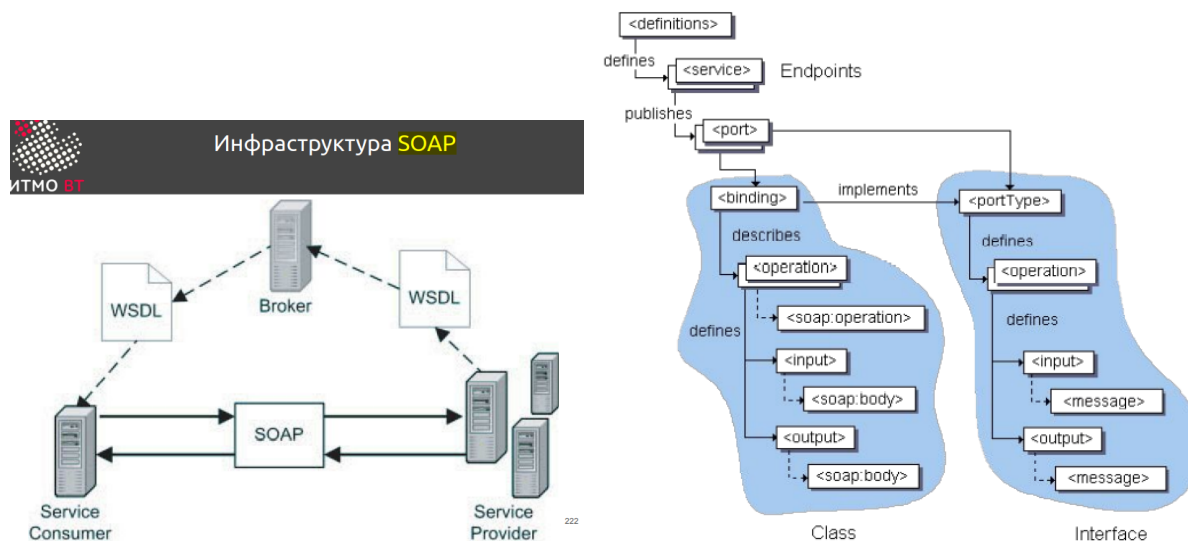


Вариант 1

1. Все про SOAP

SOAP - формат обмена сообщениями

- Протокол для разработки веб-сервисов.
- Базируется на идеологии RPC.
- Стандартизирован W3C.
- Есть реализации “по умолчанию” для различных платформ.
- Предполагает использование инфраструктурного ПО – реестров и сервисных шин.



Структура документа WSDL:

- Types (определение типов данных) — определение вида отправляемых и получаемых сервисом XML-сообщений.
- Message (элементы данных) — сообщения, используемые web-сервисом
- PortType (абстрактные операции) — список операций, которые могут быть выполнены сообщениями.
- Binding (связывание сервисов) — способ, которым сообщение будет доставлено.

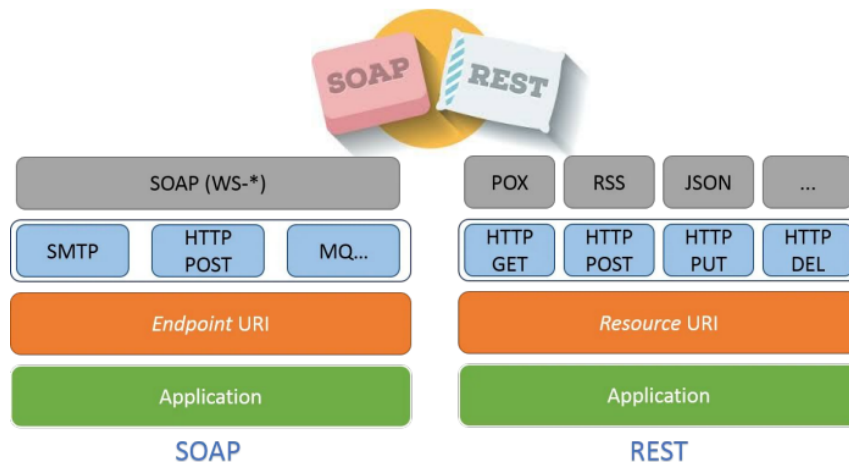
SOAP всегда содержит 3 уровня в структуре (+1 дополнительный):

— Envelope (конверт) – корневой элемент, который определяет сообщение и пространство имен, использованное в документе,
— Header (заголовок) – содержит атрибуты сообщения, например: информация о безопасности или о сетевой маршрутизации,
— Body (тело) – содержит сообщение, которым обмениваются приложения,
— Fault – необязательный элемент, который предоставляет информацию об ошибках, которые произошли при обработке сообщений. И запрос, и ответ должны соответствовать структуре SOAP.

