Математические структуры и моделирование
2014. №2(30). С. 77–86

УДК 004.9

РАЗРАБОТКА СЕССИОННОГО КОНТЕЙНЕРА СЕРВЕРНОГО JAVA-КОДА

В.С. Виноградов
студент, e-mail: it.vinogradov@yandex.ru

Факультет компьютерных наук, Омский государственный университет

Аннотация. В работе описана задача создания контейнера серверного кода, поддерживающего работу в режиме постоянного соединения между сервером и клиентом, а также приведены основные принципы практической реализации подобной программной технологии.

Ключевые слова: java, сеть, программирование, сервер, клиент, сокеты.

Введение

Активное развитие информационных технологий за последнее десятилетие и популяризация их среди населения повлекли за собой развитие сетевого программирования. Из-за чрезвычайной популярности интернет-приложений и сетевых сервисов потребовалось разработать программную технологию, позволяющую эффективно управлять сложной логикой подобных приложений, эффективно использовать ресурсы, предоставляемые сервером, и упростить процесс развёртывания и интеграции написанных разработчиками программ. Так была разработана концепция, получившая название «контейнер кода». Отдельным популярным подвидом современных контейнеров кода стали так называемые «контейнеры сервлетов», которые пришли к нам из мира java-разработки. Сервлет — это спецификация, описывающая программные объекты, обслуживающие запросы к серверу приложения.

Концепция «контейнера кода» получила широкое распространение и была реализована в том или ином виде для большинства современных языков программирования, ориентированных на разработку веб-сервисов или сетевых приложений. К сожалению, большая часть существующих решений работает по протоколу http [1] (в т.ч. и упомянутые выше сервлеты, хотя это и не ограничивается их спецификацией), который подходит для реализации далеко не всех классов задач, которые встают перед современными программистами.

Часто необходимо использовать постоянное соединение с сервером приложения. При этом использование технологии «контейнера кода» часто является предпочтительной для разработки серверной части с точки зрения ускорения процесса разработки и достижения определённого уровня абстракции кода приложения. Возникает необходимость для разработки библиотеки для подобных целей, поскольку использование готовых универсальных решений не всегда является возможным.

1. Контейнер серверного кода

Появление контейнеров серверного кода — результат исторического развития подходов к разработке программного обеспечения. Изначально системы имели монолитную архитектуру. В связи с увеличением требования к сложности и гибкости систем, начался процесс роста их степени распределённости и усложнения. В итоге, разработчики программного обеспечения пришли к использованию распределённой компонентной архитектуры.

