей необходимой информацией, а именно: имена публикуемых методов, количество и тип принимаемых и возвращаемых методами параметров - он обладает.

Для того чтобы воспользоваться указанной возможностью, нужно выбрать нужный нам сервис в списке сервисов (в правой части окна браузера) и нажать кнопку "Test" .

Откроется новое окно браузера, в котором отобразится динамически построенная сервером страница. На этой странице перечислены все опубликованные методы Web -сервиса (в нашем случае - один метод **sayHello)** и реализован интерфейс для их вызова. Если ввести в соответствующее поле строку и нажать кнопку - вызовется метод Web сервиса и введенное значение будет передано ему в качестве параметра. Кроме всего прочего, на результирующей странице отобразятся SOAP сообщения, соответственно, отправленные Web -сервису и пришедшие от него в качестве ответа.

Результирующая страница будет иметь следующий вид.

На странице видны значения и типы переданных параметров, ответ, который возвратил метод Web -сервиса, - как и ожидалось, ответ представляет

собой строку "Hello, Web-service test", - а также отправленный и полученный пакеты.

Таким образом, разработанный нами Web -сервис успешно инсталлирован в сервере приложений и может обрабатывать запросы клиентов, в чем мы убедились, использовав тестовое окружение, предоставляемое сервером приложений.

## Лекция 4 Многопоточные приложения

Параллельное программирование, связанное с использованием легковесных процессов, или подпроцессов (multithreading, light-weight processes) — концептуальная парадигма, в которой вы разделяете свою программу на два или несколько процессов, которые могут исполняться одновременно.

В системах без параллельных подпроцессов используется подход, называемый циклом обработки событий. В этой модели единственный подпроцесс выполняет бесконечный цикл, проверяя и обрабатывая возникающие события. Синхронизация между различными частями программы происходит в единственном цикле обработки событий. Такие среды называют синхронными управляемыми событиями системами. Apple Macintosh, Microsoft Windows, X11/Motif — все эти среды построены на модели с циклом обработки событий.

Если вы можете разделить свою задачу на независимо выполняющиеся подпроцессы и можете автоматически переключаться с одного подпроцесса, который ждет наступления события, на другой, которому есть чем заняться, за тот же промежуток времени вы выполните больше работы. Вероятность того, что больше чем одному из подпроцессов одновременно надолго потребуется процессор, мала.

Модель легковесных процессов в Java

Исполняющая система Java в многом зависит от использования подпроцессов, и все ее классовые библиотеки написаны с учетом особенностей программирования в условиях параллельного выполнения подпроцессов. Java использует подпроцессы для того, чтобы сделать среду программирования асинхронной. После того, как подпроцесс запущен, его выполнение можно временно приостановить (suspend). Если подпроцесс остановлен (stop), возобновить его выполнение невозможно. Приоритеты подпроцессов

Приоритеты подпроцессов — это просто целые числа в диапазоне от 1 до 10 и имеет смысл только соотношения приоритетов различных подпроцессов. Приоритеты же используются для того, чтобы решить, когда нужно остановить один подпроцесс и начать выполнение другого. Это называется *переключением контекста.* Правила просты. Подпроцесс может добровольно отдать управление — с помощью явного системного вызова или при

блокировании на операциях ввода-вывода, либо он может быть приостановлен принудительно. В первом случае проверяются все остальные подпроцессы, и управление передается тому из них, который готов к выполнению и имеет самый высокий приоритет. Во втором случае, низкоприоритетный подпроцесс, независимо от того, чем он занят, приостанавливается принудительно для того, чтобы начал выполняться подпроцесс с более высоким приоритетом.

**Синхронизация**

Поскольку подпроцессы вносят в ваши программы асинхронное поведение, должен существовать способ их синхронизации. Для этой цели в Java реализовано элегантное развитие старой модели синхронизации процессов с помощью *монитора.*

Сообщения

Коль скоро вы разделили свою программу на логические части - подпроцессы, вам нужно аккуратно определить, как эти части будут общаться друг с другом. Java предоставляет для этого удобное средство — два подпроцесса могут “общаться” друг с другом, используя методы wait и notify. Работать с параллельными подпроцессами в Java несложно. Язык предоставляет явный, тонко настраиваемый механизм управления созданием подпроцессов, переключения контекстов, приоритетов, синхронизации и обмена сообщениями между подпроцессами.

**Подпроцесс**

Класс Thread инкапсулирует все средства, которые могут вам потребоваться при работе с подпроцессами. При запуске Java-программы в ней уже есть один выполняющийся подпроцесс. Вы всегда можете выяснить, какой именно подпроцесс выполняется в данный момент, с помощью вызова статического метода Thread.currentThread. После того, как вы получите дескриптор подпроцесса, вы можете выполнять над этим подпроцессом различные операции даже в том случае, когда параллельные подпроцессы отсутствуют. В очередном нашем примере показано, как можно управлять выполняющимся в данный момент подпроцессом.

```
class CurrentThreadDemo {
public static void main(String args[]) {  Thread
t = Thread.currentThread(); t.setName("My
Thread");
System.out. println("current thread: " + t);
try { for (int n = 5; n > 0; n--) {
System.out.println(" " + n);
Thread.sleep(1000);
} }
catch (InterruptedException e) {
System.out.println("interrupted");
```

*} }*
*}*

В этом примере текущий подпроцесс хранится в локальной переменной t. Затем мы используем эту переменную для вызова метода setName, который изменяет внутреннее имя подпроцесса на “My Thread”, с тем, чтобы вывод программы был удобочитаемым. На следующем шаге мы входим в цикл, в котором ведется обратный отсчет от 5, причем на каждой итерации с помощью вызова метода Thread.sleep() делается пауза длительностью в 1 секунду. Аргументом для этого метода является значение временного интервала в миллисекундах, хотя системные часы на многих платформах не позволяют точно выдерживать интервалы короче 10 миллисекунд. Обратите внимание — цикл заключен в try/catch блок. Дело в том, что метод Thread.sleep() может возбуждать исключение InterruptedException. Это исключение возбуждается в том случае, если какому-либо другому подпроцессу понадобится прервать данный подпроцесс. В данном примере мы в такой ситуации просто выводим сообщение о перехвате исключения. Ниже приведен вывод этой программы:

*C:\> java CurrentThreadDemo*
*current thread: Thread[My Thread,5,main]*
*5*
*4*
*3*
*2*
*1*

Обратите внимание на то, что в текстовом представлении объекта Thread содержится заданное нами имя легковесного процесса — My Thread. Число 5 — это приоритет подпроцесса, оно соответствует приоритету по умолчанию, “main” — имя группы подпроцессов, к которой принадлежит данный подпроцесс. **Runnable**

Не очень интересно работать только с одним подпроцессом, а как можно создать еще один? Для этого нам понадобится другой экземпляр класса Thread. При создании нового объекта Thread ему нужно указать, какой программный код он должен выполнять. Вы можете запустить подпроцесс с помощью любого объекта, реализующего интерфейс Runnable. Для того, чтобы реализовать этот интерфейс, класс должен предоставить определение метода run. Ниже приведен пример, в котором создается новый подпроцесс.

*class ThreadDemo implements Runnable {*
*ThreadDemo() {*
*Thread ct = Thread.currentThread();*
*System.out.println("currentThread: " + ct);*
*Thread t = new Thread(this, "Demo Thread");*
*System.out.println("Thread created: " + t); t.start();*
*try {*

```
Thread.sleep(3000);
}
catch (InterruptedException e) {
System.out.println("interrupted");
}
System.out.println("exiting main thread");
}
public void run() {
try { for (int i = 5; i > 0;
i--) {
System.out.println("" + i);
Thread.sleep(1000);
} }
catch (InterruptedException e) {
System.out.println("child interrupted");
}
System.out.println("exiting child thread"); }
public static void main(String args[]) {
new ThreadDemo();
} }
```

Обратите внимание на то, что цикл внутри метода run выглядит точно так же, как и в предыдущем примере, только на этот раз он выполняется в другом подпроцессе. Подпроцесс main с помощью оператора new Thread(this, "Demo Thread") создает новый объект класса Thread, причем первый параметр конструктора — this — указывает, что нам хочется вызвать метод run текущего объекта. Затем мы вызываем метод start, который запускает подпроцесс, выполняющий метод run. После этого основной подпроцесс (main) переводится в состояние ожидания на три секунды, затем выводит сообщение и завершает работу. Второй подпроцесс — “Demo Thread” — при этом по-прежнему выполняет итерации в цикле метода run до тех пор пока значение счетчика цикла не уменьшится до нуля. Ниже показано, как выглядит результат работы этой программы этой программы после того, как она отработает 5 секунд.

```
C:\> java ThreadDemo
Thread created: Thread[Demo Thread,5,main]
5
4 3
exiting main thread
2 1
exiting child thread
```

Приоритеты подпроцессов

Если вы хотите добиться от Java предсказуемого независимого от платформы поведения, вам следует проектировать свои подпроцессы таким образом, чтобы они по своей воле освобождали процессор. Ниже приведен пример с двумя подпроцессами с различными приоритетами, которые не ведут себя одинаково на различных платформах. Приоритет одного из подпроцессов с помощью вызова setPriority устанавливается на два уровня выше Thread. NORM_PRIORITY, то есть, умалчиваемого приоритета. У другого подпроцесса приоритет, наоборот, на два уровня ниже. Оба этих подпроцесса запускаются и работают в течение 10 секунд. Каждый из них выполняет цикл, в котором увеличивается значение переменной-счетчика. Через десять секунд после их запуска основной подпроцесс останавливает их работу, присваивая условию завершения цикла while значение true и выводит значения счетчиков, показывающих, сколько итераций цикла успел выполнить каждый из подпроцессов.

```
class Clicker implements Runnable {
int click = 0; private Thread t;
private boolean running = true;
public clicker(int p) {
t = new Thread(this);
t.setPriority(p); }
public void run() { while
(running) {
click++;
} }
public void stop() {
running = false; } public
void start() { t.start();
} } class
HiLoPri {
public static void main(String args[]) {
Thread.currentThread().setPriority(Thread.MAX_PRIORITY); clicker
hi = new clicker(Thread.NORM_PRIORITY + 2);
clicker lo = new clicker(Thread.NORM_PRIORITY - 2);
lo.start(); hi.start(); try Thread.sleep(-10000) {
}
catch (Exception e) {
} lo.stop();
hi.stop();
System.out.println(lo.click + " vs. " + hi.click);
} }
```

По значениям, фигурирующим в распечатке, можно заключить, что подпроцессу с низким приоритетом достается меньше на 25 процентов времени процессора:

*C:\>java HiLoPri 304300*

*vs. 4066666*

Синхронизация

Когда двум или более подпроцессам требуется параллельный доступ к одним и тем же данным (иначе говоря, к совместно используемому ресурсу), нужно позаботиться о том, чтобы в каждый конкретный момент времени доступ к этим данным предоставлялся только одному из подпроцессов. Java для такой синхронизации предоставляет уникальную, встроенную в язык программирования поддержку. В других системах с параллельными подпроцессами существует понятие *монитора.* Монитор — это объект, используемый как защелка. Только один из подпроцессов может в данный момент времени владеть монитором. Когда под-процесс получает эту защелку, говорят, что он *вошел* в монитор. Все остальные подпроцессы, пытающиеся войти в тот же монитор, будут заморожены до тех пор пока подпроцесс-владелец не выйдет из монитора.

У каждого Java-объекта есть связанный с ним неявный монитор, а для того, чтобы войти в него, надо вызвать метод этого объекта, отмеченный ключевым словом synchronized. Для того, чтобы выйти из монитора и тем самым передать управление объектом другому подпроцессу, владелец монитора должен всего лишь вернуться из синхронизованного метода.

```
class Callme { void call(String
msg) { System.out.println("["
+ msg); try Thread.sleep(-
1000) {}  catch(Exception e)
{}
System.out.println("]");
} }
class Caller implements Runnable {
String msg; Callme
target;
public Caller(Callme t, String s) {
target = t; msg = s; new
Thread(this).start();
}
public void run() {
target.call(msg);
} }
class Synch { public static void
main(String args[]) {  Callme target =
```

```
new Callme(); new Caller(target,
"Hello."); new Caller(target,
"Synchronized");
new Caller(target, "World");
}
}
```

Вы можете видеть из приведенного ниже результата работы программы, что sleep в методе call приводит к переключению контекста между подпроцессами, так что вывод наших 3 строк-сообщений перемешивается:

```
[Hello.
[Synchronized
]
[World
] ]
```

Это происходит потому, что в нашем примере нет ничего, способного помешать разным подпроцессам вызывать одновременно один и тот же метод одного и того же объекта. Для такой ситуации есть даже специальный термин — race condition (состояние гонки), означающий, что различные подпроцессы пытаются опередить друг друга, чтобы завершить выполнение одного и того же метода. В этом примере для того, чтобы это состояние было очевидным и повторяемым, использован вызов sleep. В реальных же ситуациях это состояние, как правило, трудноуловимо, поскольку непонятно, где именно происходит переключение контекста, и этот эффект менее заметен и не всегда воспроизводятся от запуска к запуску программы. Так что если у вас есть метод (или целая группа методов), который манипулирует внутренним состоянием объекта, используемого в программе с параллельными подпроцессами, во избежание состояния гонки вам следует использовать в его заголовке ключевое слово synchronized.

