Распределенные информационно-аналитические системы

Лекция № 2.

Взаимодействие компонент распределенной системы

Профессор кафедры КБ-2: д.т.н. Шатовкин Р.Р.

Учебные цели:

Изучить основы научных знаний по моделям взаимодействия компонент распределенной системы; обмену сообщениями; удаленному вызову процедур; использованию удаленных объектов.

Учебные вопросы:

1. Модели взаимодействия компонент распределенной системы.

2. Обмен сообщениями.

3. Удаленный вызов процедур.

4. Использование удаленных объектов.

1. Модели взаимодействия компонент распределенной системы

Ключевым сервисом промежуточной среды для создания распределенных систем является ***обеспечение обмена данными между компонентами распределенной системы***.

В настоящий момент существуют ***две концепции взаимодействия программных компонент***:

- обмен сообщениями между компонентами;
- вызов процедур или методов объекта удаленной компоненты по аналогии с локальным вызовом процедуры.

Поскольку в настоящее время любое взаимодействие между удаленными компонентами в конечном итоге основано на сокетах *TCP/IP*, первичным с точки зрения промежуточной среды является низкоуровневый обмен сообщениями на основе сетевых сокетов, сервис которых никак не определяет формат передаваемого сообщения.

На базе протоколов *TCP* или *HTTP* затем могут быть построены прикладные протоколы обмена сообщений более высокого уровня абстракции для реализации более сложного обмена сообщениями или удаленного вызова процедур.

Удаленный вызов является моделью, происходящей от языков программирования высокого уровня, а не от реализации интерфейса транспортного уровня сетевых протоколов. Поэтому протоколы удаленного вызова должны обязательно базироваться на какой-либо системе передачи сообщений, включая как непосредственное использование сокетов *TCP*/IP, так и основанные на нем другие промежуточные среды для обмена сообщениями.

Реализация высокоуровневых служб обмена сообщениями, в свою очередь, может использовать удаленный вызов процедур, основанный на более низкоуровневой передаче сообщений, использующей, например, непосредственно сетевые сокеты. Таким образом, одна промежуточная среда может использовать для своего функционирования сервисы другой промежуточной среды, аналогично тому, как один протокол транспортного или сетевого уровня может работать поверх другого протокола при туннелировании протоколов.

2. Обмен сообщениями

Существует ***два метода передачи сообщений от одной удаленной системы к другой***:

- непосредственный обмен сообщениями;
- использование очередей сообщений.

В первом случае передача происходит напрямую, и она возможна только в том случае, если принимающая сторона готова принять сообщение в этот же момент времени.

Во втором случае используется посредник – ***менеджер очередей сообщений***. Компонента посылает сообщение в одну из очередей менеджера, после чего она может продолжить свою работу. В дальнейшем получающая сторона извлечет сообщение из очереди менеджера и приступит к его обработке.

Простейшей реализацией непосредственного обмена сообщениями является использование транспортного уровня сети через интерфейс сокетов, минуя какое-либо промежуточное программное обеспечение. Однако такой способ взаимодействия обычно не применяется в системах автоматизации предприятия, поскольку в этом случае реализация всех функций промежуточной среды ложится на разработчиков приложения. При таком подходе сложно получить расширяемую и надежную распределенную систему, поэтому для разработки прикладных распределенных систем обычно используются ***системы очередей сообщений***.

Существует несколько разработок в области промежуточного программного обеспечения, реализующие высокоуровневые сервисы для обмена сообщениями между программными компонентами. К ним относятся, в частности, *Microsoft Message Queuing*, *IBM MQSeries* и *Sun Java System Message Queue*.

Такие системы дают возможность приложениям использовать следующие базовые примитивы по использованию очередей:

- добавить сообщение в очередь;
- взять первое сообщение из очереди, процесс блокируется до появления в очереди хотя бы одного сообщения;
- проверить очередь на наличие сообщений;
- установить обработчик, вызываемый при появлении сообщений в очереди.

Менеджер очереди сообщений в таких системах может находиться на компьютере, отличном от компьютеров с участвующими в обмене компонентами. В этом случае сообщение первоначально помещается в исходящую очередь на компьютере с посылающей сообщения компонентой, а затем пересылается менеджеру требуемой.

Для создания крупных систем обмена сообщениями может использоваться маршрутизация сообщений, при которой сообщения не передаются напрямую менеджеру, поддерживающему очередь, а проходят через ряд промежуточных менеджеров очередей сообщений (рисунок 1).

Рисунок 1 – Системы очередей сообщений

Использование очередей сообщений ориентировано на ***асинхронный обмен данными***.