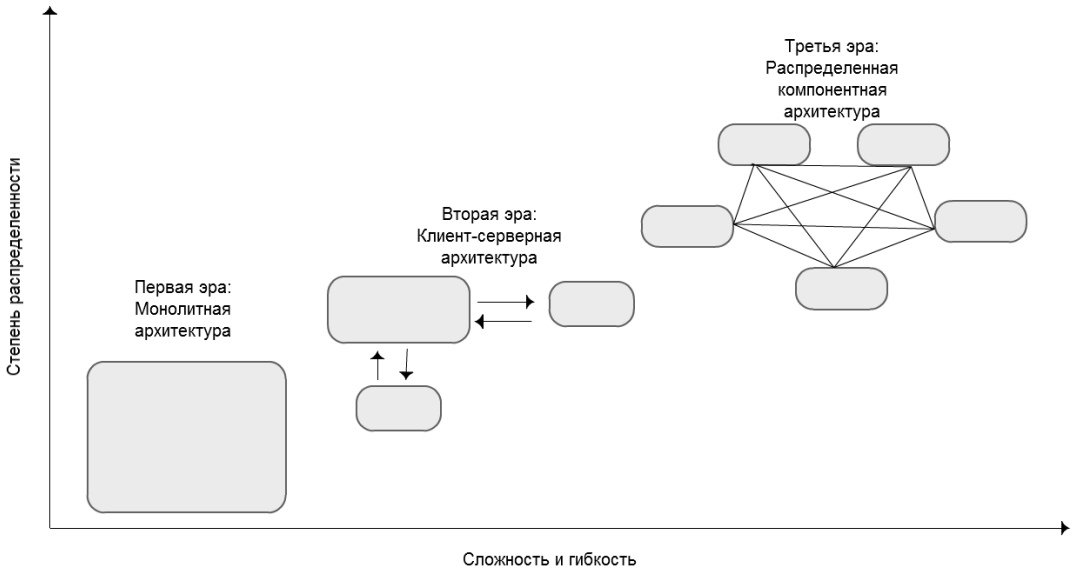

Рис. 1. Основные этапы исторического развития подходов к разработке ПО

Основные этапы исторического развития приведены на рисунке 1.

Контейнер серверного кода — это программный компонент, являющийся частью вычислительной системы, который, во-первых, выполняет обслуживающие (а также сервисные) функции по запросу клиента, организуя для него доступ к специальным объектам, содержащим программный код бизнес-логики, а во-вторых, с помощью реализованных в нем алгоритмов, управляет объектами с программным кодом, запросами клиентов и другими логическими структурами, и, в-третьих, может выполнять специфичные функции, например, построение распределённой системы серверов бизнес-логики в автоматическом режиме, маршрутизация запросов и другое.

Понятие контейнера серверного кода впервые пришло к нам из мира программирования Java. Там понятие контейнера кода применяется, в основном, в области использования контейнеров сервлетов [2]. Стоит упомянуть, что сервлетом называются специальные java-интерфейсы, расширяющие возможности контейнера кода и описанные в стандарте JSR-340. Мы будем называть сервлетом любой программный объект, содержащий код бизнес-логики, позволяющий контейнеру взаимодействовать с ним путём вызова специальных методов. Контейнер контролирует обмен данными между клиентом и сервером,

создаёт программную среду для выполняющегося сервлета, идентификацию и аутентификацию клиентов, организацию сессий для них. Контейнер сервлетов может работать самостоятельно, а может интегрироваться с сторонними веб-серверами, либо быть частью сервера приложений.

Сервер приложений — более сложная и общая структура, включающая в себя большое количество модулей. Обычно контейнер сервлетов является частью сервера приложений. Данная программная платформа предназначена для эффективного исполнения процедур, необходимых для построения и развёртывания приложений. Компоненты сервера, как правило, доступны разработчику через специальные программные интерфейсы.

2. Преимущества использования контейнеров серверного кода

Использование контейнеров серверного кода при разработке серьёзных приложений, имеющих сложный функционал, является предпочтительным. Существует ряд неоспоримых преимуществ, которые обеспечивают популярность контейнеров кода и серверов приложений среди современных разработчиков программного обеспечения. Можно выделить 4 комплексных причины такой популярности данной технологии.

Первая причина — контейнеры серверного кода обеспечивают целостность данных и хранимого в них кода бизнес-логики. Используя данную технологию, разработчики выделяют бизнес-логику на выделенные сервера или на несколько таких серверов, что может гарантировать получение обновлений или улучшений приложения всеми пользователями одновременно. Ситуация, в которой старая версия приложения получит доступ к данным или, например, сможет изменить их старым, несовместимым образом, является невозможной.

Во-вторых, часто в ходе модернизации или доработки программной системы возникает необходимость сменить некоторые технологии, используемые при разработке, либо совершать иные централизованные изменения и настройку. Использование контейнера серверного кода позволяет производить централизовано такие изменения, как изменение сервера базы данных или изменение общих глобальных настроек системы.