Взаимодействие подпроцессов

В Java имеется элегантный механизм общения между подпроцессами, основанный на методах wait, notify и notifyAll. Эти методы реализованы, как final-методы класса Object, так что они имеются в любом Java-классе. Все эти методы должны вызываться только из синхронизованных методов. Правила использования этих методов очень просты:

- wait — приводит к тому, что текущий подпроцесс отдает управление и переходит в режим ожидания — до тех пор пока другой под-процесс не вызовет метод notify с тем же объектом.
- notify — выводит из состояния ожидания первый из подпроцессов, вызвавших wait с данным объектом.
- notifyAll — выводит из состояния ожидания все подпроцессы, вызвавшие wait с данным объектом.

Ниже приведен пример программы с наивной реализацией проблемы поставщик-потребитель. Эта программа состоит из четырех простых классов: класса Q, представляющего собой нашу реализацию очереди, доступ к которой мы пытаемся синхронизовать; поставщика (класс Producer), выполняющегося в отдельном подпроцессе и помещающего данные в очередь; потребителя (класс Consumer), тоже представляющего собой подпроцесс и извлекающего данные из очереди; и, наконец, крохотного класса PC, который создает по одному объекту каждого из перечисленных классов.

```
class Q {  int
n;
synchronized int get() {
System.out.println("Got: " + n); return
n;
}
synchronized void put(int n) {  this.n
= n;
System.out. println("Put: " + n);
} }
class Producer implements Runnable {
Q q;
Producer(Q q) {  this.q
= q;
new Thread(this,         "Producer").start();
}
public void run()        {
int i = 0; while
(true) {
q.put(i++);
} } }
class Consumer implements Runnable {
Q q;
Consumer(Q q) {
this.q = q; new Thread(this,
     "Consumer").start();
}
public void run()        {
while (true) { q.get();
}
} } class PC { public static void
main(String args[]) { Q q = new Q();
new Producer(q);
new Consumer(q);
```

*} }*

Хотя методы put и get класса Q синхронизованы, в нашем примере нет ничего, что бы могло помешать поставщику переписывать данные по того, как их получит потребитель, и наоборот, потребителю ничего не мешает многократно считывать одни и те же данные. Так что вывод программы содержит вовсе не ту последовательность сообщений, которую нам бы хотелось иметь:

C:\> java PC
*Put: 1*
*Got: 1*
*Got: 1*
*Got: 1*
*Got: 1*
*Got: 1*
*Put: 2*
*Put: 3*
*Put: 4*
*Put: 5*
*Put: 6*
*Put: 7*
*Got: 7*

Как видите, после того, как поставщик помещает в переменную n значение 1, потребитель начинает работать и извлекает это значение 5 раз подряд. Положение можно исправить, если поставщик будет при занесении нового значения устанавливать флаг, например, заносить в логическую переменную значение true, после чего будет в цикле проверять ее значение до тех пор пока поставщик не обработает данные и не сбросит флаг в false.

Правильным путем для получения того же результата в Java является использование вызовов wait и notify для передачи сигналов в обоих направлениях. Внутри метода get мы ждем (вызов wait), пока Producer не известит нас (notify), что для нас готова очередная порция данных. После того, как мы обработаем эти данные в методе get, мы извещаем объект класса Producer (снова вызов notify) о том, что он может передавать следующую порцию данных. Соответственно, внутри метода put, мы ждем (wait), пока Consumer не обработает данные, затем мы передаем новые данные и извещаем (notify) об этом объект-потребитель. Ниже приведен переписанный указанным образом класс Q.

*class Q {*
*int n;*
*boolean valueSet = false;*
*synchronized int get() { if*
*(!valueSet) try wait();*

```
catch(InterruptedException e):
System.out.println("Got: " + n);
valueSet = false;
notify(); return
n;
}
synchronized void put(int n) {
if (valueSet) try wait();
catch(InterruptedException e); this.n =
n; valueSet = true;
System.out.println("Put: " + n);
notify(); }
}
```

А вот и результат работы этой программы, ясно показывающий, что синхронизация достигнута.

```
C:\> java Pcsynch
Put: 1
Got: 1
Put: 2
Got: 2
Put: 3
Got: 3
Put: 4
Got: 4
Put: 5 Got:
5
```

Клинч (deadlock)

Клинч — редкая, но очень трудноуловимая ошибка, при которой между двумя легковесными процессами существует кольцевая зависимость от пары синхронизированных объектов. Например, если один подпроцесс получает управление объектом X, а другой — объектом Y, после чего X пытается вызвать любой синхронизированный метод Y, этот вызов, естественно блокируется. Если при этом и Y попытается вызвать синхронизированный метод X, то программа с такой структурой подпроцессов окажется заблокированной навсегда. В самом деле, ведь для того, чтобы один из подпроцессов захватил нужный ему объект, ему нужно снять свою блокировку, чтобы второй подпроцесс мог завершить работу.

Сводка функций программного интерфейса легковесных процессов

Ниже приведена сводка всех методов класса Thread, обсуждавшихся в этой лекции.

Методы класса

Методы класса — это статические методы, которые можно вызывать непосредственно с именем класса Thread.

currentThread

Статический метод currentThread возвращает объект Thread, выполняющийся в данный момент. yield

Вызов метода yield приводит к тому, что исполняющая система переключает контекст с текущего на следующий доступный подпроцесс. Это один из способов гарантировать, что низкоприоритетные подпроцессы когданибудь получат управление. sleep(int n)

При вызове метода sleep исполняющая система блокирует текущий подпроцесс на n миллисекунд. После того, как этот интервал времени закончится, подпроцесс снова будет способен выполняться. В большинстве исполняющих систем Java системные часы не позволяют точно выдерживать паузы короче, чем 10 миллисекунд. Методы объекта

start

Метод start говорит исполняющей системе Java, что необходимо создать системный контекст подпроцесса и запустить этот подпроцесс. После вызова этого метода в новом контексте будет вызван метод run вновь созданного подпроцесса. Вам нужно помнить о том, что метод start с данным объектом можно вызвать только один раз.

run

Метод run — это тело выполняющегося подпроцесса. Это — единственный метод интерфейса Runnable. Он вызывается из метода start после того, как исполняющая среда выполнит необходимые операции по инициализации нового подпроцесса. Если происходит возврат из метода run, текущий подпроцесс останавливается. stop

Вызов метода stop приводит к немедленной остановке подпроцесса. Это — способ мгновенно прекратить выполнение текущего подпроцесса, особенно если метод выполняется в текущем подпроцессе. В таком случае строка, следующая за вызовом метода stop, никогда не выполняется, поскольку контекст подпроцесса “умирает” до того, как метод stop возвратит управление. Более аккуратный способ остановить выполнение подпроцесса — установить значение какой-либо переменной-флага, предусмотрев в методе run код, который, проверив состояние флага, завершил бы выполнение подпроцесса. suspend

Метод suspend отличается от метода stop тем, что метод приостанавливает выполнение подпроцесса, не разрушая при этом его системный контекст. Если выполнение подпроцесса приостановлено вызовом suspend, вы можете снова активизировать этот подпроцесс, вызвав метод resume.

resume

Метод resume используется для активизации подпроцесса, приостановленного вызовом suspend. При этом не гарантируется, что после

вызова resume подпроцесс немедленно начнет выполняться, поскольку в этот момент может выполняться другой более высокоприоритетный процесс. Вызов resume лишь делает подпроцесс способным выполняться, а то, когда ему будет передано управление, решит планировщик. setPriority(int p)

Метод setPriority устанавливает приоритет подпроцесса, задаваемый целым значением передаваемого методу параметра. В классе Thread есть несколько предопределенных приоритетов-констант: MIN_PRIORITY, NORM_PRIORITY и MAX_PRIORITY, соответствующих соответственно значениям 1, 5 и 10. Большинство пользовательских приложений должно выполняться на уровне NORM_PRIORITY плюс-минус 1. Приоритет фоновых заданий, например, сетевого ввода-вывода или перерисовки экрана, следует устанавливать в MIN_PRIORITY. Запуск подпроцессов на уровне MAX_PRIORITY требует осторожности. Если в подпроцессах с таким уровнем приоритета отсутствуют вызовы sleep или yield, может оказаться, что вся исполняющая система Java перестанет реагировать на внешние раздражители. SetPriority

Этот метод возвращает текущий приоритет подпроцесса — целое значение в диапазоне от 1 до 10. setName(String name)

Метод setName присваивает подпроцессу указанное в параметре имя. Это помогает при отладке программ с параллельными подпроцессами. Присвоенное с помощью setName имя будет появляться во всех трассировках стека, которые выводятся при получении интерпретатором неперехваченного исключения. getName

Метод getName возвращает строку с именем подпроцесса, установленным с помощью вызова setName.

Есть еще множество функций и несколько классов, например, ThreadGroup и SecurityManager, которые имеют отношение к подпроцессам, но эти области в Java проработаны еще не до конца. Скажем лишь, что при необходимости можно получить информацию об этих интерфейсах из документации по JDK API.

А дорога дальше вьется

Простые в использовании встроенные в исполняющую среду и в синтаксис Java легковесные процессы — одна из наиболее веских причин, по которым стоит изучать этот язык. Освоив однажды параллельное программирование, вы уже никогда не захотите возвращаться назад к программированию с помощью модели, управляемой событиями. После того, как вы освоились с основами программирования на Java, включая создание классов, пакетов, и модель легковесных процессов, для вас не составит труда разобраться в той коллекции Java-классов, к обсуждению которой мы сейчас приступим.

# РАЗДЕЛ 3 РАЗРАБОТКА WEB-СИСТЕМ НА ОСНОВЕ ТЕХНОЛОГИИ JAVA

## Лекция 5 Сервлеты

Главное предназначение сервлетов заключается в увеличении эффективности серверных приложений путем "экономного" использования системных ресурсов.

**Сервлет Java** — это Java-приложение, выполняющееся, как и программы CGI, на сервере. Единственное отличие сервлетов от CGIпрограмм заключается в том, что они выполняются с помощью **специальной сервлет-машины** (нечто наподобие виртуальной машины Java), за счет которой как раз и удается достичь увеличения эффективности обработки пользовательских запросов. **Сервлет-запросы** полностью аналогичны запросам CGI. После получения такого запроса Web-сервер вместо запуска серверного приложения, что произошло бы при обработке запроса CGI, передает запрос на обработку сервлет-машине. После этого сервлет-машина, наконец-то, передает запрос обрабатывающему его сервлету.

Существует несколько самых известных сервлет-машин, которые распространяются в виде надстроек к популярным Web-серверам, таким как Apache. В качестве примера можно привести **машины JServ и JRun**. Как только сервлет-машина будет установлена, настроена и интегрирована с Web-сервером, можно начинать использование сервлетов в качестве серверных приложений, предназначенных для обработки запросов.