Основные достоинства таких систем:

- время функционирования сервера может быть не связано со временем работы клиентов;

- независимость промежуточной среды от средства разработки компонент и используемого языка программирования;

- считывать и обрабатывать заявки из очереди могут несколько независимых компонент, что дает возможность достаточно просто создавать устойчивые и масштабируемые системы.

Недостатки систем очередей сообщений являются продолжением их достоинств:

- необходимость явного использования очередей распределенным приложением;

- сложность реализации синхронного обмена;

- определенные накладные расходы на использование менеджеров очередей;

- сложность получения ответа: передача ответа может потребовать отдельной очереди на каждый компонент, посылающий заявки.

3. Удаленный вызов процедур

Идея ***удаленного вызова процедур*** (***remote procedure call, RPC***) появилась в середине 80-х годов и заключалась в том, что при помощи промежуточного программного обеспечения функцию на удаленном компьютере можно вызывать так же, как и функцию на локальном компьютере.

Чтобы удаленный вызов происходил прозрачно с точки зрения вызывающего приложения, промежуточная среда должна предоставить ***процедуру-заглушку*** (***stub***), которая будет вызываться клиентским приложением.

После вызова процедуры-заглушки промежуточная среда преобразует переданные ей аргументы в вид, пригодный для передачи по транспортному протоколу, и передает их на удаленный компьютер с вызываемой функцией.

На удаленном компьютере параметры извлекаются промежуточной средой из сообщения транспортного уровня и передаются вызываемой функции (рисунок 2).

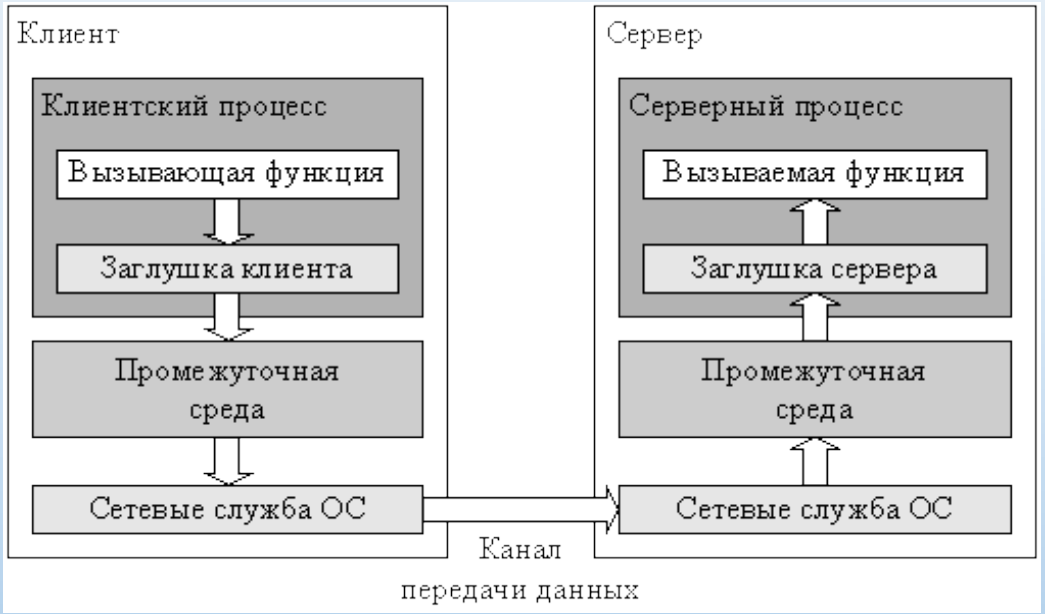

Рисунок 2 – Удаленный вызов процедур

Аналогичным образом на клиентскую машину передается результат выполнения вызванной функции.

Существует ***три возможных варианта удаленного вызова процедур***:

Синхронный вызов: клиент ожидает завершения процедуры сервером и при необходимости получает от него результат выполнения удаленной функции.

Однонаправленный асинхронный вызов: клиент продолжает свое выполнение, получение ответа от сервера либо отсутствует, либо его реализация возложена на разработчика (например, через функцию клиента, удалено вызываемую сервером).

Асинхронный вызов: клиент продолжает свое выполнение, при завершении сервером выполнения процедуры он получает уведомление и результат ее выполнения, например через *callback-функцию*, вызываемую промежуточной средой при получении результата от сервера.

Процесс преобразования параметров для передачи их между процессами (или доменами приложения в случае использования *.NET*) при удаленном вызове называется ***маршализацией (marshalling)***.