Значимой причиной также является улучшение безопасности в разрабатываемых системах. В таком случае, контейнер является центральной точкой, через которую поставщики различных сервисов могут организовывать управление доступом к данным или частям самой системы, что считается преимуществом защиты. Наличие центральной точки позволяет переложить идентификацию и аутентификацию с потенциально небезопасного уровня клиента на уровень контейнера, при этом дополнительно изолируя уровень базы данных от несанкционированного доступа. Контейнер также является центральным хранилищем сеансов взаимодействия с клиентами, которыми он сам и управляет. Что положительно сказывается на общем улучшении безопасности разрабатываемых приложений. Также большим преимуществом является поддержка транзакционности контейнером серверного кода. Транзакция в данном контексте трак-

туется как единица активности, во время которой большое число изменений ресурсов (в одном или нескольких источниках) может быть выполнено атомарно, т.е. либо все изменения выполняются вместе, либо не выполняются вообще.

Упомянутая поддержка транзакций позволяет получать выгоду конечным пользователям системы, т.к. стандартизированное поведение системы с такой архитектурой ведёт к уменьшению времени, которое требуется на разработку, а также к снижению стоимости продукта для пользователей. Поскольку контейнер автоматизирует выполнение большого числа технических действий и выполняет массу нужного генерирования кода, разработчики могут сфокусироваться на написании бизнес-логики приложения, что позволяет эффективно использовать интеллектуальный ресурс программистов и их производительность труда.

3. Общая функциональная архитектура

Архитектуры существующих решений в области контейнеров серверного кода отличаются между собой. Но, несмотря на это, можно выделить общие моменты и описать модули и слои, которые включает в себя большая часть известных решений.

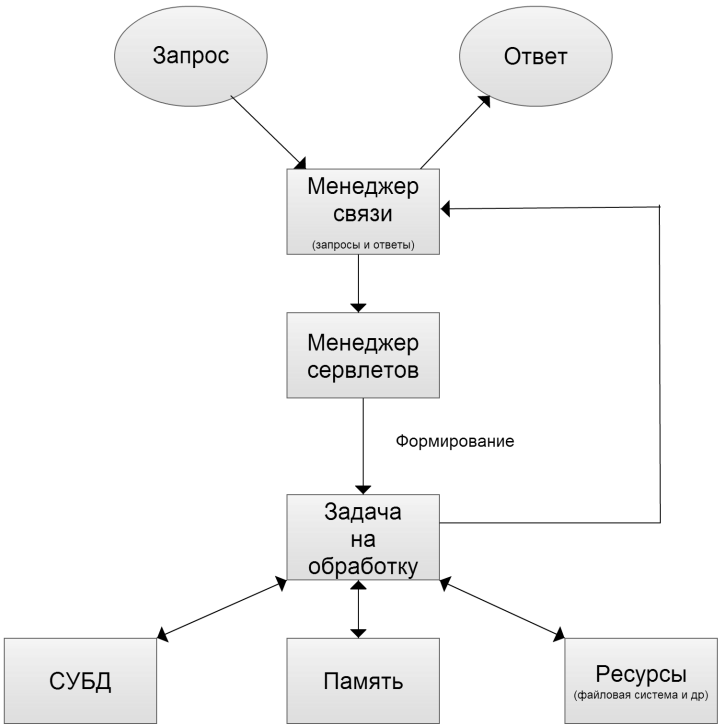

Рис. 2. Общая архитектура существующих решений

Рисунок 2 наглядно демонстрирует, из каких модулей состоят контейнеры серверного кода. Все запросы, приходящие с клиентской стороны на сервер

приложения, попадают в менеджер связи. Он отвечает за работу с сетью — управляет открытием и закрытием соединения, занимается обработкой пришедших пакетов, приводит данные и поля запроса к специальному виду, чтобы к ним было легко обращаться в коде бизнес-логики. Он же отвечает за упаковку данных, подготовленных к отправке клиенту, и отправку этих данных.