Сервлет-машина загружает класс в процесс **виртуальной машины Java (JVM)** и создает один экземпляр данного класса (который, естественно, представляет собой сервлет). Экземпляр сервлета может быть создан сразу же при запуске сервлет-машины или же после поступления первого пользовательского запроса к этому сервлету. С появлением каждого нового запроса создается отдельный поток, использующий один и тот же экземпляр сервлета. Таким образом, для обработки множественных запросов требуется всего лишь один экземпляр сервлета, что позволяет значительно уменьшить использование системных ресурсов. Поскольку один экземпляр сервлета используется для обслуживания различных Web-запросов, любые данные (глобальные) этого экземпляра будут одновременно доступны всем запросам. Если в классе сервлета требуется объявить переменные, значения которых предполагается изменять, то необходимым требованием к такому сервлету выступает обеспечение синхронизации. Что касается локальных переменных сервлета (переменных, объявленных внутри его методов), то, они, естественно, не становятся доступными всем запросам, а поэтому в них можно без опаски сохранять локальную информацию данного сервлета. Для лучшего понимания всего вышесказанного рассмотрим небольшой пример:

```
public class MyServlet extends HttpServlet {
int globalVar; public void myMethod() {
int localVar;
}
}
```

Значение переменной globalVar доступно всем потокам, обрабатывающим запросы, которые поступили к экземпляру сервлета MyServlet, поэтому, изменив это значение без предварительного обеспечения синхронизации, появляется риск повреждения данных. Однако поскольку переменная localVar объявлена внутри метода myMethod(), ее значение не используется одновременно различными потоками. Изменив ее в одном потоке, значение этой переменной не изменится ни в одном другом потоке, использующем данный экземпляр сервлета MyServlet.

**Строение сервлета**

Все сервлеты реализуют и используют интерфейсы и классы, объявленные в пакете javax.servlet. Помимо этого, все сервлеты должны "расширять" (ключевое слово extend в заголовке сервлета) класс Servlet, либо один из его потомков. Рассмотрим все сервлеты расширяющие специализированный именно для этих целей класс HttpServlet. Это связано с тем, что здесь рассматривается техника использования сервлетов для обработки HTTP-запросов.

В сервлете может быть объявлен метод init(). Данный метод вызывается всего лишь один раз за все "время жизни" сервлета. Это происходит либо при запуске сервлет-машины, либо при поступлении первого запроса к этому сервлету. В методе init() следует разместить все процедуры, которые необходимое выполнить при инициализации сервлета. Делать это, не обязательно, так как метод init() не является тем методом, который обязательно должен присутствовать в каждом сервлете. Очень похожим методом является метод destroy(), который вызывается после завершения работы сервлет-машины. В данном методе следует размещать команды, предназначенные для освобождения ресурсов, занятых сервлетом во время своей инициализации. Аналогично методу init(), метод destroy() вызывается только один раз за все время существования экземпляра сервлета.

Рассмотрим метод doPost(), который вызывается после поступления запроса типа POST (т.е. когда пользователь щелкнет на кнопке Submit формы, у которой значение атрибута METHOD равно POST). В теле данного метода следует разместить все команды, отвечающие за обработку пользовательских данных. В качестве команд могут, естественно, использоваться вызовы других

методов. Сервлет-машина передает методу doPost() в качестве входных параметров два объекта.

Объект HttpServletRequest. Он может быть использован для получения такой информации о запросе, как IP-адрес пользовательского компьютера, с которого был инициирован этот запрос, а также имена и значения всех переменных соответствующей пользовательской формы.

Объект HttpServletResponse. Его можно использовать для генерации ответа на пользовательский запрос.

Обычно в качестве ответа на запрос пользователь получает сгенерированную сервлетом (в общем случае, приложением, обрабатывающим запрос) страницу HTML, которая, в свою очередь, нередко содержит еще одну форму. Однако это не является правилом в ответ на запрос пользователь может получить все что угодно, т.е. файл произвольного типа. Одним из популярных форматов файлов, пересылающихся по Internet, является формат .PDF — стандартный формат программы Acrobat Reader. После получения такого файла на компьютере пользователя автоматически запускается данная программа, в которой сразу же открывается полученный файл. Для того чтобы пользователь смог узнать, какой тип ответа он получит от сервера, обрабатывающее запрос приложение должно непосредственно указать тип пересылаемых данных еще до отправки самого ответа. Чтобы сделать это, следует воспользоваться одним из методов класса HttpServletResponse.

Если бы вместо метода отправки информации POST был использован метод GET, то сервлет должен был бы реализовать метод doGet() вместо метода doPost().Оба метода похожи друг на друга, единственное отличие между ними заключается в их именах.

**Сервлет-машина JServ**

После установки и запуска сервлет-машины она готова к приему сервлет-запросов. Сервлет-запрос очень напоминает CGI-запрос. В некотором смысле эта аналогия распространяется и на настройку сервлетмашины. При настройке JServ должны быть определены каталоги сервлетов и соответствующие им псевдонимы (которые еще называются зонами). Зоны необходимо объявлять в конфигурационном файле JServ jserv.conf. Каждой зоне соответствует файл свойств, в котором указывается соответствующий этой зоне физический каталог сервера (каталог, в котором расположены файлы классов сервлетов). Приведем небольшой пример. Пусть сервлетмашина настроена таким образом, что в ней определена зона testZone с псевдонимом testZone. В этом случае в файле jserv.conf должна быть следующая строка:

ApJServMount /testZone /testZone

Также должен быть создан соответствующий данной зоне файл свойств testZone.properties. Одной из его строк будет примерно следующая:
repositors=/home/vsharma/SERVLETS/JSERV/Apache-JServ-1.Ob1/testZone

Таким образом, все сервлеты зоны testZone расположены в каталоге /home/vsharma/SERVLETS/JSERV/Apache-JServ-1.Ob1/testZone В пределах сервлет-машины можно определить сразу несколько зон. Это необходимо делать для разделения различных типов приложений по отдельным логическим областям. Все зоны должны быть определены в файле jserv.properties.

Еще раз обратимся к примеру. Из приведенных ниже строк можно сделать вывод, что данная сервлет-машина настроена с двумя зонами:

zones=example, testZone

testZone.properties=/local/vsharraa/SERVLET/JSERV/Apache-JServ-1.Ob1/testZone/ testZone.properties

Также отсюда можно узнать полный путь к файлу свойств testZone.properties зоны testZone.

Другим, не менее важным элементом файла jserv.properties является список каталогов и файлов с расширением .jar, которые должны быть включены в переменную CLASSPATH. (Как-никак, а сервлет-машина JServ запускает виртуальную машину Java, просматривающую переменную CLASSPATH для того, чтобы получить список каталогов, в которых хранятся классы Java.) Этот список определяется с помощью параметра wrapper, CLASSPATH. Для каждого каталога или .jar-файла, который необходимо включить в переменную CLASSPATH, должна быть отведена отдельная строка. Рассмотрим пример:

wrapper.classpath=/local/vsharma/SERVLETS/JSERV/Apache-JServ-1.Ob1/src/java/ Apache-JServ.jar

wrapper.classpath=/local/vsharma/SERVLETS/JSDK/JSDK2.0/lib/jsdk.jar

wrapper.classpath=/local/vsharma

wrapper.classpath=/local/vsharma/JAF/jaf/activation.jar

wrapper.classpath=/local/vsharma/JAVAMAIL/javamail-1.1/mail.jar

Для того чтобы внесенные изменения данного файла вступили в силу, после добавления новой строки wrapper.classpath или внесения каких-либо других изменений необходимо перезапустить сервлет-машину JServ. Чтобы сделать это, потребуется заодно перезапустить и сам Web-сервер Apache.

Теперь, учитывая приведенную выше конфигурацию сервлетмашины, посмотрим, что должно произойти, когда пользователь запросит информацию по адресу: http: X/myServer/testZone/MyServlet. Web-сервер воспримет данный запрос как обращение к сервлету, расположенному в зоне testZone. После этого сервер передаст запрос сервлет-машине, которая попытается выяснить,

существует ли экземпляр класса MyServer.class. Если он существует, сервлет-машина создаст поток, в котором будет выполняться данный экземпляр (если полученный запрос создан из пользовательской формы, поддерживающей метод передачи информации POST, то сразу же будет вызван метод doPost()). Если же такого экземпляра пока что не существует, то он будет незамедлительно создан, и первым вызванным методом будет метод init(). Затем, после выполнения всех инициализирующих команд, будет вызван метод doPost().

Сервлет-машина будет "искать" класс MyServlet в каталоге, который был указан в файле свойств testZone, properties зоны testZone. Это каталог

/home/vsharma/SERVLETS/JSERV/Apache-JServ-1.Ob1/testZone

**Создаем первый сервлет**

Ниже приведен исходный код сервлета, который, по предположению, находится в зоне testZone. Начнем, однако, с рассмотрения исходного кода документа HTML (файл firstServlet.html), содержащего пользовательскую форму.

```
<HTML>
<HEAD>
</HEAD> <BODY>
<FORM ACTION='http: //myServer/testZone,/FirstServlet' METHDO='POST ' >
<INPUT TYPE='submit' NAME='submit' VALUE='Execute'> </FORM> </BODY> </HTML>
```

Когда пользователь щелкнет на кнопке Submit, на сервере запускается следующий класс (сервлет). Файл FirstServlet. Java имеет такой вид:

```
import java.io.*; import javax.servlet.*;
import javax.servlet.http.*; public class
FirstServlet extends HttpServlet
{
public void doPost (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
{
 PrintWriter out;
res.setContentType("text/html");  out
= res.getWriter();
out .println ("<HTML><HEAD></HEAD>") ; out.println("<BODY
bgcolor=\"#FFFFFF\">");  out.println("<B>Первый сервлет
</B>");
```

```
out. println ("</BODY></HTML>" );
out.close ();
}
}
```

Для того чтобы протестировать работоспособность данного сервлета, для начала перенесите файл firstServlet.html в корневой каталог файлов Webсервера (в терминологии Web-сервера Apache — DocumentRoot). Откомпилируйте класс FirstServlet. Java (заметьте, что для успешной компиляции этого класса в переменную CLASSPATH обязательно должен быть включен файл jsdk.jar) и перенесите файл FirstServlet. class в каталог, соответствующий зоне testZone. После этого можно смело открыть в обозревателе Internet страницу firstServlet.html и щелкнуть на кнопке Execute (Выполнить).

Когда пользователь щелкнет на кнопке Execute, сервер запустит сервлет, обрабатывающий данные формы.

Рассмотрим исходный код сервлета FirstServlet. Вначале импортируются нескольких пакетов, необходимых для работы сервлета. Первым импортируется пакет java.io, так как ответ сервлета на поступивший запрос (страница HTML) будет отправлен на пользовательский компьютер через выходной поток. В пакетах Javaservlet и Java, servlet.http находятся все классы, использующиеся в данном сервлете. Класс FirstServlet.class реализует метод doPost(), который принимает два входных параметра: объект HttpServletRequest и объект HttpServletResponse. Этот класс является расширением класса HttpServlet. Заметьте, что все сервлеты должны реализовывать интерфейс Servlet. Класс HttpServlet является расширением класса GenericServlet, который, в свою очередь, реализует интерфейс Servlet.

Одной из самых важных операций, выполняемых в методе doPost(), является получение дескриптора выходного потока. Для этого используется метод getWriter() класса HttpServletResponse, который возвращает объект

```
PrintWriter.out = res.getWriter();
```

Все данные, записанные в этот объект, будут отправлены обратно обозревателю Internet пользовательского компьютера, с которого был передан запрос на сервер. Как упоминалось ранее, обязательным является указание типа информации, содержащейся в ответе. Поскольку в данном примере в ответ на пользовательский запрос сервер отсылает ответ в виде страницы HTML, тип содержимого должен быть установлен в text/html.

```
res.setContentType("text/html") ;
```

Все методы out.println() используются для вывода строк исходного кода HTML-документа, который будет отображен клиентским обозревателем Internet. С точки зрения языка HTML, сгенерированная сервером Webстраница

ничем не отличается от обычных Web-страниц, статически хранящихся на сервере.

**Переменные пользовательской формы**

Разобравшись с общим механизмом работы сервлета, рассмотрим способ, с помощью которого сервлет может извлечь данные пользовательской формы, переданные вместе с запросом. Для того чтобы это сделать, необходимо воспользоваться некоторыми достаточно полезными методами, объявленными в классе HttpServletRe-quest. Приведем небольшой пример. Предположим, чтопользовательская форма содержит текстовое поле ввода и набор переключателей. Рассматриваемый сервлет считывает значения всех переменных формы и отображает их вместе с именами переменных.