Преобразование экземпляра какого-либо типа данных в пригодный для передачи за границы вызывающего процесса набор байт называется ***сериализацией***.

Десериализация (процедура, обратная сериализации) – заключается в создании копии сериализованного объекта на основе полученного набора байт.

Такой подход к передаче объекта между процессами путем создания его копий называется ***маршализацией по значению (marshal by value)***.

Процесс сериализации должен быть определен для всех типов данных, передаваемых при удаленном вызове. К ним относятся параметры вызываемой функции и возвращаемый функцией результат.

В случае передачи параметров по ссылке сериализации подлежат ссылаемые объекты, поскольку для самих указателей сериализация не может быть применена. Это затрудняет использование механизма удаленного вызова в языках, поддерживающих указатели на объекты неизвестного типа.

4. Использование удаленных объектов

В связи с переходом разработчиков прикладных программ от структурной парадигмы к объектной появилась необходимость в использовании ***удаленных объектов*** (***remote method invocation, RMI***).

Удаленный объект представляет собой некоторые данные, совокупность которых определяет его состояние. Это состояние можно изменять путем вызова его методов.

Обычно возможен *прямой доступ* к данным удаленного объекта, при этом происходит неявный удаленный вызов, необходимый для передачи значения поля данных объекта между процессами.

Методы и поля объекта, которые могут использоваться через удаленные вызовы, доступны через некоторый *внешний интерфейс класса объекта*.

Внешний интерфейс компоненты распределенной системы в таких системах обычно совпадает с внешним интерфейсом одного из входящих в компоненту классов.

В момент, когда клиент начинает использовать удаленный объект, на стороне клиента создается клиентская заглушка, называемая ***посредником*** (***proxy***). Посредник реализует тот же интерфейс, что и удаленный объект.

Вызывающий процесс использует методы посредника, который маршализирует их параметры для передачи по сети, и передает их по сети серверу.

Промежуточная среда на стороне сервера десериализует параметры и передает их заглушке на стороне сервера, которую называют ***каркасом*** (***skeleton***) или, как и в удаленном вызове процедур, заглушкой (рисунок 3).

