**20. Раскройте значение понятий Сервис и Репозиторий в контексте архитектуры Java приложения и Spring Framework**

**@Component** используется для указания класса в качестве компонента спринг. При использовании поиска аннотаций, такой класс будет сконфигурирован как spring bean.

**@Controller** специальный тип класса, применяемый в MVC приложениях. Обрабатывает запросы и часто используется с аннотацией @RequestMapping.

**@Repository** указывает, что класс используется для работы с поиском, получением и хранением данных. Аннотация может использоваться для реализации шаблона DAO.

**@Service** указывает, что класс является сервисом для реализации бизнес логики (на самом деле не отличается от Component, но просто помогает разработчику указать смысловую нагрузку класса).

Для указания контейнеру на класс-бин можно использовать любую из этих аннотаций. Но различные имена позволяют различать назначение того или иного класса.

В Spring 2.0 и более поздних версиях аннотация @Repository является маркером для любой класс, который выполняет роль или стереотип (также известный как Data Access Object или DAO) репозитория. Среди применений этого маркера является автоматическим переводом исключений.

Spring 2.5 вводит дополнительные аннотации к стереотипам: @Component, @Service и @Controller. @Component - общий стереотип для любого Spring - управляемый компонент. @Repository, @Service и @Controller - специализации @Component для более конкретных случаев использования, для например, в уровнях персистентности, обслуживания и представления, соответственно.

Поэтому вы можете комментировать классы компонентов с помощью @Component, но аннотируя их с помощью @Repository, @Service или @Controller вместо этого ваши классы более подходят для обработки инструментами или связывание с аспектами. Например, эти аннотации для стереотипов сделать идеальные цели для pointcut.

Таким образом, если вы выбираете между использованием @Component или @Service для ваш сервисный уровень, @Service, безусловно, лучший выбор. По аналогии, как указано выше, @Repository уже поддерживается как маркер для автоматический перевод исключений в вашем уровне персистентности.

**21. Раскройте значение понятий Контроллер и Репозиторий в контексте архитектуры Java приложения и Spring Framework**

**@Component** используется для указания класса в качестве компонента спринг. При использовании поиска аннотаций, такой класс будет сконфигурирован как spring bean.

**@Controller** специальный тип класса, применяемый в MVC приложениях. Обрабатывает запросы и часто используется с аннотацией @RequestMapping.

**@Repository** указывает, что класс используется для работы с поиском, получением и хранением данных. Аннотация может использоваться для реализации шаблона DAO.

**@Service** указывает, что класс является сервисом для реализации бизнес логики (на самом деле не отличается от Component, но просто помогает разработчику указать смысловую нагрузку класса).

Для указания контейнеру на класс-бин можно использовать любую из этих аннотаций. Но различные имена позволяют различать назначение того или иного класса.

В Spring 2.0 и более поздних версиях аннотация @Repository является маркером для любой класс, который выполняет роль или стереотип (также известный как Data Access Object или DAO) репозитория. Среди применений этого маркера является автоматическим переводом исключений.

Spring 2.5 вводит дополнительные аннотации к стереотипам: @Component, @Service и @Controller. @Component - общий стереотип для любого Spring - управляемый компонент. @Repository, @Service и @Controller - специализации @Component для более конкретных случаев использования, для например, в уровнях персистентности, обслуживания и представления, соответственно.

Поэтому вы можете комментировать классы компонентов с помощью @Component, но аннотируя их с помощью @Repository, @Service или @Controller вместо этого ваши классы более подходят для обработки инструментами или связывание с аспектами. Например, эти аннотации для стереотипов сделать идеальные цели для pointcut.

Таким образом, если вы выбираете между использованием @Component или @Service для ваш сервисный уровень, @Service, безусловно, лучший выбор. По аналогии, как указано выше, @Repository уже поддерживается как маркер для автоматический перевод исключений в вашем уровне персистентности.

**22. Раскройте значение понятий Контроллер и Модель в контексте архитектуры Java приложения и Spring Framework**

Основы шаблона mvc:

• Model — представление данных, сами данные

• View — представление, вид, отображение

• Controller — управление, связь между моделью и видом.