Исходный код документа HTML имеет следующий вид.

```
<HTML>
<HEAD>
</HEAD> <BODY>
<FORM ACTION='http://myServer/testZone/ShowFormVariables' METHOD='POST'>
<INPUT TYPE='text' NAME='firstName' VALUE="XBR> <INPUT
TYPE='radio’ NAME='rd' VALUE=' val1 "XBR>
<INPUT TYPE='radio' NAME='.rd' VALUE=' val2 '><BR>
<INPUT TYPE='submit' NAME='Submit' VALUE='Execute'> </FORM>
</BODY> </HTML>
```

Исходный код сервлета имеет такой вид.

```
import java.io.*; import java.util.*; import
javax.servlet.*; import javax.servlet.http.*; public
class ShowFormVariables extends HttpServlet
{
public void doPost (HttpServletRequest req, HttpServletResponse res)
throws ServletException, lOException
{
PrintWriter out;
res.setContentType("text/html"); out
= res.getWriter();
out.println("<HTML><HEAD></HEAD>");
out.println("<BODY bgcolor=\"#FFFFFF\">"); String
txt = req.getParameterValues("firstName")[0];
out.println("Значение текстового поля ввода, firstName, равно: " + txt);
out.println("<BR>А вот все переменные формы и их значения<BR>");
Enumeration names = req.getParameterNames();
```

```
while(names.hasMoreElements()) {
        String name = (String)names.nextElement();  String
        value = req.getParameterValues(name)[0];
        out.println("Имя: " + name + "Значение: " + value + "<BR>"); }
        out. println ("</BODY></HTML>" );  out.close
        (); } }
```

В классе HttpServletRequest есть метод getParameterValues() , который возвращает значение любой переменной формы по имени этой переменной. Возвращаемый результат представляет собой массив строк String. Почему именно массив строк, а не просто отдельную строку? Все дело в том, что некоторые переменные формы могут иметь несколько значений, например, те переменные, что соответствуют группам флажков. Представим, что форма содержит четыре флажка, объединенных в одну группу (напомним, что в этом случае они все имеют одинаковое имя). Если пользователь выберет, скажем, три флажка и щелкнет на кнопке Submit, то на сервер будет отправлено три значения, соответствующих одной переменной. Естественно, что самым удобным способом представления таких данных является массив. Если же переменной формы соответствует всего лишь одно значение, к нему можно достаточно просто обратиться как к первому элементу массива, возвращаемого методом getParameterValues().

Именно таким образом извлекается значение переменной firstName:

```
String txt = req.getParameterValues("firstName")[0];
```

В данном классе приведен также альтернативный метод доступа к переменным формы, который не предполагает предварительного знания их имен. Метод getParameterNames() возвращает объект Enumeration, в котором содержатся все имена переменных формы (включая кнопку Submit).

```
Enumeration names = req.getParameterNames();
```

С помощью этого объекта можно получить значения всех имен переменных формы. Каждое имя хранится в виде объекта String. Передав имя переменной в качестве входного параметра методу getParameterValues(), в ответ можно получить значение данной переменной.

Результат работы такого сервлета выглядит примерно так:

Значение текстового поля ввода, firstName, равно: Это текстовое поле
А вот все переменные формы и их значения

Имя: rd Значение: val2 Имя: Submit Значение: Execute Имя: firstName Значение: Это текстовое поле

В данном случае перед тем, как щелкнуть на кнопке Submit, пользователь ввел в текстовом поле ввода строку Это текстовое поле и активизировал второй переключатель (первый переключатель при этом автоматически был снят).

**Работаем с cookies**

По умолчанию каждый пользовательский запрос на получение Webстраницы или обработку данных формы не зависит от предыдущих запросов, сделанных этим же пользователем. С одной стороны, это, может быть, и хорошо, однако, с другой стороны, это накладывает строгие ограничения на процесс накопления информации, который происходит по мере посещения пользователем данного Web-узла. Одним из классических примеров, характеризующих эту проблему, является пример ввода пользователем своего имени при регистрации на Web-узле. Если пользователь затем вызовет какой-то другой сервлет, то он не будет иметь никаких средств, с помощью которых можно было бы определить имя пользователя, введенное ранее при входе на Web-узел. Таким образом, второй сервлет не сможет, например, поприветствовать пользователя, что является, в общем-то, стандартной практикой. Для того чтобы обойти это ограничение, была разработана концепция файлов cookie. В принципе это не единственный подход к решению данной проблемы, однако вторая технология — технология сеансов (Sessions) — будет рассмотрена далее в главе.

Файл cookie — это файл небольшого размера, который создается серверным приложением и размещается на компьютере пользователя. Когда пользователь обращается к Web-узлу, серверное приложение которого создало файл cookie, этот файл автоматически пересылается данному Webсерверу. Обратимся вновь к примеру с двумя сервлетами. Первый сервлет может создать файл cookie, сохранив в нем такие данные о пользователе, как его имя, фамилию и т.п. Второй (как, впрочем, и любой следующий по времени вызова) сервлет, в свою очередь, может обратиться к только что созданному файлу cookie, считать из него всю информацию и отобразить какое-либо "персонализированное" сообщение пользователю.

Однако, прежде чем перейти к более подробному рассмотрению технологии файлов cookie, следует сказать несколько слов об ограничениях, которые, похоже, не обходят уже ни одно явление в компьютерном мире. Большинство обозревателей Internet накладывают ограничения на размер файла cookie и общее количество cookie, которые могут быть установлены на пользовательском компьютере приложениями одного Web-сервера. Таким образом, вполне понятно, что при всем желании в файле cookie нельзя сохранить большой объем информации. Если все же без этого не обойтись, то в таком случае придется воспользоваться технологией сеансов, которая будет рассмотрена в следующих разделах данной главы.

В приведенном ниже примере рассматривается механизм установки файла cookie на пользовательском компьютере и способ его чтения серверным приложением. Чтобы протестировать соответствующие сервлеты, для начала

необходимо переместить файл cookieExample.html в корневой каталог Web-сервера (DocumentRoot). Затем следует откомпилировать классы SetCookie и GetCookie и переместить их в зону example.

Открыв файл cookieExample.html с помощью обозревателя Internet, пользователь увидит форму, в которой ему необходимо заполнить два поля: поле с именем и поле с фамилией пользователя. После щелчка на кнопке Submit вызывается сервлет SetCookie. В функции данного сервлета входит чтение информации, введенной пользователем в форму, и ее запись в файл cookie с именем MYSITE. Файл cookie имеет название и хранящееся в этом файле значение. Поскольку в одном файле cookie можно хранить лишь одно значение, то здесь возникает маленькая проблема, так как в данном случае нам необходимо записать в cookie как имя пользователя, так и его фамилию. Эта проблема легко решается путем использования объекта String, в который записываются оба эти значения, разделенные некоторым специальным символом. Символ-разделитель используется затем серверным приложением для того, чтобы отделить имя пользователя от его фамилии.

Отработав, сервлет SetCookie передает управление другому сервлету — сервлету GetCookie. Данный сервлет считывает значение файла cookie MYSITE и выделяет из него имя и фамилию пользователя. Затем он генерирует страницу HTML, помещает туда эту информацию (а также всю информацию, которая находится в остальных файлах cookie, полученных от данного клиента) и возвращает ее пользователю в качестве ответа.

Файл cookieExample.html имеет следующий вид.

```
<HTML><HEAD></HEAD>
<BODY>
```

Данный пример наглядно демонстрирует процесс установки и чтения файлов<BP> cookie с использованием для этой цели сервлетов.<BR>

Первый сервлет сохраняет в файле cookie информацию об имени и фамилии<BR> пользователя, второй же сервлет используется для чтения-этих значений.<BR>

```
<FORM ACTION='/example/SetCookie' METHOD='POST'>
<TABLE>
<TR><TD>Имя</TD>
<TD><INPUT TYPE='text' NAME='firstName' SIZE=' 10 ' > </TD></TR>
<TR><TD>Фамилия</TD>
<TD><INPUT TYPE='text' NAME='lastName' SIZE=' 10 ' > </TD></TR>
<TR> <TD> </TD>
<TD> INPUT TYPE='submit' NAME='submit' VALUE= ' Ok' > </TD> </TR>
</TABLE></FORM></BODY></HTML>
```

Файл SetCookie.java имеет такой вид:

```
Import java.util.*;  import java.io.*;  import
javax.servlet.*;  import javax.servlet.http.*;
public class SetCookie extends HttpServlet {
String cookieName = "MYSITE";
String cookieDomain = "YOUR_DOMAIN"; String
delim = "DLM:";
int maxAge = 3600; // 3600 секунд = 1 час String
redirectUrl = "/example/GetCookie";
public void doGet (HttpServletRequest req, HttpServletResponse res)
{
doPost(req, res);
}
public void doPost (HttpServletRequest req, HttpServletResponse res)
{ try{
String firstName = "", lastName = "";
Enumeration values = req. getParameterNames ()•;
while(values.hasMoreElements ()) {
String name = (String)values.nextElement();  String value
= req.getParameterValues(name)[0];
if(name.equalsIgnoreCase("firstName")) firstName =
value; if(name.equalsIgnoreCase("lastName")) lastName =
value; }
String cookieVal = firstName + delim + lastName; Cookie ck = new
Cookie(cookieName, cookieVal); ck.setDomain(cookieDomain);
ck.setMaxAge(maxAge);  ck.setPath("/");  res.addCookie(ck);
res.sendRedirect(redirectUrl); }
catch(Exception ex){} } }
```

В состав пакета разработчика сервлетов Java (JSDK) входит класс Cookie, который, как несложно догадаться по его названию, служит для представления файлов cookie. С помощью данного класса можно создать новый файл cookie, а затем отправить его на пользовательский компьютер вместе с ответом на запрос. По большому счету, файл cookie имеет всего лишь два параметра, которые можно изменять — это имя и значение. Помимо этого, с помощью метода setMaxAge() можно определить так называемый "срок жизни" файла cookie. Что же представляет собой этот срок? Это время в секундах, отсчитываемое от момента создания файла cookie до того момента, когда информация, хранящаяся в нем, "морально" устаревает, т.е. становится непригодной.

Также для файла cookie можно определить имя домена и путь. Имя домена определяет группу серверов, которым пользовательский компьютер должен переслать файл cookie вместе с запросом. По умолчанию файл cookie отправляется на тот сервер, приложением которого он был установлен. Имя домена полезно тогда, когда файл cookie должен передаваться на несколько серверов, принадлежащих одному и тому же домену. Приведем небольшой пример. Пусть несколько серверов принадлежат домену xyz. com. Предположим, что имена этих серверов имеют вид abc.xyz.com, def.xyz.com и т.д. Если файл cookie был установлен приложением сервера def.xyz.com, и необходимо обеспечить его рассылку любому серверу данного домена (abc.xyz.com, def.xyz.com и т.д.), к которому только поступит пользовательский запрос, то для достижения этой цели следует воспользоваться методом setDomain(), как показано ниже:

```
cookie.setDomain(".xyz.com");
```

Практически аналогичным образом можно определить, чтобы файл cookie пересылался только тогда, когда пользователь запрашивает адрес URL, принадлежащий какой-то определенной части Web-узла. По умолчанию, файл cookie пересылается только тем приложениям или Webстраницам, которые расположены в каталоге, где находится приложение или страница, создавшие данный файл. Для того чтобы сделать файл cookie доступным для всех приложений данного Web-узла, необходимо воспользоваться методом setPath(). Приведенная ниже строка кода определяет, что файл cookie будет доступен всем приложениям или страницам Web-узла:

```
cookie.setPath("/");
```

А вот небольшой фрагмент кода из файла SetCookie.Java, в котором непосредственно производится создание и установка файл cookie:

```
//Создание нового файла cookie с определенным именем и значением
1
Cookie ck = new Cookie(cookieName, cookieVal); //Указание
имени домена, срока использования и пути
ck.setDomain(cookieDomain); 1 ck.setMaxAge(maxAge);
ck.setPathCV");
//Добавление файла cookie к ответу, отправляемому на компьютер
пользователя  res.addCookie(ck);
```

Одной из самых последних команд сервлета SetCookie является вызов метода sendRedirect() класса HttpServletResponse. Данный метод используется для того, чтобы вместо стандартного ответа в виде страницы HTML (или чего-то там еще) пользовательский обозреватель Internet получил команду сделать переход на другой адрес URL. В рассматриваемом случае обозреватель клиентского компьютера получает команду перейти по адресу /example/GetCookie, т.е. обратиться к еще одному сервлету — сервлету GetCookie.

```
res.sendRedirect(redirectUrl);
Файл GetCookie.Java имеет следующий вид.
import javax.servlet.*; import
javax.servlet.http.*; import java.io.*; public
class GetCookie extends HttpServlet
{
String cookieName = "MYSITE"; String
delim = "DLM:";
public void doGet (HttpSevletRequest req, HttpServletResponse res)
{
doPost(req, res); } public void doPost (HttpSevletRequest req,
HttpServletResponse res) { try{
PrintWriter out = null; res.setContentType("text/html")
; out = res.getWriter();
Cookie[] cookieList = res.getCookies () ;
int size = cookieList.length; for(int i=0;
i<size; i++)
{
Cookie c = cookieList [i] ;
String name = c.getName (); String value =
c.getValue(); out.printIn("Имя cookie " +
name + "<BR>");
if(name.equals(cookieName)) int ind =
value.indexOf(delim);
String firstName = value.substring(0, ind);
String lastName = value.substring(ind + delim.length());
out.println("Значения, сохраненные в данном файле cookie:<BR>");
out.println("Имя: " + firstName + "Фамилия: " + lastName);
} }
out.close(); }catch(Exception ex){} } }
```

Запрос к данному сервлету может быть выполнен для списка файлов cookie, переданных серверу клиентским компьютером. Метод getCookies() возвращает массив, элементами которого являются файлы cookie. Для того чтобы узнать имена и значения файла cookie, следует пройтись по всем элементам полученного массива, используя при этом такие методы класса Cookie, как getName() и getValue() (возвращающие, соответственно, имя и значение). Сервлет GetCookie возвращает страницу HTML, на которой отображаются имена всех файлов cookie, переданных серверу. Когда сервлет находит файл cookie с именем MYSITE, он обрабатывает его значение, выделяя из него имя и фамилию пользователя (для чего применяется

символразделитель, используемый в сервлете SetCookie) и отображая эти значения на странице.