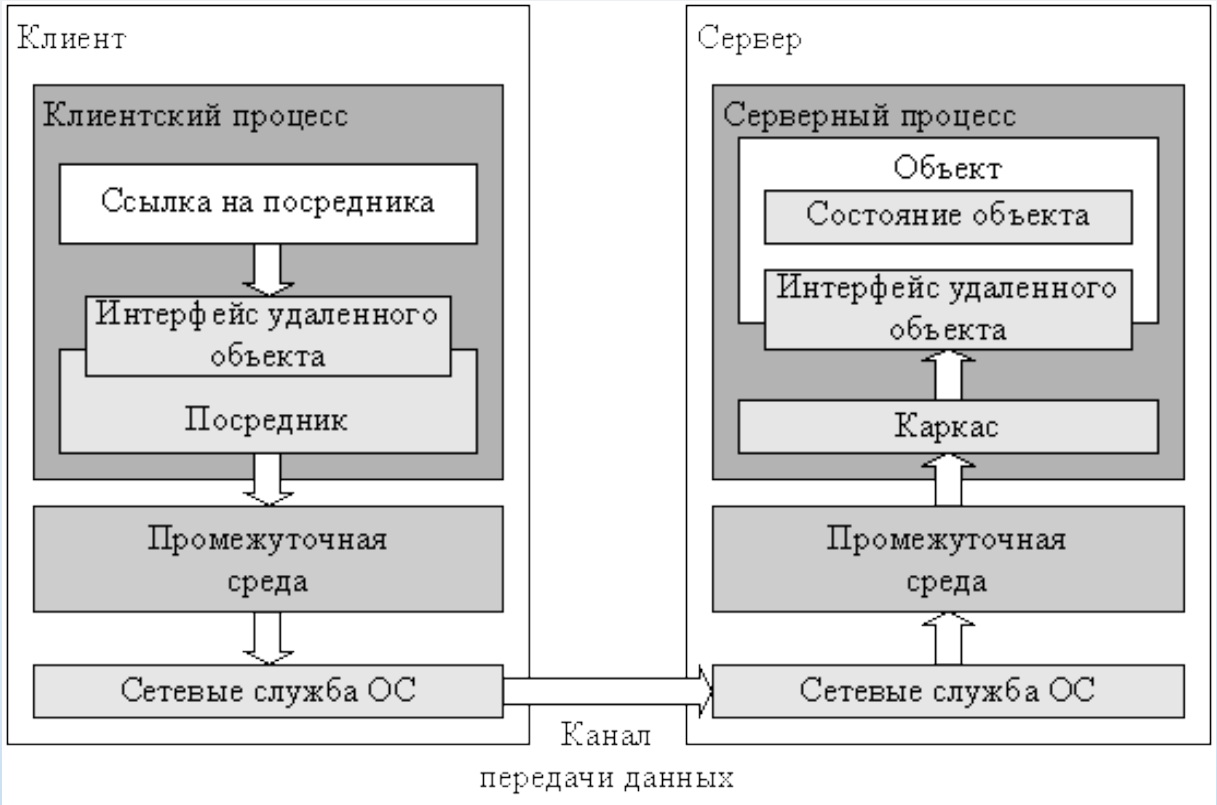

Рисунок 3 – Использование удаленных объектов

Каркас связывается с некоторым экземпляром удаленного объекта. Это может быть как вновь созданный, так и существующий экземпляр объекта, в зависимости от применяемой модели использования удаленных объектов, которые будут рассмотрены ниже.

Весь описанный процесс называется ***маршализацией удаленного объекта по ссылке*** (***marshal by reference***). В отличие от маршализации по значению, экземпляр объекта находится в процессе сервера и не покидает его, а для доступа к объекту клиенты используют посредников. При маршализации же по значению само значение объекта сериализуется в набор байт для его передачи между процессами, после чего следует создание его копии в другом процессе.

Как и удаленный вызов процедур, вызов метода удаленного объекта может быть как ***синхронным***, так и ***асинхронным***.

Существует ***две особенности при использовании удаленных объектов***, не встречавшихся в удаленном вызове процедур.

Во-первых, если на момент формирования концепции удаленного вызова процедур ***исключения (exceptions)*** еще не поддерживались распространенными языками и не использовались, то в дальнейшем они стали методом информирования вызывающей стороны о встреченных вызываемой стороной проблемах. Таким образом, в системах, использующих удаленные объекты, сериализации подвержены как параметры метода и его результат, так и выбрасываемые при выполнении удаленного метода исключения.

Во-вторых, в качестве параметра или результата методов могут тоже передаваться ссылки на удаленный объект (рисунок 4), если вызывающий удаленный метод клиент также может является сервером *RMI*.

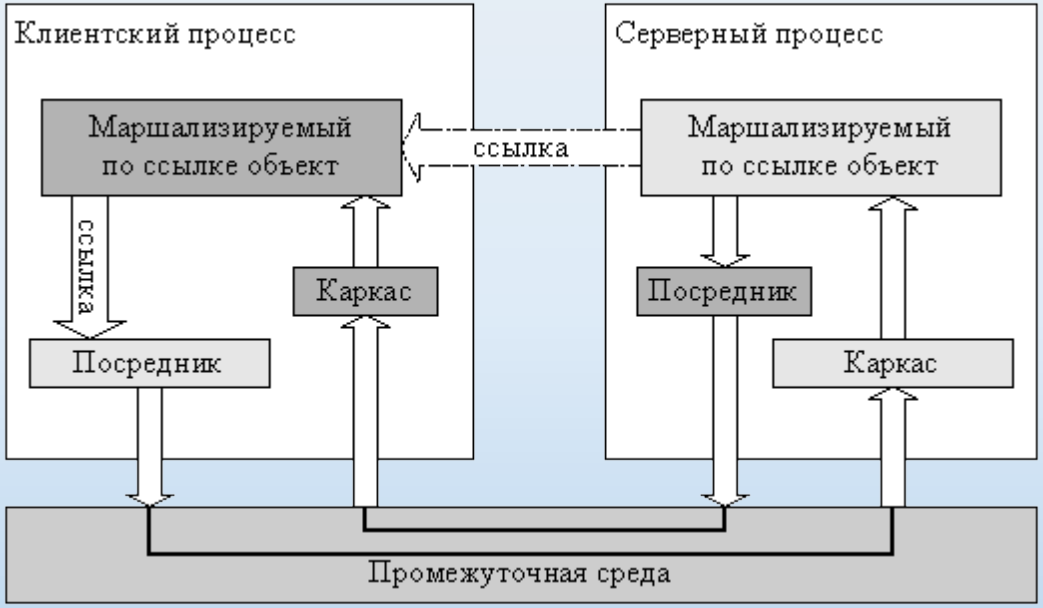

Рисунок 4 – Передача удаленному методу ссылки на объект, маршализируемый по ссылке

При использовании удаленных объектов проблемными являются вопросы о времени их жизни:

- в какой момент времени создается экземпляр удаленного объекта;
- в течение какого промежутка времени он существует.