Данные, извлечённые из сети, передаются в менеджер сервлетов. Тот по полям запроса определяет, какой сервлет необходимо вызвать. Формируется задача на обработку. Если данный сервлет не загружен в память контейнера, то выполняются 3 основных шага: класс контейнера загружается контейнером, контейнер создает экземпляр сервлета и вызывает метод инициализации сервлета (который вызывается 1 раз за весь жизненный цикл). Менеджер определяет тип запроса, вызывается специальный метод сервлета, которому передаются параметры запроса, происходит обработка запроса. В ходе выполнения кода бизнес-логики сервлет с помощью специальных программных интерфейсов может обращаться к контейнеру, осуществляя взаимодействие с общей памятью, СУБД и иными ресурсами, например, файловой системой.

Далее, с помощью специальных алгоритмов по оптимизации использования ресурса вычислительной системы, учитывая конфигурационную информацию, контейнер определяет, выгружать сервлет из памяти, либо оставить его до получения следующего запроса (либо до истечения определённого времени, после которого сервлет становится неактивным). Подготовленный ответ передаётся в менеджер связи, где из него формируется пакет определённого формата, отсылаемый на клиент.

4. Примеры популярных контейнеров кода

Самыми популярными технологиями на сегодняшний день являются контейнеры: Apache Tomcat [3], Jetty [4] и сервера приложений: WildFly [5], GlassFish [6], WebSphere [7], WebLogic [8]. К разрабатываемой архитектуре наиболее близки Apache Tomcat и Jetty. Сделав обзор этих двух приложений, мы выберем наиболее подходящие для решения нашей задачи и сравним их между собой.

5. Apache Tomcat

Это контейнер сервлетов с открытым исходным кодом, выпускается под лицензией Apache License 2.0 [9]. Разработчик — Apache Software Foundation. Реализует спецификацию сервлетов и спецификацию JSP [10], JSF [11]. Позволяет разворачивать веб-приложения, содержит ряд программ для самоконфигурирования. Контейнер может использоваться в качестве самостоятельного веб-сервера, в паре с отдельным веб-сервером, либо в качестве контейнера сервлета в серверах приложений.

Состоит из трёх основных компонентов. Catalina — это непосредственно контейнер сервлетов. Реализует упомянутые выше спецификации. В качестве компонента стека HTTP используется Coyote, который прослушивает входящие

запросы, передаёт данные на обработку и рассылает ответы. Последним важным модулем является Jasper, который реализует механизм JSP-контейнера. Умеет автоматически обнаруживать изменения JSP-файлов и перекомпилировать их в java-код.

6. Jetty

Jetty — свободно распространяемый контейнер сервлетов, полностью написанный на языке Java. Может использоваться в качестве http-сервера, может работать в паре с другим сервером. Изначально разрабатывался под лицензией Apache License 2.0, но в 2009 году стал доступен также под лицензией Eclipse Public License. Полностью поддерживает спецификацию JSP и Servlet API 3.0.

7. Сравнение Apache Tomcat и Jetty

Поскольку эти контейнеры максимально подходят под архитектуру предполагаемого решения, сравнение разрабатываемого контейнера с ними будет корректно. Для более подробного понимания проблемы имеет смысл выявить сходства и различия между этими двумя технологиями.

И Tomcat, и Jetty являются контейнерами серверного кода в чистом виде. И тот, и другой контейнер могут использоваться как самостоятельно, так и в паре со сторонним веб-сервером. Обе технологии реализуют последние и актуальные спецификации. Оба распространяются под свободными лицензиями и активно разрабатываются, не отставая в развитии от других платформ. На этом их сходства заканчиваются.

В отличие от Jetty, Tomcat часто используется в качестве компонента в серверах приложений. Tomcat, по своей архитектуре, достаточно близок к таким серверам. Jetty же имеет модульную архитектуру, что позволяет добавлять необходимый или исключать лишний функционал. Tomcat больше подходит для реализации приложений, где бизнес-логика предполагает работу с высоконагруженными [12] соединениями и большим количеством легковесных запросов и раздачей динамического контента. Jetty же хорошо справляется с раздачей тяжёлого статического контента. К тому же, он способен хорошо масштабироваться под платформу Web.