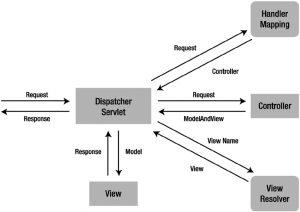
Фреймворк Spring Web model-view-controller (MVC) или по нашему модель-представление-контроллер построен вокруг DispatcherServlet, который распределяет запросы по обработчикам. В нём настраивается мэппинг запросов, локали, временные зоны и многое другое. Обработчик по умолчанию строится на аннотациях @Controller и @RequestMapping, которые предоставляют широкий набор гибких методов для обработки запросов. После версии Spring 3.0. механизм @Controller так же позволяет создавать RESTful веб сайты и приложения, используя аннотацию @PathVariable и другие возможности.

В Spring Web MVC вы можете использовать любой объект в качестве команды или объекта с обратной связью; вам нет необходимости реализовывать какой-либо специальный интерфейс фреймворка или базовый класс. Связывание данных в Spring является очень гибким: например, оно рассматривает несоответствие типов как ошибки валидации и поэтому это может быть обработано в приложении, а не в качестве системных ошибок. Таким образом, вам не нужно дублировать свойства бизнес-объектов, в качестве простых нетипизированных строк для ваших объектов форм. Поэтому можно легко обрабатывать неправильные подтверждения (сабмиты) или правильно конвертировать их в строки. Вместо этого, желательно связывать такие объекты напрямую с объектами бизнес логики.

Spring MVC построен вокруг центрального сервлета, который распределяет запросы по контроллерам, а также предоставляет другие широкие возможности при разработке веб приложений. На самом деле DispatcherServlet — полностью интегрированный сервлет в Spring IoC контейнер и таким образом получает доступ ко всем возможностям Spring.

Обработка запросов в DispatcherServlet показана на рисунке ниже. Используется паттерн «pattern-savvy reader», который распознает DispatcherServlet как выражение из шаблона проектирования «Front Controller».

Ключевым интерфейсом в Spring MVC является Controller. Контроллер обрабатывает запросы к действиям, осуществляемые пользователями в пользовательском интерфейсе, взаимодействуя с уровнем обслуживания, обновляя модель и направляя пользователей на соответствующие представления в зависимости от результатов выполнения. Controller — управление, связь между моделью и видом



Основным контроллером в Spring MVC является org.springframework.web.servlet.DispatcherServlet. Задается аннотацией @Controller и часто используется с аннотацией @RequestMapping, которая указывает какие запросы будут обрабатываться этим контроллером.

**23. Объясните значение термина "Непрерывная интеграция"**

Все мы прекрасно понимаем, что тестирование является неотъемлемой частью жизненного цикла разработки ПО. Чем чаще мы тестируем наш код, тем быстрее мы сможем обнаружить ошибку, вкравшуюся в него в ходе разработки, и быстрее её исправить. При этом стоит понимать, что тестирование крайне желательно проводить в окружении, максимально близком к боевому (ОС, ПО, Hardware, Нагрузка), что бы иметь возможность обнаружить ошибки, которые не проявляются на сервере разработки, но могут появиться в бою. Компануя два вышесказанных тезиса вместе мы получаем концепцию, называемую Continuous Integration.

Суть CI заключается в постоянной (например, после каждого commit'а) сборке и тестировании разрабатываемого ПО в максимально приближенной к боевой среде с целью как можно более раннего обнаружения ошибок и оповещения о них разработчиков.

Для автоматизации процесса непрерывной сборки существуют готовые решения (Hudson, CruiseControl), интеграцию одного из которых (Hudson)

*Задача*

И так, допустим у нас есть два проекта: Java-сервис (со своей БД), и PHP-клиент (со своей БД) для него. Оба проекта распространяются в виде deb-пакетов. Необходимо настроить инфраструктуру непрерывной интеграции этих проектов.

*Реализация*

Для того, что бы иметь представление о том, чего же мы хотим в итоге добиться — начнём с конца: рассмотрим схему, которую мы хотим реализовать:

- рабочая машина программиста — написание кода,

- сервер SVN — хранение кода,

- сервер Staging — установка и тестирование собранных проектов,

- сервер Selenium — тестирование web-интерфейса,

- сервер Repo — хранение собранных пакетов,

- сервер CI — соединение всех узлов системы в единое целое.

Разработчик вносит изменения в проект на своей машине и commit'ит их в SVN. На сервере SVN срабатывает post-commit hook, который инициирует процесс build'а соответствующего проекта на сервере CI. Сервер CI обновляет версию пакета из SVN, компилирует проект, запускает unit-тесты, выкладывает проект на staging-сервер.

Для проектов без web-интерфейса запускаются интеграционные тесты, для проектов с web-интерфейсом запускаются тесты Selenium. Сервер CI формирует отчёты и при необходимости (в случае провала на любом из этапов сборки проекта) отправляет email-уведомление пользователю.