Для описания ***жизненного цикла*** в системах с удаленными объектами используются два дополнительных понятия:

- активация объекта: процесс перевода созданного объекта в состояние обслуживания удаленного вызова, то есть связывания его с каркасом и посредником;
- деактивация объекта: процесс перевода объекта в неиспользуемое состояние.

Выделяют ***три модели использования удаленных объектов***:

- модель единственного вызова (***singlecall***),
- модель единственного экземпляра (***singleton***),
- модель активации объектов по запросу клиента (***client activation***).

Первые две модели так же иногда называют ***моделями серверной активации (server activation)***, хотя, строго говоря, активация всегда происходит на сервере после какого-либо запроса от клиента.

Модель единственного вызова. При использовании данной модели объект активируется на время единственного удаленного вызова. В простом случае, для каждого вызова удаленного метода объекта клиентом на сервере создается и активируется новый экземпляр объекта, который деактивируется и затем удаляется сразу после завершения удаленного вызова метода объекта. Таким образом, удаленные вызовы разных клиентов изолированы друг от друга. Благодаря удалению объектов после вызова достигается экономное расходование ресурсов памяти, но могут тратиться значительные ресурсы процессора на постоянное создание и удаление объектов. Посредник на клиенте и заглушка на сервере существуют до уничтожения посредника объекта (рисунок 5).

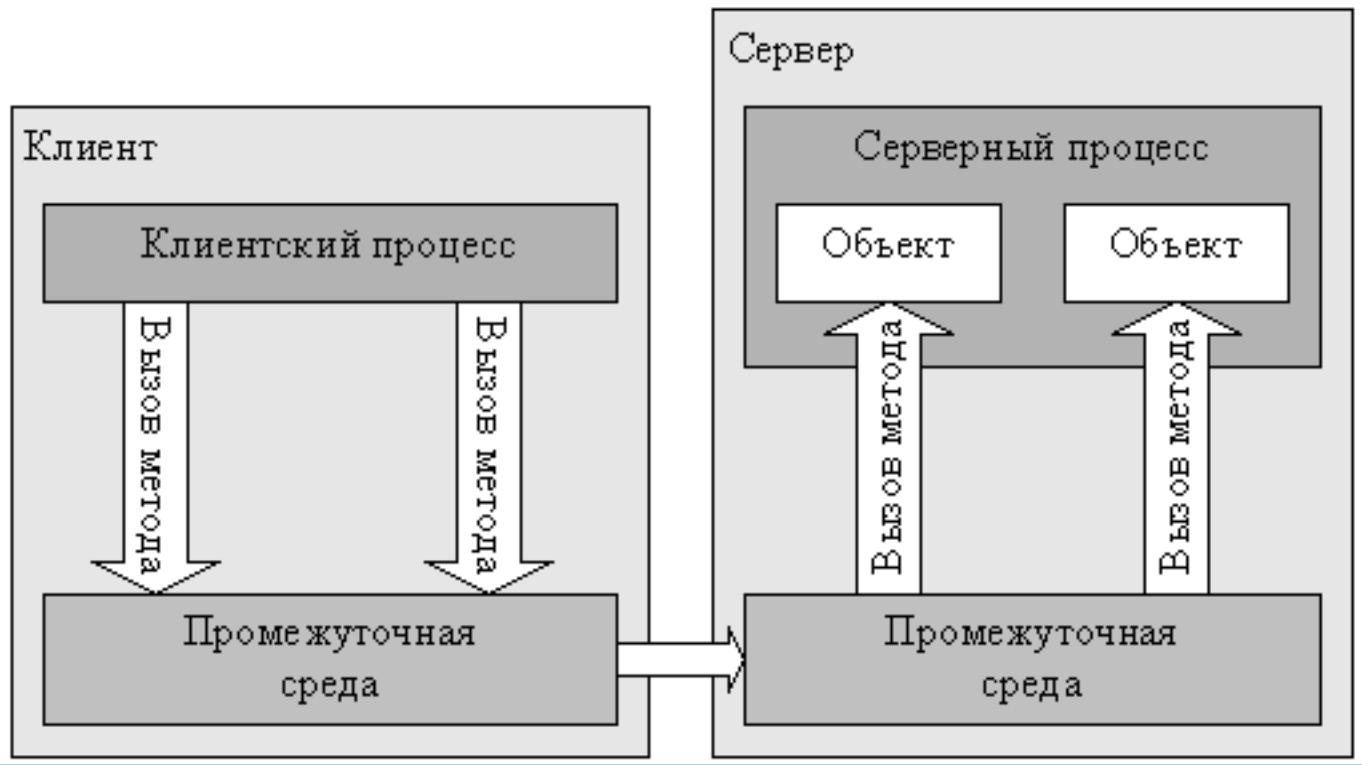

Рисунок 5 – Режим единственного вызова удаленного метода

Данный метод использования удаленных объектов можно рассматривать как некоторый вариант удаленного вызова процедур, поскольку объект не сохраняет свое состояние между вызовами. Тем не менее, сервер использует свои ресурсы на поддержание каркаса и канала между посредником и заглушкой.