### Сеансы

Итак, мы подошли к последнему разделу данной главы, в котором рассматриваются принципиально важные моменты, касающиеся темы сервлетов, — сеансы. Web представляет собой статичную систему. По умолчанию Web-сервер не обеспечивает поддержку сеансовых данных, что, естественно, является большим недостатком для приложений, которые, как правило, работают с несколькими открытыми окнами. Представим себе следующую ситуацию. Пусть есть некоторое приложение, использующее для своей работы три окна. В каждом из этих окон находятся поля ввода, которые должны быть заполнены прежде, чем данные приложения будут отправлены на обработку. Для того чтобы отправить информацию на обработку, необходимо щелкнуть на кнопке Submit, расположенной в третьем окне приложения. После этого все данные, введенные в первых двух окнах, должны быть переданы в третье окно с помощью скрытых переменных, и только затем их следует отправить на сервер (где они, предположительно, заносятся в некую базу данных). Подобный процесс передачи введенных пользователем данных является слишком громоздким и неуклюжим, так как он требует практически постоянной и ничем не обоснованной пересылки информации. Намного лучшим решением в данной ситуации представляется способ, при котором вся вводимая информация хранится некоторое определенное время на самом сервере. Подобный подход реализован в пакете JSDK в виде интерфейса HttpSession.

Сеанс представляет собой соединение между клиентом и сервером, устанавливаемое на определенное время, за которое пользователь может отправить на сервер сколько угодно запросов. Другими словами сеанс — это такой вид соединения, который в принципе не поддерживается Webсервером по умолчанию. Как правило, сеансы используются для обеспечения хранения данных во время нескольких запросов на получение Web-страницы или на обработку информации, введенной в пользовательскую форму. Самым распространенным методом поддержки сеанса является метод, в котором используются файлы cookie. В этом случае практически любое приложение может подключиться к сеансу и получить принадлежащие ему данные (которые затем это же приложение может и изменить).

Для того чтобы открыть новый сеанс, следует воспользоваться методом getSession() класса HttpServletRequest. В качестве входного параметра этот метод принимает булево значение; если оно равно true, то сервлет-машина проверяет наличие активного сеанса, установленного с данным клиентом. В случае успеха метод getSessionO возвращает дескриптор этого сеанса. В

противном случае метод устанавливает новый сеанс. Выглядит все это примерно следующим образом:

HttpSession session = request.getSession(true);

После установки или подключения к сеансу можно использовать все методы, объявленные в интерфейсе HttpSession, для записи или чтения данных, принадлежащих этому сеансу. Обратите внимание, что это можно делать как из сервлета, непосредственно инициировавшего подключение к сеансу, так и из любого другого сервлета, выполняющегося в рамках данной сервлет-машины.

Для того чтобы сохранить значение переменной в текущем сеансе, следует воспользоваться методом putValue(); для того, чтобы прочесть значение некоторой переменной текущего сеанса, следует воспользоваться методом getValue(). Например, если необходимо сохранить для использования в будущем одну из переменных сеанса, представляющую собой целое число, следует добавить следующий код:

Integer i = new Integer(22); session.putValue("myinteger",
i);

Данной переменной в контексте текущего сеанса присваивается имя myinteger, для сохранения ее значения вызывается упомянутый ранее метод putValue(). После этого любой подключившийся к текущему сеансу сервлет сможет прочесть значение переменной myinteger с помощью следующей команды:

Integer theVal = (Integer) session.getValue("myinteger");

Если же переменной с таким именем (myinteger) в текущем сеансе не существует, метод getValue() возвратит значение null.

Для того чтобы удалить переменную из текущего сеанса, следует воспользоваться методом removeValue(name). Список имен всех переменных, сохраненных в текущем сеансе, можно получить с помощью вызова метода String[] getValueNames()

Для того чтобы завершить сеанс, следует вызвать метод invalidate(). В результате следующий вызов метода getSession(true) приведет к созданию нового сеанса, а данные, сохраненные в старом сеансе, будут навсегда утеряны для всех сервлетов.

Итак, теперь пришло время рассмотреть один небольшой пример использования технологии сеансов. Щелкнув на кнопке Submit, пользователь тем самым осуществит первый вызов сервлета SessionServlet, который иллюстрирует простейшие основы работы с сеансом. Ниже приведен исходный код страницы HTML, на которой размещена соответствующая форма.

<HTML>
<HEAD> </HEAD> <BODY>
<FORM ACTION='http://myServer/testZone/SessionServlet'

METHOD='POST'>

<INPUT TYPE='submit' NAME='Submit' VALUED='Execute'>

</FORM> </BODY> </HTML>

В сервлете SessionServlet создается сеанс с клиентским компьютером (на базе интерфейса HttpSession). Для того чтобы установить сеанс, используется метод getSession(true). Параметр true, как уже упоминалось выше, означает подключение к текущему сеансу, если таковой существует (при этом в качестве результата данный метод возвращает дескриптор текущего сеанса), или создание нового сеанса.

В качестве данных сеанса (т.е. переменных, сохраняемых и извлекаемых из сеанса) выступает объект типа Integer. Его значение увеличивается на единицу при каждом следующем вызове данного сервлета.

В ответ на пользовательский запрос сервлет SessionServlet возвращает страницу HTML, на которой отображается значение сеансовой переменной типа Integer. Это значение соответствует числу раз, которое данный клиент вызывал сервлет SessionServlet в текущем сеансе. Помимо этого, страница HTML содержит форму, значение атрибута ACTION которой указывает на этот же сервлет, так что при желании, щелкнув на кнопке Submit, пользователь может вызвать его еще раз.

Прежде чем перейти к рассмотрению исходного кода сервлета SessionServlet, необходимо отметить еще одну принципиально важную вещь, касающуюся механизма установки сеанса. Сеанс устанавливается непосредственно между клиентом и Web-сервером. Каждый клиент устанавливает с сервером свой собственный сеанс. Таким образом, если запрос к сервлету SessionServlet поступил с компьютера А, и этот запрос был повторен, скажем, четыре раза, то в результате на возвращаемых в качестве ответа страницах HTML будут отображены соответственно числа 1, 2, 3 и 4. Если запрос будет повторен в пятый раз (опять-таки, с компьютера А), то значение, отображаемое на возвращенной в качестве ответа странице HTML, будет равно 5. А вот если запрос к сервлету поступит с компьютера Б, то в этом случае будет установлен новый сеанс, и значение, выведенное на странице HTML, будет равно 1 (а не 6, как было бы в том случае, если бы запрос поступил с компьютера А).

Для того чтобы получше понять технологию установки сеансов, выполните следующее.

1. Создайте страницу HTML, код которой приведен выше (обязательным элементом здесь является форма, щелчок на кнопке Submit которой приводит к вызову сервлета SessionServlet).
2. Откройте данную страницу на каком-нибудь компьютере и щелкните на кнопке Submit.
3. Повторите действия п. 2 несколько раз.

4. Перейдите на другой компьютер, откройте эту же страницу HTML и щелкните на кнопке Submit. Значение, отображаемое на возвращенной в качестве ответа Web-странице, должно быть равно 1, а не увеличенному на единицу значению, показанному в последний раз при обращении к сервлету SessionServlet с первого компьютера. Однако если перейти опять на первый компьютер и щелкнуть на кнопке Submit, то возвращаемые значения будут начинаться с увеличенного на единицу значения, показанного при последнем обращении к сервлету SessionServlet.

Таким образом, приведенная выше последовательность действий еще раз наглядно демонстрирует, что для каждого клиента устанавливается отдельный сеанс.

```
import java.io.*; import javax.servlet.*; import
javax.servlet.http.*; public class SessionServlet
extends HttpServlet
{
public void doPost (HttpServletRequest req, HttpServletResponse res)
throws ServletException, lOException { PrintWriter
out;
//Получить дескриптор текущего сеанса или создать новый сеанс
HttpSession session = req.getSession(true);
res.setContentType("text/html"); out
= res.getWriter();
/*Получить значение переменной "totalAccesses". Если это первое обращение к сервлету SessionServlet, то метод getValueO возвратит null */
Integer accesses = (Integer)session.getValue("totalAccesses");
/*Если это первый вызов данного сервлета (новый сеанс), создать объект типа Integer со значением, равным 1. В противном случае, прочесть значение этого объекта и увеличить его на 1.*/ if(accesses == null)
accesses = new Integer (1); else
{
int total = accesses.intValue(); total++;
accesses = new Integer(total); }
/*Поместить объект типа Integer в список переменных, принадлежащих сеансу*/ session.putValue("totalAccesses", accesses);
/*Возвратить в качестве ответа страницу HTML, разместив на ней значение переменной типа Integer*/ out. print In ("<HTML><HEAD></HEAD>") ; out.println("<BODY> bgcolor=\"#FFFFFF\">");
out.println("<H3>Количество обращений на текущий момент в данном сеансе равно: " + accesses.toString() + "</H3>");
```

```
Добавить форму на страницу HTML, когда пользователь щелкнет на
кнопке Submit данной формы, выполнится сервлет SessionServlet*/
out.println("<FORM ACTION='http://myServer/testZone/SessionServlet'
METHOD='POST'>"); out.println("<INPUT TYPE='submitf
        NAME='Submit'
VALUE='Execute'>");  out.println("</FORM>")
        ; out.println("</BODY></HTML>");
        out.close();}}
```

## Лекция 6 JSP-страницы

JavaServer Pages (JSP) - это стандартное расширение Java, которое определено поверх сервлетных расширений. Цель JSP состоит в упрощении создания и управления динамическими Web страницами.

Как упоминалось ранее, свободно распространяемый Tomcat, реализация от jakarta.apache.org, автоматически поддерживает JSP.

JSP позволяет вам комбинировать HTML Web страницы с кусочками кода Java в одном документе. Java код обрамляется специальными тегами, которые говорят JSP контейнеру, что он должен использовать код для генерации единого документа или его части. Выгода JSP состоит втом, что вы можете содержать единый документ, которые представляет и страницу, и Java код, который в нее включен. Недостаток состоит в том, что тот, кто поддерживает работоспособность JSP страницы, должен иметь опыт и HTML, и в Java (однако, со временем ожидается появление визуальных построителей JSP страницы).

При первой загрузке JSP загружается JSP контейнером (который обычно ассоциирован, или является частью Web сервера), далее сервлетный код, который обязательно выделяется JSP тегами, автоматически генерируется, компилируется и загружается в контейнер сервлетов. Статическая часть HTML страницы производится посредством посылки статического объекта типа String в метод write( ). Динамическая часть включается прямо в сервлет.

С этого момента, пока исходная JSP страница не будет изменена, она будет вести себа так, как будто бы это была статическая HTML страница, ассоциированная с сервлетом (однако весь HTML код на самом деле генерируется сервлетом). Если вы измените исходный код для JSP, он автоматически перекомпилируется и перезагрузится при следующем запросе этой страницы. Конечно, по причине такого динамизма, вы увидите замедленный ответ для первого доступа к JSP. Однако, так как JSP обычно используется гораздо чаще, чем меняется, вы обычно не будете чувствовать влияние этой задержки.