8. Постановка задачи

Использование контейнера серверного кода в разработке приложений является большим преимуществом: это позволяет сэкономить время разработки, занимаясь написанием бизнес-логики, а не технических моментов приложения, уменьшить конечную стоимость продукта для пользователя и формализовать процесс разработки как таковой. Для определённых задач требуется наличие возможности инициировать передачу данных не только со стороны клиента, но и со стороны сервера. Отсюда вытекает постановка задачи.

Требуется разработать контейнер серверного кода, поддерживающий двустороннюю передачу данных как от клиента к серверу, так и от сервера к клиенту, использующий постоянно открытое соединение для снижения накладных расходов на передачу данных и удовлетворяющий основным принципам и архитектурным решениям, описанным в предыдущих пунктах.

9. Архитектура

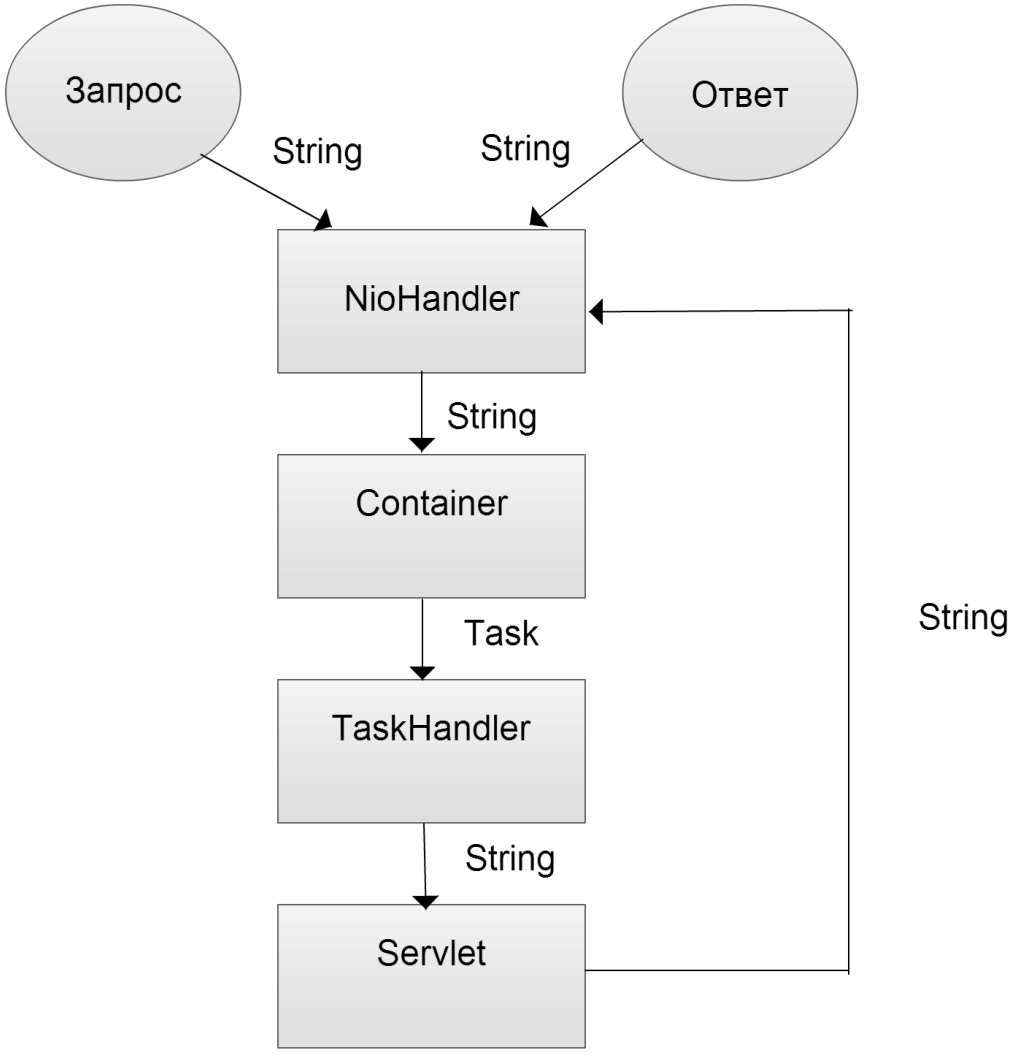

Рис. 3. Архитектура прототипа

На рисунке 3 изображена предполагаемая схема прототипа, которая была специально упрощена, чтобы прототип точно следовал архитектуре из первой главы.