Определенным ***недостатком метода одного вызова*** является частое создание и удаление экземпляров объектов, поэтому в промежуточным средах может существовать сервис, позволяющий поддерживать некоторое количество уже созданных, но еще не активированных объектов, которые используются для обработки удаленных вызовов. Такой набор объектов, ожидающих своей активации, называется ***пулом объектов*** (***object pooling***). По завершении удаленного вызова объекты деактивируются и могут либо быть помещены в пул и использованы повторно в дальнейшем, либо удаляются, если размер пула достиг некоторого максимального значения. Такая технология позволяет достичь баланса между скоростью обработки запроса и объемом используемых ресурсов сервера. Как видно из описания, в системах с пулом объектов активация не всегда следует непосредственно после создания объекта, а удаление не всегда следует сразу за деактивацией.

Отличительной особенностью метода одного вызова являются наименьшие затраты на организацию системы балансировки нагрузки и ее наибольшая эффективность, поскольку каждый обслуживающий запросы сервер может обработать вызов любого удаленного метода.

Модель единственного экземпляра. При использовании модели единственного экземпляра удаленный объект существует не более чем в одном экземпляре. Созданный объект существует, пока есть хоть один использующий его клиент (рисунок 6).

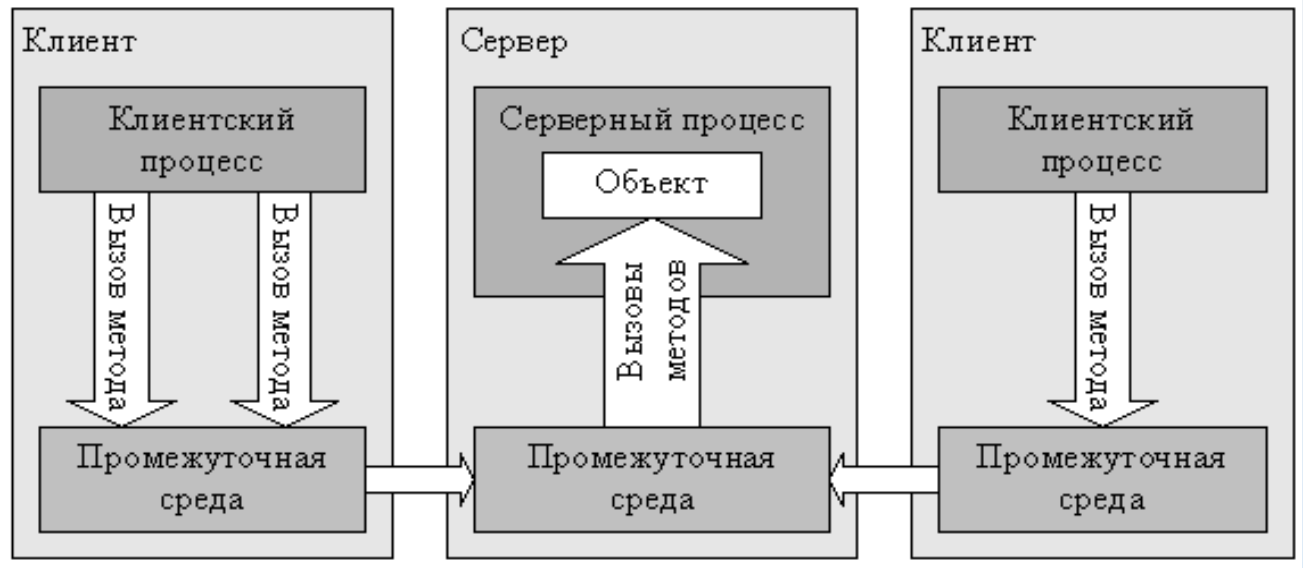

Рисунок 6 – Использование удаленных объектов в режиме единственного экземпляра

При использовании модели единственного объекта вызовы различных клиентов работают с одним и тем же экземпляром удаленного объекта. Поскольку вызовы клиентов не изолированы друг от друга, то используемый объект не должен иметь какого-либо внутреннего состояния. Модель единственного объекта позволяет получить наиболее высокую производительность, поскольку объекты не создаются и не активируются сервером при каждом вызове метода объекта.