Выкладка пакетов проекта в репозитарий для боевых серверов производится разработчиком вручную во время релиза.

**24. Опишите процесс внедрения (deploy) Java приложения**

Перед установкой приложения нужно подготовить сервер приложений.

Вы можете подготовить новый сервер приложений, или повторно использовать существующий сервер приложений.

Убедитесь, что вы установили все необходимые программы и что у вас настроена нужная среда.

Задайте для утилиты базы данных для DB2 или Oracle ваш системный путь. В случае Microsoft SQL Server этот шаг не требуется.

Проверьте рабочий лист конфигурирования сервера приложений, чтобы понять основные параметры, которые вас попросят ввести в процессе установки. Заполнение рабочего листа дает гарантии того, что у вас будет базовая информация, необходимая для выполнения установки. Если вам требуется несколько экземпляров, скопируйте рабочую таблицу и подготовьте отдельную рабочую таблицу для каждого внедрения.

Используйте ID wasadmin при подготовке сервера приложений. Если вы используете DB2, этот ID должен являться владельцем клиента DB2 или экземпляра DB2. Этот ID должен быть членом группы mqm в WebSphere MQ, если вы используете WebSphere MQ для обмена сообщениями. Эта группа используется для администрирования WebSphere MQ.

Убедитесь, что вы задали переменную WAS\_HOME и путь Java™ JAVA\_HOME для IBM® WebSphere Application Server.

Убедитесь, что у вас нет сервера с именем server или кластера с именем cluster.

Для аргументов размера кучи JVM WebSphere Application Server Deployment Manager (Dmgr) должны быть заданы значения 512MB и 1024MB. Чтобы увеличить размер кучи:

1. Откройте консоль WebSphere Application Server Integrated Solutions Console и перейдите в Администрирование системы > Менеджер внедрения.
2. Раскройте в разделе Инфраструктура сервера Java и управление процессами и щелкните по Определение процесса.
3. В разделе Дополнительные свойства выберите опцию Виртуальная машина Java.
4. Задайте для параметра Начальный размер кучи 512 МБ, а для параметра Максимальный размер кучи - 1024 МБ.
5. Щелкните по OK, сохраните изменения и синхронизируйте изменения с узлами.

**25. Объясните для чего нужны SSH ключи. Что означает количество бит в ssh ключе**

Использование SSH-ключей —простой и надёжный способ  обеспечения безопасности соединения с сервером.  В отличие от пароля, взломать SSH-ключ практически невозможно. Сгенерировать SSH-ключ очень просто.

Ключи SSH служат средством идентификации вас при подключении к серверу SSH с использованием криптосистемы с открытым ключом и аутентификации вызов-ответ. Одним из непосредственных достоинств этого метода перед традиционной идентификацией с помощью пароля является то, что вы можете быть авторизованы на сервере без регулярной необходимости отсылать ваш пароль через сеть. Даже если кто-либо будет прослушивать ваше соединение, у него не будет возможности перехватить и взломать ваш пароль, поскольку фактически он никогда не передается. Также использование для идентификации ключей SSH устраняет риск, связанный с брут-форс (brute-force) атаками, за счет существенного уменьшения шанса атакующего угадать правильные учетные данные.

Идентификация при помощи ключей SSH предоставляет дополнительную безопасность, а также может быть более удобным способом, чем традиционная идентификация при помощи пароля. При использовании вместе с программой, называемой агентом SSH, ключи SSH могут подключать вас к серверу или нескольким серверам без необходимости помнить и вводить ваш пароль отдельно для каждой системы.

У метода использования ключей SSH есть определенные недостатки, и он подходит не для всех, но при этом во многих случаях он может предложить несколько существенных преимуществ. Общее понимание того, как работают ключи SSH, поможет вам решить, как и когда применять их для удовлетворения ваших потребностей. Если вы собираетесь читать эту статью, предполагается, что вы уже знаете основы работы протокола Secure Shell и установили пакет openssh.

Что означает количество бит в ssh ключе

чем больше количество битов в ключе, тем сложнее взломать его при помощи подбора

**26. Опишите один из способов защиты HTTP сервисов от неавторизованного доступа**

Основной задачей любой интеграции, будь то КД, Web-сервисы или HTTP-сервисы, является что-нибудь откуда-нибудь передать, что-нибудь с этим сделать и вернуть ответ. Вот в таком формате мы и рассмотрим новую технологию.