Структура JSP страницы - это что-то среднее между сервлетом и HTML страницей. JSP тэги начинаются и заканчиваются угловой скобкой, также как и HTML коды, но теги также включают знаки процента, так что все JSP теги обозначаются так: <% JSP code here %>.

За лидирующим знаком процента могут следовать другие символы, которые определяют точный тип JSP кода в тэге.

Вот чрезвычайно простой JSP пример, который использует стандартный библиотечный Java вызов для получения текущего времени в миллисекундах, которое затем делится на 1000, чтобы получить время в секундах. Так как используется JSP выражение (<%=), результат вычислений преобразуется в строку, а затем поместится в сгенерированную Web страницу:

```
//:! c15:jsp:ShowSeconds.jsp
<html><body>
<H1>The time in seconds is:
<%= System.currentTimeMillis()/1000 %></H1>
</body></html>
///:~
```

В JSP примерах этой книги восклицательный знак в первой "строке комментариев" означает, что первая и последняя строки не будут включаться в реальный файл кода, который помещен в дерево исходного кода этой книги.

Когда клиент создает запрос к JSP странице, Web сервер должен быть сконфигурирован, чтобы переправить запрос к JSP контейнеру, который затем вовлекает страницу. Как упомянуто ранее, при первом обращении запрашивается страница, генерируются компоненты, указанные на странице, и JSP контейнером компилируется один или несколько сервлетов. В приведенном выше примере сервлет будет содержать код конфигурации объекта HTTPServletResponse, производя объект PrintWriter'a (который всегда называется out), а затем включится расчет времени с помощью очень краткой инструкции, но среднестатистический HTML программист/Web дизайнер не имеет опыта в написании такого кода.

### Неявные объекты

Сервлеты включают классы, которые предоставляют удобные утилиты, такие как HttpServletRequest, HttpServletResponse, Session и т. п. Объекты этих классов встроены в JSP спецификацию и автоматически доступны для использования в вашем JSP без написания дополнительных строк кода. Неявные объекты JSP детально перечислены в приведенной ниже таблице.

Область видимости каждого объекта может значительно варьироваться. Например, объект сессии имеет область видимости, которая превышает страницу, так как она может распространяться на несколько клиентских запросов и страниц. Объект приложения может предоставить

обслуживание группе JSP страниц, которые вместе представляют Web приложение.

JSP директивы

Директивы являются сообщениями JSP контейнеру и обозначаются символом "@":

<%@ directive {attr="value"}* %>

Директивы ничего не посылают в выходной поток, но они важны для настройки атрибутов вашей JSP страницы и зависимостей JSP контейнера. Например, строка:

<%@ page language="java" %>

сообщает, что скриптовой язык, используемый внутри JSP страницы, это Java. Фактически, JSP спецификация только описывает семантику скриптов для языковых атрибутов, эквивалентных "Java". Смысл этой директивы состоит во встраивании гибкости в JSP технологию. В будущем, если вы выберите другой язык, скажем Python (хороший выбор для скриптов), то такой язык должен иметь поддержку Java Run-time Environment, выставляя наружу объектную модель Java технологии для скриптового окружения, особенно для неявных переменных, определенных выше, свойств JavaBeans и публичных методов.

Наиболее важными директивами являются директивы страницы. Они определяют несколько атрибутов страницы и взаимодействие этих атрибутов с JSP контейнером. Эти атрибуты включают: language, extends, import, session, buffer, autoFlush, isThreadSafe, info и errorPage. Например:

<%@ page session=”true” import=&dqut;java.util.*&dqut; %>

Эта строка, прежде всего, указывает, что эта страница требует участие в HTTP сессии. Так как мы не установили директиву языка, JSP контейнер по умолчанию использует Java и неявную переменную скриптового языка с названием session типа javax.servlet.http.HttpSession. Если бы директива использовала false, то неявная переменная session будет недоступна. Если переменная session не определена, то по умолчанию считается "true".

Атрибут import описывает типы, которые доступны для скриптовой среды. Этот атрибут используется так же, как и в языке программирования Java, т. е., разделенный запятыми обычный список выражений import. Этот список импортируется транслируемой реализацией JSP страницы и доступен для скриптового окружения. Скажем еще раз, что в настоящее время это определено, только если значением директивы языка является "java".

Скриптовые элементы JSP

Как только вы использовали директивы для установки скриптового окружения, вы можете использовать скриптовые элементы. JSP 1.1 имеет три скриптовых языковых элемента - декларацию, скриплет и выражение. Декларация декларирует элементы, скриплеты являются фрагментами

инструкций, а выражения являются полным языковым выражением. В JSP каждый скриптовый элемент начинается с "<%". Синтаксис каждого из них:

<%! declaration %>
<% scriptlet %> <%=
expression %>

Пробелы после "<%!", "<%", "<%=" и перед "%>" не обязательны.

Все эти теги основываются на XML; вы даже можете сказать, что JSP страница может быть отражена на XML документ. Эквивалентный синтаксис для скриптовых элементов, приведенных выше, может быть:

<jsp:declaration> declaration </jsp:declaration>
<jsp:scriptlet> scriptlet </jsp:scriptlet>
<jsp:expression> expression </jsp:expression> Кроме
тоге, есть два типа комментариев:
<%-- jsp comment --%> <!--
html comment -->

Первая форма позволяет вам добавлять комментарии в исходный код JSP, которые ни в какой форме не появятся в HTML странице, посылаемой клиенту. Конечно, вторая форма комментариев не специфична для JSP - это обычный HTML комментарий. Интересно то, что вы можете вставлять JSP код внутрь HTML комментария и результат будет показан в результирующей странице.

Декларации используются для объявления переменных и методов в скриптовом языке (в настоящее время только в Java), используемых на JSP странице. Декларация должна быть законченным Java выражением и не может производить никакого вывода в выходной поток. В приведенном ниже примере Hello.jsp декларации для переменных loadTime, loadDate и hitCount являются законченными Java выражениями, которые объявляют и инициализируют новые переменные.

```
//:! c15:jsp:Hello.jsp
<%-- Этот JSP комментарий не появится
в сгенерированном html --%> <%-- Это
JSP директива: --%>
<%@ page import="java.util.*" %> <%--
Эта декларации: --%>
<%!
  long loadTime= System.currentTimeMillis();
  Date loadDate = new Date();
  int hitCount = 0;
%>
<html><body>
```

```
<%-- Следующие несколько строк являются результатом
JSP выражений, вставленных в сгенерированный html; знак
'=' указывает на JSP выражение --%>
<H1>Эта страница была загружена <%= loadDate %> </H1>
<H1>Hello, world! It's <%= new Date() %></H1>
<H2>Here's an object: <%= new Object() %></H2>
<H2>This page has been up
<%= (System.currentTimeMillis()-loadTime)/1000 %> seconds</H2>
<H3>Page has been accessed <%= ++hitCount %>  times
since <%= loadDate %></H3>
<%-- "Скриплет", которые пишет на консоли сервера и
на странице клиента.
Обратите, что необходимо ставить ';': --%>
<%
  System.out.println("Goodbye");
  out.println("Cheerio");
%>
</body></html>
///:~
```

Когда вы запустите эту программу, вы увидите, что переменные loadTime, loadDate и hitCount содержат свои значения между обращениями к странице, так что они явно являются полями, а не локальными переменными.

В конце примера помещен скриплет, который пишет "Goodbye" на консоль Web сервера и "Cheerio" в неявный объект вывода JspWriter. Скриплет может содержать любые фрагменты кода, которые являются имеющими силу инструкциями Java. Скриплеты выполняются во время обработки запроса. Когда все фрагменты скриплета в данном JSP будут скомбинированы по порядку своего появления в JSP странице, они должны дать имеющую силу инструкцию, определенную для языка программирования Java. Будет ли скриплет производить вывод в выходной поток или нет, зависит только от кода скриплета. Вы должны знать, что скриплет может воздействовать на объекты, которые видимы для него.

JSP выражения можно найти вперемешку с HTML в среднем разделе Hello.jsp. Выражения должны быть законченными Java инструкциями, которые вычисляются, переводятся в строку и посылаются в вывод. Если результат инструкции не может быть переведен в строку (String), будет выброшено исключение ClassCastException.

Извлечение полей и значений

Следующий пример похож на приведенный ранее в разделе о сервлетах. При первом показе страницы она определяет, что у вас нет полей и возвращает страницу, содержащую форму с помощью того же самого кода, что и в примере с сервлетом, но в формате JSP. Когда вы отправляете форму с заполненными полями по тому же самому JSP URL'у, страница обнаруживает поля и отображает их. Это прелестная техника, поскольку она позволяет вам получить две страницы, одна из которых содержит форму для заполнения пользователем, а вторая содержит код ответа на эту страницу, в едином файле, таким образом, облегчается создание и поддержка.

```
//:! c15:jsp:DisplayFormData.jsp
<%-- Извлечение данных из HTML формы. --%> <%--
Эта JSP также генерирует форму. --%>
<%@ page import="java.util.*" %>
<html><body>
<H1>DisplayFormData</H1><H3>
<%
 Enumeration flds = request.getParameterNames();
if(!flds.hasMoreElements()) { // Нет полей %>
   <form              method="POST"
action="DisplayFormData.jsp">
<%  for(int i = 0; i < 10; i++) {  %>
     Field<%=i%>: <input type="text" size="20"
name="Field<%=i%>" value="Value<%=i%>"><br>
<%  } %>
   <INPUT TYPE=submit name=submit
   value="Submit"></form>
<%} else {
   while(flds.hasMoreElements()) {
     String field = (String)flds.nextElement();
     String value = request.getParameter(field);
%>
     <li><%= field %> = <%= value %></li>
<%  }
 } %>
</H3></body></html>
///:~
```

Более интересное свойство этого примера заключается в том, что он демонстрирует, как код скриплета может быть перемешан с HTML кодом даже в том месте, где генерируется HTML внутри Java цикла. Это особенно удобно для построения форм любого рода, в противном случае необходимо было бы вставлять повторяющийся HTML код.

Атрибуты JSP страницы и область видимости

Просматривая документацию в HTML для сервлетов и JSP, вы найдете возможность, которая сообщает информацию о текущем запущенном сервлете или JSP. Следующий пример отображает некоторую часть этих данных.

```
//:! c15:jsp:PageContext.jsp
<%-- Просмотр атрибутов pageContext--%>
<%-- Обратите внимание, что вы можете включить любое количество
кода внутри тэгов скриплета --%>
<%@ page import="java.util.*" %>
<html><body>
Servlet Name: <%= config.getServletName() %><br> Servlet
container supports servlet version:
<% out.print(application.getMajorVersion() + "."
+ application.getMinorVersion()); %><br>
<%
  session.setAttribute("My dog", "Ralph");
for(int scope = 1; scope <= 4; scope++) {  %>
   <H3>Scope: <%= scope %> </H3>
<%  Enumeration e =
     pageContext.getAttributeNamesInScope(scope);
while(e.hasMoreElements()) {
out.println("\t<li>" +
       e.nextElement() + "</li>");
   }
  }
%>
</body></html>
///:~
```

Этот пример также показывает использование встроенного HTML и записи в out, чтобы в результате получить HTML страницу.

Первая часть информации производит имя сервлета, которое, вероятнее всего, будет просто "JSP", но это зависит от вашей реализации. Вы можете также определить текущую версию контейнера сервлетов, используя объект application. И, наконец, после установки атрибута сессии в обычной области видимости отображаются "имена атрибутов". Обычно вы не используете область видимости в большинстве JSP; они показаны здесь просто, чтобы добавить интереса в этот пример. Есть три следующие атрибута области видимости: область видимости страницы (scope 1), область видимости запроса (scope 2), область видимости сессии (scope 3 - здесь доступен только

один элемент - это "My dog", добавленный прямо перед циклом) и область видимости приложения (scope 4), основанная на объекте ServletContext. Есть один ServletContext на каждое "Web приложение" в каждой Java Машине. ("Web приложение" - это набор сервлетов и содержимого, установленного под определенным подмножеством URL'ов Сервера, таких как /catalog. Они устанавливаются с помощью конфигурационного файла.) В области видимости приложения вы увидите объекты, которые представляют пути для рабочего каталога и временного каталога.