Активация по запросу клиента. При каждом создании клиентом ссылки на удаленный объект (точнее, на посредника) на сервере создается новый объект, который существует, пока клиент не удалит ссылку на посредника. При таком методе использования вызовы различных клиентов изолированы друг от друга, и каждый объект сохраняет свое состояние между вызовами, что приводит к наименее рациональному использованию ресурсов памяти сервера (рисунок 7).

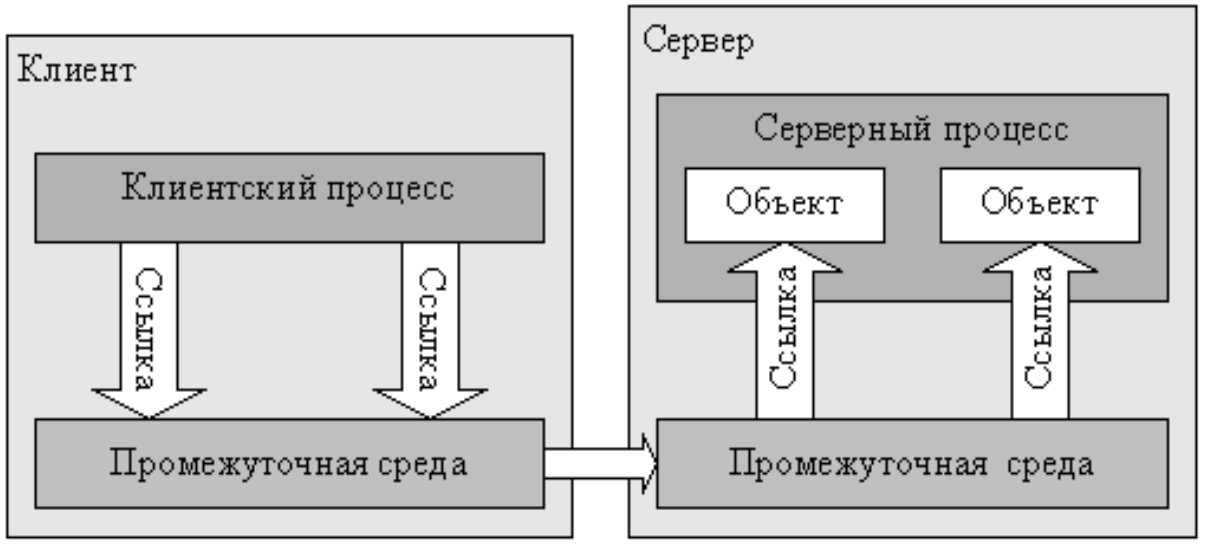

Рисунок 7 – Объекты, активируемые клиентом

***Состояние компоненты распределенной системы*.** Программные компоненты с точки зрения пользователей своих сервисов можно разделить на две категории:

- компоненты без сохраняемого между удаленными вызовами своих методов внутреннего состояния (***stateless components***);

- компоненты с внутренним состоянием, сохраняемым между удаленными вызовами своих методов (***statefull components***).

Под *состоянием* в данном случае понимается совокупность значений полей реализующих компоненту объектов, хранящихся в памяти сервера. Если компонента в ходе своей работы сохраняет какие-либо данные во внешнем хранилище, например в базе данных или очереди сообщений, это обычно не рассматривается как ее внутреннее состояние.

Модель единственного вызова не сохраняет состояния удаленного объекта между вызовами его методов, в силу чего данная модель может использоваться только с распределенными компонентами без внутреннего состояния.

Модель одного экземпляра может быть использована для вызова компонент с внутренним состоянием, но это вряд ли часто имеет смысл, поскольку ее состояние будет меняться каждым из клиентов в произвольном порядке.

Модель активации по запросу клиента может быть использована с любыми компонентами, но для компонент без внутреннего состояния такой подход обычно ведет к непроизводительному расходу памяти при некотором выигрыше в затратах времени процессора по сравнению с моделью одного вызова.

Компоненты без сохранения внутреннего состояния, используемые вместе с моделью единственного вызова с пулом объектов, имеют наибольшие возможности масштабирования системы при оптимальном балансе между затратами памяти и нагрузкой на процессор.