Servlet — это собственный Java-интерфейс, в котором определён метод doRequest с параметром data типа String. Далее по тексту будем называть

сервлетами объекты, реализующие наш интерфейс Servlet и хранящие код бизнес-логики.

Контейнер из канала связи получает некоторое сообщение, которое является строкой в специальном формате. За это отвечает класс NioHandler, который является обработчиком всех событий и процессов, связанных с передачей данных. Формат строки состоит из двух ключевых частей: `идентификатор_сервлета[системный_разделитель]строка_данных`. NioHandler реализован на основе библиотеки Netty, которая предназначена для асинхронной обработки высоконагруженных соединений.

Container является главным классом нашей библиотеки-контейнера, он хранит в себе все разделяемые данные, которые должны быть доступны всем сервлетам, в том числе и служебные данные. Также он управляет всеми основными процессами, которые протекают в ходе функционирования контейнера серверного кода. Он получает от NioHandler строку в описанном выше формате, делит ее на две части, удаляя системный разделитель, получает имя сервлета и формирует задачу, которая в коде представляется объектом класса Task.

Task — это специальный класс, хранящий задачу на обработку для контейнера серверного кода. В нашем прототипе он содержит разделённые данные: идентификатор сервлета, который нужно вызвать, и строку полученных данных. Конструктор объектов данного класса принимает эти данные, после чего созданный экземпляр передаётся специальному обработчику.

TaskHandler — это обработчик задач, который является модулем нашего контейнера. Он обращается к базе сервлетов, хранящейся в общей памяти, и вызывает метод исполнения бизнес-логики сервлета, соответствующего хранящемуся в задаче идентификатору. Также он обрабатывает исключительные ситуации, например, отсутствие в базе сервлета с соответствующим идентификатором. После выполнения кода логики сервлет возвращает (передаёт) строку-ответ экземпляру класса NioHandler, который высылает ответ клиенту по сети.

10. Методика тестирования

Методика тестирования была разработана с целью проверить, имеет ли практический смысл разработка прототипа. Параметр, подлежащий оценке в ходе тестов — время отклика от сервера, которое является промежутком от отправки запроса с клиента и до получения ответа от сервера на нем же.

Для оценки разрабатываемого решения использовались результаты тестов нашего контейнера в сравнении с результатами тестов контейнеров сервлетов Apache Tomcat и Jetty.

Для тестирования на всех трёх системах был написан echo-сервер, который получает заданную строку и отправляет её обратно на клиент. Это позволяет оценить затраты на передачу данных и служебные операции и исключить влияние кода бизнес-логики на результаты тестирования. Тест был проведён 3 раза, каждая итерация теста включала в себя 10 000 запросов к серверу, замер времени и усреднения результатов.

11. Результаты тестирования

Результаты тестирования показывают, что разработка контейнера серверного кода с поддержкой постоянного соединения имеет практический смысл. Среднее время на запрос при использовании Apache Tomcat составило 0.99 мс на запрос, при использовании Jetty — 1.12 мс. Реализованный контейнер серверного кода за счёт постоянно установленного соединения показал цифру на 0.07 мс на запрос, что превосходит наилучший результат тестирования существующего решения примерно в 14 раз. В совокупности с преимуществами, заложенными в контейнер при проектировании, например, возможности инициировать передачу данных от сервера к клиенту, имеется достаточный повод для продолжения разработки библиотеки по выбранному направлению.