Манипуляция сессиями в JSP

Сессии были введены в предыдущем разделе, посвященном сервлетам и также доступны в JSP. Следующие примеры исследуют сессионные объекты и позволяют вам манипулировать промежутком времени, прежде, чем сессия станет недействительной.

```
//:! c15:jsp:SessionObject.jsp
<%-- Получение и установка значений объекта сессии --%>
<html><body>
<H1>Session id: <%= session.getId() %></H1>
<H3><li>This session was created at
<%= session.getCreationTime() %></li></H1>
<H3><li>Old MaxInactiveInterval =
  <%= session.getMaxInactiveInterval() %></li>
<% session.setMaxInactiveInterval(5); %>
<li>New MaxInactiveInterval=
  <%= session.getMaxInactiveInterval() %></li>
</H3>
<H2>If the session object "My dog" is  still
around, this value will be non-null:<H2>
<H3><li>Session value for "My dog" =
<%= session.getAttribute("My dog") %></li></H3>
<%-- Теперь добавим объект сессии "My dog" --%>
<% session.setAttribute("My dog",
                new String("Ralph")); %>
<H1>My dog's name is
<%= session.getAttribute("My dog") %></H1>
<%-- See if "My dog" wanders to another form --%>
<FORM TYPE=POST ACTION=SessionObject2.jsp>
<INPUT TYPE=submit name=submit
Value="Invalidate"></FORM>
<FORM TYPE=POST ACTION=SessionObject3.jsp>
<INPUT TYPE=submit name=submit
Value="Keep Around"></FORM>
</body></html>
```

```
///:~
```

Объект сессии предоставляется по умолчанию, так что он доступен без дополнительного кода. Вызов getID( ), getCreationTime( ) и getMaxInactiveInterval( ) используются для отображения информации об этом сессионном объекте.

Когда вы в первый раз получите эту сессию, вы увидите, что MaxInactiveInterval составляет, например, 1800 секунд (30 минут). Это зависит от конфигурации вашего контейнера JSP/сервлетов.
MaxInactiveInterval сокращается до 5 секунд, чтобы сделать вещи интереснее. Если вы перегрузите страницу до того, как истекут 5 секунд, то вы увидите:

Session value for "My dog" = Ralph

Но если вы подождете больший интервал, то "Ralph" станет null.

Чтобы посмотреть, как сессионная информация переносится на другие страницы, а также, чтобы посмотреть эффект становления объекта сессии недействительным по сравнению с простым вариантом, когда вы дали ему устареть, созданы две новые страницы. Первая (доступна при нажатии кнопки "invalidate" в SessionObject.jsp) читает сессионную информацию, а затем явно делает сессию недействительной:

```
//:! c15:jsp:SessionObject2.jsp
<%-- Объект сессии переноситься --%>
<html><body>
<H1>Session id: <%= session.getId() %></H1>
<H1>Session value for "My dog"
<%= session.getValue("My dog") %></H1>
<% session.invalidate(); %>
</body></html>
///:~
```

Чтобы экспериментировать с этим, обновите SessionObject.jsp, затем сразу же кликните на кнопку "invalidate", чтобы перейти к странице SessionObject2.jsp. В этом месте вы все еще будете видеть "Ralph" и сразу после этого (прежде, чем пойдет интервал в 5 секунд), обновите SessionObject2.jsp, чтобы увидеть, что сессия была успешно сделана недействительной и "Ralph" исчез.

Если вы вернетесь на SessionObject.jsp, обновите страницу так, чтобы вы снова имели 5-ти секундный интервал, затем нажмете кнопку "Keep Around", то вы попадете на следующую страницу SessionObject3.jsp, которая не делает сессию недействительной:

```
//:! c15:jsp:SessionObject3.jsp
<%-- Переход объекта сессии по страницам --%>
<html><body>
<H1>Session id: <%= session.getId() %></H1>
```

```
<H1>Session value for "My dog"
<%= session.getValue("My dog") %></H1>
<FORM TYPE=POST ACTION=SessionObject.jsp>
<INPUT TYPE=submit name=submit Value="Return">
</FORM>
</body></html>
///:~
```

Поскольку эта страница не делает сессию недействительной, "Ralph" будет оставаться во время каждого обновления страницы до тех пор, пока не пройдет 5-ти секундный интервал между обновлениями. Это мало чем отличается от игрушки "Tomagotchi" - до тех пор, пока вы играете с "Ralph", он будет здесь, в противном случае он исчезнет.

Создание и изменение cookies

Cookie были введены в предыдущем разделе о сервлетах. Опять таки, краткость JSP делает обращение с cookies более простым, чем при использовании сервлетов. Следующий пример показывает это с помощью получения cookie, которые пришли в запросе, чтения и изменения максимального возраста (дата устаревания) и присоединения нового cookie в ответный запрос:

```
//:! c15:jsp:Cookies.jsp
<%-- Эта программа ведет себя по разному  в
разных броузерах! --%>
<html><body>
<H1>Session id: <%= session.getId() %></H1>
<%
Cookie[] cookies = request.getCookies(); for(int
i = 0; i < cookies.length; i++) { %>
  Cookie name: <%= cookies[i].getName() %> <br>
value: <%= cookies[i].getValue() %><br>   Old
max age in seconds:
  <%= cookies[i].getMaxAge() %><br>
<% cookies[i].setMaxAge(5); %>   New
max age in seconds:
  <%= cookies[i].getMaxAge() %><br>
<% } %>
<%! int count = 0; int dcount = 0; %>
<% response.addCookie(new Cookie(
    "Bob" + count++, "Dog" + dcount++)); %>
</body></html>
///:~
```

Так как каждый браузер хранит cookie по-своему, вы можете видеть разное поведение в разных браузерах (не убежден, но это может быть некоторой ошибкой, которая может быть исправлена в то время, когда вы читаете это). Также вы можете получить разные результаты, если вы выгрузите браузер и запустите его, по сравнению с тем, если вы посетите другую страницу, а затем вернетесь на Cookies.jsp. Обратите внимание, что использование объекта сессии выглядит более уместным, чем прямое использование cookies.

После отображения идентификатора сессии каждый cookie в массиве cookies, пришедший с объектом запроса, отображается на странице наряду с его максимальным возрастом. Максимальный возраст изменяется и отображается снова, чтобы проверить новое значение, затем новый cookie добавляется в ответ. Однако ваш браузер может проигнорировать этот максимальный возраст; стоит поиграть с этой программой и поизменять значение максимального возраста, чтобы посмотреть поведение под различными браузерами.

Заключение о JSP

Этот раздел является только кратким обзором JSP, и даже с тем, что рассмотрено здесь (наряду с тем, что вы выучите о Java в остальной части книги, и вместе с вашим знанием HTML), вы можете начать писать замысловатые web страницы с помощью JSP. Синтаксис JSP не предназначен быть таинственным или сложным, так что если вы понимаете что было представлено в этом разделе, вы готовы к продуктивной работе с JSP. Вы можете найти больше информации в большинстве имеющихся книг о сервлетах или на java.sun.com.

Это особенно удобно иметь доступ к JSP, даже если вашей целью является только создание сервлетов. Вы обнаружите, что если у вас есть вопрос относительно поведения функции сервлета, гораздо легче и быстрее написать тестовую программу на JSP, чтобы получить ответ, чем писать сервлет. Часть удобства состоит в том, что нужно писать меньше кода и можно смешивать отображаемый HTML с Java кодом, но это преимущество становится особенно явным, когда вы увидите, что JSP Контейнер обрабатывает всю перекомпиляцию и перезагрузку JSP за вас в любое время, когда меняется исходный код.

Однако весь ужас JSP состоит в том, что необходимо учитывать, что создание JSP требует большего уровня умения, чем простое программирования на Java или просто создание Web страниц. Кроме того, отладка неработающего JSP не такое простое дело, как в случае Java программы, так как (в настоящее время) сообщения об ошибках более непонятны. Это должно измениться с улучшением систем разработки, но мы можем также видеть другие технологии, основанные на Java и Web, которые лучше адаптированы к умениям дизайнеров Web сайтов.

## Лекция 7 Шаблоны проектирования

Моделирование данных является важнейшим процессом при проектировании программного обеспечения (ПО). По этой причине, разработчики CASE-средств в своих продуктах вынуждены уделять моделированию данных повышенное внимание. Являясь признанным лидером в области объектных методологий, фирма Rational Software Corporation, тем не менее, до недавнего времени такого средства не имела. Основной причиной этого, по-видимому, является ориентация на язык Unified Modeling Language (UML), как универсальный инструмент моделирования. UML полностью покрывает потребности моделирования данных. Сложившаяся на протяжении десятилетий технология моделирования данных, традиции, система понятий и колоссальный опыт разработчиков не могли далее игнорироваться. Немаловажную роль здесь сыграла и необходимость формального контроля моделей данных, что является абсолютно необходимым при проектировании мало-мальски больших схем баз данных и что UML не обеспечивает в достаточной степени. И, наконец, последней причиной, побудившей специалистов Rational Software Corporation к созданию собственного средства моделирования данных, является требование построения эффективных физических моделей, прежде всего для конкретных СУБД - лидеров рынка.

### Шаблон проектирования MVC (Model-View-Controller)

При использовании шаблона MVC поток выполнения приложения всегда обязан проходить через контроллер приложения. Контроллер направляет запросы – в данном случае HTTP(S)-запросы – к соответствующему обработчику. Обработчики запроса связаны с бизнес-моделью, и в итоге каждый разработчик приложения должен только обеспечить взаимодействие между запросом и бизнес-моделью. В результате реакции системы на запрос вызывается соответствующая JSP-страница, выполняющая в данной схеме роль представления.

В результате модель отделена от представления, и все связывающие их команды проходят через контроллер приложения. Приложение, соответствующее этим принципам, становится более понятным с точки зрения разработки, поддержки и совершенствования, а отдельные его части довольно легко могут быть использованы повторно.

В оригинальной концепции была описана сама идея и роль каждого из элементов: модели, представления и контроллера. Но связи между ними

были описаны без конкретизации. Кроме того, различали две основные модификации:

– Пассивная модель — модель не имеет никаких способов воздействовать на представление или контроллер, и используется ими в качестве источника данных для отображения. Все изменения модели отслеживаются контроллером и он же отвечает за перерисовку представления, если это необходимо. Такая модель чаще используется в структурном программировании, так как в этом случае модель представляет просто структуру данных, без методов их обрабатывающих.

– Активная модель — модель оповещает представление о том, что в ней произошли изменения, а представления, которые заинтересованы в оповещении, подписываются на эти сообщения. Это позволяет сохранить независимость модели как от контроллера, так и от представления.

Классической реализацией концепции MVC принято считать версию именно с активной моделью.

С развитием объектно-ориентированного программирования и понятия о шаблонах проектирования был создан ряд модификаций концепции MVC, которые при реализации у разных авторов могут отличаться от оригинальной. Так, например, Эриан Верми в 2004 году описал пример обобщенного MVC.

Концепция MVC позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

Модель (англ. Model). Модель предоставляет знания: данные и методы работы с этими данными, реагирует на запросы, изменяя своё состояние. Не содержит информации, как эти знания можно визуализировать.

Представление, вид (англ. View). Отвечает за отображение информации (визуализацию). Часто в качестве представления выступает форма (окно) с графическими элементами.

Контроллер (англ. Controller). Обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Важно отметить, что как представление, так и контроллер зависят от модели. Однако модель не зависит ни от представления, ни от контроллера. Тем самым достигается назначение такого разделения: оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

Для реализации схемы Model-View-Controller используется достаточно большое число шаблонов проектирования (в зависимости от сложности архитектурного решения), основные из которых «наблюдатель», «стратегия», «компоновщик».

Наиболее типичная реализация отделяет вид от модели путем установления между ними протокола взаимодействия, используя аппарат событий (подписка/оповещение). При каждом изменении внутренних данных в модели она оповещает все зависящие от неё представления, и представление обновляется. Для этого используется шаблон «наблюдатель». При обработке реакции пользователя вид выбирает, в зависимости от нужной реакции, нужный контроллер, который обеспечит ту или иную связь с моделью. Для этого используется шаблон «стратегия», или вместо этого может быть модификация с использованием шаблона «команда». А для возможности однотипного обращения с подобъектами сложно-составного иерархического вида может использоваться шаблон «компоновщик». Кроме того, могут использоваться и другие шаблоны проектирования, например, «фабричный метод», который позволит задать по умолчанию тип контроллера для соответствующего вида.