***Использование свойств удаленных объектов*.** Рассмотрим следующий фрагмент кода, заполняющий свойства удаленного объекта информацией о человеке и вызывающий некий удаленный метод для ее обработки:

```
processing.FirstName = "Иван";
processing.SecondName = "Иванов";
processing.ThirdName = "Иванович";
processing.City = "Москва";
result = processing.Run();
```

Если `processing` – локальный объект, то в методе `Run()` он будет иметь доступ к установленным здесь значениям своих методов. Предположим, что `processing` – удаленный объект. В этом случае возможны, например, следующие варианты.

1. Используется модель единственного вызова, причем установка свойства объекта рассматривается как вызов метода установки свойства (`set` в C#). В этом случае к моменту вызова метода `Run` состояние полей объекта не задано (при отсутствии пула объектов) или неопределенно (при использовании пула объектов).

2. Используется модель единственного вызова, но установка свойств объекта не рассматривается как удаленный вызов. Например, их значения сохраняются на стороне клиента и передаются на сервер в момент вызова метода, где на основе этих значений заполняются поля объекта. В этом случае результат будет корректен.

3. Используется модель активации по запросу клиента, или такой вариант модели единственного вызова, в котором присваивание свойств объекта приводит к передаче данных на сервер без последующей деактивации объекта (по существу, такая модель уже не является моделью одного вызова). В этом случае результат так же будет корректен, но присваивание свойств объекту приводит к непроизводительным удаленным вызовам.

4. Используется модель одного экземпляра. В этом случае результат будет некорректным, поскольку поля объекта будут заполняться параллельно разными клиентами.

Таким образом, один и тот же код может давать разные результаты в зависимости от модели использования удаленной компоненты, причем только в одном из примеров был получен корректный результат при минимальных накладных расходах. Можно сделать заключение, что использование в программе свойств удаленного объекта приводит к зависимости ее результата от используемой промежуточной среды и ее конфигурации. Наилучшим вариантом решения этой проблемы представляется отсутствие публичных свойств у удаленного объекта. В таком случае использование удаленного объекта может не отличаться от вызова метода локального объекта. Однако применение такого подхода «в лоб» (так называемый ***chunky design***) может привести к плохо читаемому коду, например к следующему:

```
result = processing.Run("Иванов", "Иван", "Иванович", "Москва");
```

В отличие от предыдущего фрагмента, это код не является самодокументирующимся, и увеличивает вероятность ошибки в последовательности аргументов вызываемого метода. В данном примере так же наблюдается смешивание двух сущностей: объекта, содержащего данные о человеке, и объекта, производящего с ним операции.

Правильный подход заключается в создании маршализируемого по значению класса, содержащего все параметры удаленного вызова, и передача его экземпляра как параметра удаленного вызова метода:

```
person.FirstName = "Иван";
person.SecondName = "Иванов";
person.ThirdName = "Иванович";
person.City = "Москва";
result = processing.Run(person);
```

Для многих задач представляет интерес вопрос модификации поведения удаленного объекта путем добавления некоторого дополнительного кода. Рассмотрим следующий фрагмент кода, вызывающий некий математический метод для заданной функции и заданного отрезка значений аргумента. Было бы удобно отделить реализацию самого математического метода от применяемой функции:

```
task.Epsilon = 1e-5:
task.X1 = -1;
task.X2 = 1;
task.MaximumSteps = 20;
task.Function += X2Function;
result = processing.Run(processingArguments);
...
double X2Function(double x)
{
return x*x;
}
```

Если processing – это удаленный объект, то данный код, вероятно, приведет к многочисленным удаленным вызовам функции X2Function.

В этом случае приведенный код имеет ***два главных недостатка***:

- во-первых, клиент для этого должен так же поддерживать удаленные вызовы;

- во-вторых, он неэффективен, если удаленный метод постоянно вызывает находящуюся на стороне клиента функцию.

Для решения возникшей проблемы в общем случае требуется передать тем или иным способом код функции (и, возможно, используемых ею данных) на сервер с реализующим математический метод удаленным объектом.

Такой подход неприменим в общем случае по соображениям безопасности, но при необходимости можно, например, при использовании .NET Framework передать клиенту сборку с требуемым кодом как данные при удаленном вызове. В качестве возможной альтернативы для математических задач может выступать и передача кода функции для дальнейшей компиляции встроенным компилятором языка C#.

Выводы

В ходе лекции рассмотрены следующие вопросы:

- модели взаимодействия компонент распределенной системы;

- обмен сообщениями;

- удаленный вызов процедур;

- использование удаленных объектов.

Задание на самостоятельную работу

1. Конспект лекций.

Вид и тема следующего занятия

Практическое занятие №2. Основы в ASP.NET Core (ч. 2)