все данные, которые вы хотите получить извне, можно разделить на 2 блока - обязательные и не обязательные.

**Обязательные данные/параметры** запихиваем в шаблон, тем самым если тот, кто обращается к сервису, их не заполнил, то сервис априори выдаст ошибку, а вы при разработке текста модуля обработчика будете уверены, что эти данные есть. Как это делается: в строке Шаблон в фигурных скобках "{}", чередуя с со знаком "/", пишем имена переменных. Например, нам обязательно нужен артикул - тогда пишем /{artikul}. Если нам надо получить артикул, имя и имя пользователя, строка шаблона будет выглядеть так: /{artikul}/{name}/{user} и т.д. Каждый из таких параметров в тексте модуля обработчика можно будет получить так: Запрос.ПараметрыURL["<имя параметра>"]. Если обязательных нет, то шаблон выглядит так: /\*.

**Не обязательные данные**, которые мы хотим получать через сервис, в шаблоне НЕ описываются. При построении ссылки, для обращения к сервису они описываются в конце ссылки после знака "?", разделяются символом амперсанда "&" и имеют структуру <имя параметра>=<значение параметра>. В тексте модуля обработчика к ним можно обратиться конструкцией: Запрос.ПараметрыЗапроса.Получить("<имя параметра>"). НО: важно помнить, раз они не обязательны, то их может и не быть, соответственно значение проверяем на Неопределено.

Обязательные Headers для всех запросов.

|  |  |
| --- | --- |
| **Ключ** | **Значение** |
| **x-date** | Текущая дата в Unix timestamp формате  Пример:  1449738039 |
| **x-authorization** | Значение создается из двух частей. **API Key** (пробел) (двоеточие) (пробел) **Сигнатура запроса**  {API Key} : {Сигнатура запроса}  Пример:  2mtLK4+pSfym6M5LBqIRk5v1c6z1gNXcwWo4KaOt+7A= : /kPUnSNORNiaBzmr8Bwbr3QNlp0=  Описание генерации сигнатуры запроса будет ниже. |
| **content-type** | **application/json** |
| **cache-control** | **no-cache** |

**Сигнатура запроса**

Каждый запрос на сервер должен проходить с сигнатурой.

Сигнатура каждого запроса генерируется следующим способом:

1. Получить **toDigest**

**toDigest = {Метод запроса}**aA**{MD5(тело запроса)}**Bb**{content-type запроса}**CC**{x-date запроса}**dd**{URL запроса}**

**Пример:**

**toDigest = POSTaAab57b4796adb1bbe36488f205eee53f5Bbapplication/jsonCC1449738039ddhttps://api.focuson.kz/v1/cardsinfo**

Если запрос GET, то в качестве MD5(тело запроса) берется MD5(пустой строки)

**2.** Затем необходимо сгенерировать хеш-код на основе **Private Key**, используя метод **HMAC** алгоритмом **SHA1.**

**hashHmacSha1 = hash\_hmac("sha1", toDigest, Private Key)**

Результат этой операции строка в шестнадцатеричной кодировке:

**hashHmacSha1 = fe43d49d234e44d89a0739abf01c1baf740d969d**

**3.** Пропустить значение **hashHmacSha1** через метод **hexToBase64()**

private function hexToBase64($hex)

{

$return = "";

// взять по паре символов из строки, получить его десятичный код и получив символ по этому коду, конкатенировать в строку

foreach(str\_split($hex, 2) as $pair)

{

$return .= chr(hexdec($pair));

}

return base64\_encode($return); // перевести результат в base64

}

Результат этого метода и есть сигнатура запроса: **/kPUnSNORNiaBzmr8Bwbr3QNlp0=**

**32. Ниже приведен пример объекта, описанного в формате XML. Покажите как этот же объект может выглядеть в формате JSON.**

**<person>**

**<firstName>Linus</firstName>**

**<lastName>Torvalds</lastName>**

**<address>**

**<streetAddress>Freedom Circle</streetAddress>**

**<city>Santa Clara</city>**

**<postalCode>95054</postalCode>**

**</address>**

**<phoneNumbers>**

**<phoneNumber>+1 (408) 327 9830 x328</phoneNumber>**

**<phoneNumber>+1 (408) 327 9830 x329</phoneNumber>**

**</phoneNumbers>**

**</person>**

*json*

{

"person": {

"firstName": "Linus",

"lastName": "Torvalds",

"address": {

"streetAddress": "Freedom Circle",

"city": "Santa Clara",

"postalCode": "95054"