12. Заключение

Была построена модель библиотеки, поддерживающей постоянное соединение с сервером в ходе своей работы, даны оценки недостаткам и преимуществам этой модели и сделаны выводы о перспективах её реализации под платформу Java.

Разработанная концепция контейнера серверного кода позволяет эффективно организовать процесс разработки приложений, избежать временных затрат на разработку технических модулей конечного приложения, например, модулей связи. Это позволяет уменьшить конечную стоимость приложения и получить архитектурные преимущества при его проектировании.

В ходе анализа рынка было установлено, что большинство популярных существующих решений показывают достаточную для некоторых задач производительность, но, к сожалению, многие из них работают по http-протоколу, что делает невозможным инициацию отправки данных от сервера к клиенту.

Тестирование прототипа, созданного по разработанной архитектуре, показало, что практическая разработка данного контейнера имеет смысл, поскольку для определённого класса задач такая производительность, с учётом архитектурных возможностей контейнера, является достаточной.

Развитие предложенного решения можно предположить по нескольким направлениям:

1. Добавление сервисных функций: реализация фильтров, общей памяти, встроенного популярного функционала (например, чата)
2. Внедрение криптографических сервисов для защиты передаваемой информации
3. Валидацию входных данных и универсальную систему получения параметров запроса

Литература

1. Таненбаум Э. Компьютерные сети. 4-е изд. СПб. : Питер, 2003. С. 735–741.

2. Спецификация Java Servlet API 3.1 [Электронный ресурс]. URL: http://download.oracle.com/otn-pub/jcp/servlet-3_1-fr-eval-spec/servlet-3_1-final.pdf (дата обращения: 03.04.2014).
3. Документация Apache Tomcat 7.0 [Электронный ресурс]. URL: http://tomcat.apache.org/tomcat-7.0-doc/index.html (дата обращения: 11.04.2014).
4. Документация Jetty [Электронный ресурс]. URL: http://www.eclipse.org/jetty/documentation/9.1.5.v20140505/ (дата обращения: 06.05.2014).
5. Документация WildFly [Электронный ресурс]. URL: https://docs.jboss.org/author/display/WFLY8/Documentation?_sscc=t (дата обращения: 10.05.2014).
6. Документация GlassFish [Электронный ресурс]. URL: https://glassfish.java.net/documentation.html (дата обращения: 10.05.2014).
7. Документация WebSphere [Электронный ресурс]. URL: http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/as_ditamaps/was855_welcome_ndmp.html?lang=en (дата обращения: 20.05.2014).
8. Документация WebLogic [Электронный ресурс]. URL: http://docs.oracle.com/middleware/1212/wls/index.html (дата обращения: 22.05.2014).
9. Лицензия Apache License 2.0 [Электронный ресурс]. URL: http://www.apache.org/licenses/LICENSE-2.0 (дата обращения: 11.04.2014).
10. Спецификация Java Server Pages 2.1 [Электронный ресурс]. URL: http://download.oracle.com/otn-pub/jcp/jsp-2.1-fr-eval-spec-oth-JSpec/jsp-2_1-fr-spec.pdf (дата обращения: 02.05.2014).
11. Спецификация Java Server Faces 2.2 [Электронный ресурс]. URL: https://jcp.org/en/jsr/detail?id=344 (дата обращения: 02.05.2014).
12. Бунин О. и др. Разработка высоконагруженых систем. М. : Издательство Олега Бунина, 2011. С. 117–133.

DEVELOPMENT OF A SESSION-ORIENTED CONTAINER FOR SERVER-SIDE JAVA-CODE

V.S. Vinogradov
student, e-mail: it.vinogradov@yandex.ru

Omsk State University n.a. F.M. Dostoevskiy

Abstract. In the article we describe the problem of developing a server-side code container which supports the mode of continuous client-server connection. The basic implementation principles of such software technology are also provided.

Keywords: java, network, programming, server, client, sockets.