Запрос к серверу обрабатывается классом-контроллером **ActionServlet** в соответствии с настройками в файле **web.xml**. Запрос на сервер выполняет пользователь нажатием кнопки Submit, введением URL в поле браузера, вызовом submit-формы на JavaScript и пр. Во время инициализации главный контроллер считывает (parse) конфигурационный файл **struts-config.xml**, который однозначно определяет все соответствия и альтернативы для всех запросов данного приложения и упаковывает их в объект класса **org.apache.struts.action.ActionMapping.**

Для запроса контроллер находит соответствующие ему **ActionForm**- и **Action**-классы. Первым создается ActionForm, если такой объект еще не создавался и не находится ни в одной из областей видимости (сессия, запрос). Далее поля объекта ActionForm заполняются данными, которые пришли с запросом, т.е. для каждого параметра, содержащегося в запросе, вызывается соответствующий set-метод. Например, если в запросе есть параметр **login=goch**, то в **ActionForm** будет вызван метод **setLogin()** с передачей значения параметра. В Struts существует возможность заполнения структуры объектов. Пусть в **ActionForm** с помощью методов **setCompany()** и **getCompany()** передается объект класса Company, полем которого является коллекция объектов типа **Employee**, для каждого из которых определено поле типа **Address**.

Тогда для заполнения поля адреса одного из работников достаточно присутствия параметра **company.employees[n].address.phone**

**Простое приложение**

*/* пример1: Action класс : LoginAction.java */*

package jspservlet;

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionMessage;
import org.apache.struts.action.ActionMessages;

public class LoginAction extends Action {
        public ActionForward execute(ActionMapping mapping,
                                        ActionForm form, HttpServletRequest
                                    request, HttpServletResponse response)
                        throws IOException, ServletException {
                ActionMessages errors = new ActionMessages();

                LoginForm actionForm = (LoginForm)form;
                String login = actionForm.getLogin();
                String password = actionForm.getPassword();
        if (login != null && password != null) {
                if (LoginLogic.checkLogin(login, password)) {
                        return mapping.findForward("success");
                } else {
errors.add(ActionMessages.GLOBAL_MESSAGE,
new ActionMessage("error.login.incorrectLoginOrPassword"));
                        saveErrors(request, errors);
                                        }
        }
    //загрузка формы для логина
        return mapping.findForward("loginAgain");
   }
}
```

Класс LoginForm, объект которого представляет форму, соответствующую странице ввода логина и пароля, выглядит следующим образом:

```
/* пример : класс хранения информации, передаваемой из login.jsp :
LoginForm.java */
package jspservlet;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionForm;
```

```
import org.apache.struts.action.ActionMapping;

public class LoginForm extends ActionForm {
        private String login;
        private String password;
                //очищение полей формы
        public void reset(ActionMapping mapping,
                                HttpServletRequest request) {
                super.reset(mapping, request);
                this.password = null;
        }
        public String getLogin() {
                return login;
        }
        public String getPassword() {
                return password;
        }
        public void setLogin(String login) {
                this.login = login;
        }
        public void setPassword(String password) {
                this.password = password;
        }
    }
```

Для данного приложения файл ресурсов может быть создан в виде:

```
# пример : файл ресурсов :  ApplicationResources.properties
# header и footer, которые будут использоваться для
# обрамления ошибок, выдаваемых тегом <errors/>.
errors.header=<ul>
errors.footer=</ul>
errors.prefix=<li>
errors.suffix=</li>
# разметка для элемента списка ошибок из <errors/>,
# который указывает, что логин или пароль неверны.
error.login.incorrectLoginOrPassword=<li>incorrect login or password</li>

# текстовая информация на login.jsp
jsp.login.title=Login
jsp.login.header=Login
jsp.login.field.login=Login
jsp.login.field.password=Password
```

```
jsp.login.button.submit=Enter

# текстовая информация на main.jsp
jsp.main.title=Welcome
jsp.main.header=Welcome
jsp.main.hello=Hello

# текстовая информация на error.jsp
jsp.error.title=Error
jsp.error.header=Error
jsp.error.returnToLogin=Return to login page
```

Имя и месторасположение этого файла настраиваются в struts-config.xml, который является основным конфигурационным файлом для приложения, построенного на основе Struts.

*пример : конфигурация action, forward, resource и т.д. : struts-config.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache
Software Foundation//DTD Struts Configuration 1.2//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
<struts-config>
        <data-sources>
        <!--источники данных к БД. Как правило, это организация пула
соединений. Пример приведен в конце раздела-->
        </data-sources>
<!-- ========== Form Bean Definitions -->
                <form-beans>
                <form-bean  name="loginForm"
                type="jspservlet.LoginForm" />
   </form-beans>

<!-- ============== Global Exception Definitions (если есть)-->
   <global-exceptions>
         </global-exceptions>
<!-- ============== Global Forward Definitions(если есть)  -->
        <global-forwards>
                </global-forwards>

<!-- ============== Action Mapping Definitions -->
        <action-mappings>
                <!-- Action для процесса логина -->
        <action name="loginForm"
```

```
        path="/login"
        scope="request"
<!—- Задается область видимости формы. Часто необходимо, чтобы форма лежала в сессии, а не в запросе. -->
        type="jspservlet.LoginAction"
        validate="false">
```

Получить данный пул из класса, реализующего org.apache.struts.action.Action, можно с помощью метода getDataSource(HttpServletRequest request)).

Оставшиеся JSP-страницы были изменены несущественно:

```
<%-- пример : страница, вызываемая после прохождения идентификации : main.jsp --%>
<%@ page errorPage="error.jsp" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html:html locale="true">
<head>
      <title><bean:message key="jsp.main.title"/></title>
      <html:base/>
</head>
<body>
<h3><bean:message key="jsp.main.header"/></h3>
<hr/>
<bean:message key="jsp.main.hello"/>,
            <bean:write name="loginForm" property="login"/>
<hr/>
<a href="login.do">
<bean:message
                  key="jsp.error.returnToLogin"/></a>
<html:errors/>
</body>
</html:html>
```

## Лекция 8 Remote Metod Invocation

Java Remote Method Invocation (RMI) - интерфейс вызова удаленных методов. Распределенная объектная модель, специфицирующая, каким образом производится вызов удаленных методов, работающих на другой виртуальной машине Java.

Приложения RMI часто рассматриваются как две раздельные программы – сервер и клиент. Типичное серверное приложение создает некоторые удаленные объекты, делает ссылки к ним доступными, и ожидает от клиентов вызова методов для этих объектов. Типичное клиентское приложение получает удаленную ссылку к одному или нескольким удаленным объектам на сервере, а затем вызывает для них методы. Система RMI предусматривает механизм, с помощью которого сервер и клиент связываются друг с другом и обмениваются информацией.

Распределенное приложение, построенное с помощью Java RMI, создано из интерфейсов и классов. Интерфейсы определяют методы, а классы реализуют методы, определенные в интерфейсах, а также, возможно, определяют и новые дополнительные методы. В распределенном приложении предусматривается размещение некоторых реализаций на различных виртуальных машинах. Объекты, имеющие методы, которые могут вызываться между различными виртуальными машинами, являются удаленными объектами – remote objects.

Объект становится удаленным объектом при реализации удаленного интерфейса – remote interface, который имеет следующие характеристики:

1. Удаленный интерфейс расширяет интерфейс java.rmi.Remote; 2. Каждый метод интерфейса объявляет java.rmi.RemoteException в своем условии throws, помимо любых других исключений, специфичных для конкретного приложения.

При передаче объекта от одной виртуальной машины к другой RMI рассматривает удаленный объект – remote object не так, как обычный не удаленный объект. Вместо того, чтобы делать копию реализации объекта в принимающей виртуальной машине, RMI передает удаленный stub – заглушку для удаленного объекта. Этот stub действет как локальное представление, или заместитель (proxy) для удаленного объекта, а для вызывающего является удаленной ссылкой. Вызывающий вызывает метод для локального stub, который отвечает за обработку вызова метода удаленного объекта.

Переданный stub для удаленного объекта реализует тот же набор удаленных интерфейсов, который реализует удаленный объект. Это позволяет для stub быть "передатчиком" к любому из интерфейсов, которые реализует удаленный объект. Однако это также означает, что только те методы, которые определены в удаленном интерфейсе, могут быть вызваны на принимающей виртуальной машине.

Рассмотрим создание простейшего приложения на примере Hello World. Сервер предоставляет один метод, принимающий в качестве параметра имя и возвращающий строку «Hello имя».

1. Создаем интерфейс IHelloWorld

```
package chstu.ds.lab1;
```

```
import java.rmi.Remote; import java.rmi.RemoteException;
//интерфейс должен расширять Remote public interface
IHelloWorld extends Remote {  //        и генерировать
RemoteException  public String SyaHello(String name) throws
RemoteException; }
```

2. Создаем его реализацию:

```
package chstu.ds.lab1;

import java.rmi.RemoteException; import
java.rmi.server.UnicastRemoteObject;
public class HelloImpl extends UnicastRemoteObject implements
IHelloWorld {
 /* UnicastRemoteObject предоставляет такую реализацию многих
  * методов класса java.lang.Object (equals, hashCode,
* toString), подходящие для удаленных объектов */
public HelloImpl() throws RemoteException{
        super();
 }    public String SyaHello(String name) throws
RemoteException{
        return "Hello "+name;
 }
}
```

3. Создаем сервер **package**
chstu.ds.lab1;

```
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class HelloServer {

 public static void main(String[] args) {
        // задаем файл политики безопасности
        System.setProperty("java.security.policy","D:\\rmi.policy");
  // Указываем сервер на котором регистрироваться и имя сервиса
  String name = "rmi://localhost/HelloService";       if
(System.getSecurityManager() == null) {
```

```
System.setSecurityManager(new
RMISecurityManager());
        }
                try {
                // создаем RMI Registry. Если его не создавать,
  // то требуется запускать отдельно. RMI Registry является
          // простым сервисом именований и позволяет клиенту
  // получить ссылку на удаленный объект по имени

LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
                IHelloWorld hw = new HelloImpl();
                // регистрируем объект в RMI Registry
                Naming.rebind(name, hw);
            System.out.println("ComputeEngine bound");
        } catch (Exception e) {
            System.err.println("ComputeEngine exception: " +
                        e.getMessage());
            e.printStackTrace();
        }
 }

}
```

5. Создаем клиентский класс

```
package chstu.ds.lab1; import
java.rmi.*; public class HelloClient {
public static void main(String[] args)
{
          try {
          // адрес RMI Registry и имя сервиса
           String name = "rmi://localhost/HelloService";
          // получаем клиентскую заглушку
           IHelloWorld hw = (IHelloWorld) Naming.lookup(name);
           // вызываем метод удаленного объекта
          System.out.println(hw.SyaHello("Vasya"));
        } catch (Exception e) {
          System.err.println("ComputePi exception: " +
e.getMessage());
          e.printStackTrace();
        }
 }
```

}

4. Запускаем сервер 5.
Запускаем клиент.

## Список использованных источников


1. Дейтел Х. М., Дейтел П. Д., Сантри Технология программирования на Java2. Книга 1. Графика, JavaBeans, интерфейс пользователя. - М.: Бином, 2003. ISBN: 5-9518-0017-X.
2. Дейтел Х. М., Дейтел П. Д., Сантри Технология программирования на Java2. Книга 2. Распределенные приложения. - М.: Бином, 2003. ISBN: 5-9518-0051-X.
3. Дейтел Х. М., Дейтел П. Д., Сантри Технология программирования на Java2. Книга 3. Корпоративные системы, сервлеты, JSP, web-сервисы. - М.: Бином, 2003. ISBN: 5-9518-0034-X.
4. Дейтел Х. М., Дейтел П. Д. Как программировть на Java. Книга 1. Основы программирования. - М.: Бином, 2003. ISBN: 5-9518-0015-3.
5. Дейтел Х. М., Дейтел П. Д. Как программировть на Java. Книга 2. Файлы, сети, базы данных. - М.: Бином, 2006. ISBN: 5-9518-0127-3.
6. Ноутон П., Шилдт Г. Java2. Наиболее полное руководство. - СПб.: BHV, 2001. ISBN: 0-07-211976-4, 5-94157-012-0.
7. Биллиг В.А. Основы программирования на Java/ - М.: Изд-во «Интернетуниверситет информационных технологий – ИНТУИТ.ру», 2006. – 488с.
8. Шилдт Г.С. Java: Учебный курс. – СПб.:Питер, 2002. – 512с.
9. Шилдт Г.С. Полный справочник по Java. – М.: Издательский дом «Вильямс», 2004. – 752с.