},

"phoneNumbers": {

"phoneNumber": [

"+1 (408) 327 9830 x328",

"+1 (408) 327 9830 x329"

]

}

}

}

**33. Ниже приведен пример объекта, описанного в формате XML. Покажите как этот же объект может выглядеть в формате JSON.**

**<breakfast\_menu>**

**<food>**

**<name>Тосты</name>**

**<price>300</price>**

**<description>**

**Французские тосты**

**</description>**

**<calories>650</calories>**

**</food>**

**<food>**

**<name>Манты</name>**

**<price>1000</price>**

**<description>**

**Манты с тыквой и с мясом**

**</description>**

**<calories>900</calories>**

**</food>**

**</breakfast\_menu>**

{

"breakfast\_menu": {

"food": [

{

"name": "Тосты",

"price": "300",

"description": "

Французские тосты

",

"calories": "650"

},

{

"name": "Манты",

"price": "1000",

"description": "

Манты с тыквой и с мясом

",

"calories": "900"

}

]

}

}

**34. Объясните, почему для обмена данных между информационными системами часто используют форматы обмена данными (например XML или JSON), а не передают просто текст или массив байтов.**

JSON — простой формат файлов, который легко читается любым языком программирования. Простота означает, что компьютерам с ним легче работать, чем с другими, например, с XML.

XML

XML широко используется для обмена данными, поскольку позволяет сохранять структуру в данных и способе их сохранения в файлах, а также позволяет разработчикам хранить части документации вместе с данными, не мешая их чтению.

Формат XML уже известен много лет (прим.: первая рабочая версия [*была заявлена в 1996 году*](http://www.w3.org/TR/WD-xml-961114), а [*спецификация — уже в 2000*](http://www.w3.org/TR/2000/REC-xml-20001006)), поэтому существует некоторый набор программных интерфейсов (API) для привязки данных к XML на нескольких языках программирования. Например, на Java можно использовать [JAXB](http://java.sun.com/webservices/jaxb/index.jsp) и [XmlBeans](http://xmlbeans.apache.org/) для создания XML-ответа.

С другой стороны, все интерфейсы для создания ответа на JSON появились относительно недавно. Тем не менее, на [JSON.org](http://www.json.org/) опубликован довольно впечатляющий их список на различных языках.

## Простота использования

На стороне клиентского приложения обработка JSON-данных как ответа на XMLHttpRequestчрезвычайно проста.

var person = eval(xhr.responseText);

alert(person.firstName);

Используя обычный eval(), можно преобразовать ответ в объект JavaScript. Как только эта операция произведена, можно получить доступ к данным, используя свойства преобразованного объекта. Это наиболее изящная часть всего JSON.  
  
Теперь рассмотрим XML. Чтобы сделать нижеприведенный фрагмент кода прозрачнее, я убрал все проверки на ошибки.

var xml = xhr.responseXML;

var elements = xml.getElementsByTagName(<font color="#069">"firstName"</font>);

alert(elements[<font color="#c00000">0</font>].firstChild.textContent);

Очевидно, что при обработке данных, полученных от сервера, необходимо просмотреть все DOM-дерево. Это очень трудоемкая операция, и она предрасположена к появлению ошибок. К несчастью, в браузере нам приходится иметь дело именно с DOM. Браузеры не поддерживают языка запросов, наподобие XPath, для получения узлов дерева в XML-документе. Поддержка этих функций относится уже к XSLT, но он достаточно ограничен (прим.: в браузере) в плане преобразования XML в разметку (например, в HTML). [Рабочая группа по программным Веб-интерфейсам](http://www.w3.org/2006/webapi/) (Web API Working Group) от W3C работает над [интерфейсом селекторов](http://www.w3.org/TR/selectors-api/) (Selectors API), который может быть использован для применения CSS-селекторов при выборе узлов из объекта Document. Используя такой интерфейс можно будет преобразовать вышеприведенный пример кода в xml.match("person.firstName"), чтобы получить элемент firstName. Не сказать, что это большое достижение для XML-документа из этого примера, но может оказаться полезным для работы с сильно разветвленными документами. Этот интерфейс пока еще не завершен, и пройдут еще годы, прежде чем браузеры будут его поддерживать.  
  
В общем, если я буду выбирать между XML и JSON, я предпочту JSON из-за простоты реализации обработки на стороне клиента.

**35. Вы разрабатываете платежную информационную систему. Опишите архитектуру процесса регистрации (API методы, формат обмена данными и модели)**