SOAP: Позволяет специфицировать интерфейсы веб-сервисов.

Особенности:

- Основан на XML, является расширением стандарта XML-RPC.
- Обычно работает “поверх” http.
- Обычно используется совместно с дескрипторами веб-сервисов.



WSDL: Web Services Description Language – язык спецификации SOAP веб-сервисов.

- Базируется на XML.
- Описывает весь интерфейс сервиса:
 - функции;
 - аргументы;
 - возвращаемые значения.
- Может автогенерироваться по API сервиса, или, наоборот – API сервиса может автогенерироваться по WSDL.

Пример запроса и ответа:

```
POST /Quotation HTTP/1.0
Host: www.xyz.org
Content-Type: text/xml; charset = utf-8
Content-Length: nnn

<?xml version = "1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
  SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

  <SOAP-ENV:Body xmlns:m = "http://www.xyz.org/quotations">
    <m:GetQuotation>
      <m:QuotationsName>Microsoft</m:QuotationsName>
    </m:GetQuotation>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
HTTP/1.0 200 OK
Content-Type: text/xml; charset = utf-8
Content-Length: nnn

<?xml version = "1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
  SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

  <SOAP-ENV:Body xmlns:m = "http://www.xyz.org/quotation">
    <m:GetQuotationResponse>
      <m:Quotation>Here is the quotation</m:Quotation>
    </m:GetQuotationResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Преимущества:

- Есть чёткая спецификация.

- Есть готовые инфраструктурные решения.
- Удобен для RPC-систем.

Отличие от REST: SOAP ограничивает структуры ваших сообщений, тогда как REST — это архитектурный подход, ориентированный на использование HTTP в качестве транспортного протокола.

2. Все про самоподписанные сертификаты

В криптографии под самоподписанным SSL сертификатом понимают сертификат открытого ключа, изданный и подписанный тем же лицом, которое он идентифицирует. Проще говоря, если Вы сами для своего домена или IP-адреса создали SSL сертификат он будет называться самоподписанным. Также существуют другие названия: «самоизданный» или «самозаверенный», что является одним и тем же.

- Сертификат, выданный самим его субъектом.
- Не может быть отозван.
- Технически, все сертификаты СА являются самоподписанными.
- СА - Центр сертификации, удостоверяющий центр (Certification Authority, CA) -- организация, "чья честность неоспорима, а открытый ключ широко известен" (С).
- Подтверждает подлинность ключей шифрования своим сертификатом.
- Обычно сертификаты объединяются в цепочки.

3. Реализовать rest сервис на jaxrs для работы с корзиной покупок. Формат данных - xml

```
@Data
@XmlRootElement(name="products")
@XmlAccessorType(XmlAccessType.FIELD)
public class Product {
    @XmlElement(required=true)
    protected int id;
    @XmlElement(required=true)
    protected String name;
    @XmlElement(required=true)
    protected String description;
    @XmlElement(required=true)
    protected int price;
}
```

```

@Data
@XmlRootElement(name="basket")
@XmlAccessorType(XmlAccessType.FIELD)
public class Basket {

    @XmlElement(required=true)
    protected int id;

    @XmlElement(required=true)
    protected List<Product> products;
}

@Path("/baskets")
public class BasketService {

    private Repository rep = new Repository();

    @POST
    @Consumes("application/xml")
    public Basket createBasket() {
        return Mapper.mapBasket(rep.createBasket());
    }

    @GET
    @Consumes("application/xml")
    public Basket getBasket() {
        return Mapper.mapBasket(rep.getBasket());
    }

    @PUT
    @Path("/{id}")
    @Consumes("application/xml")
    public Basket addProduct(@PathParam("id") Long id, Product prod) {
        rep.addProduct(id, Mapper.mapToEntityProduct(prod));
        return getProduct();
    }
}

```

```

    @POST
    @Path("{id}/products/{prodId}")
    @Consumes("application/xml")
    public void addProduct(@PathParam("id") Long id, @PathParam("prodId")
Long prodId) {
        rep.addProduct(id, prodId);
    }

    @DELETE
    @Path("{id}/products/{prodId}")
    @Consumes("application/xml")
    public void deleteProduct(@PathParam("id") Long id,
@PathParam("prodId") Long prodId) {
        rep.deleteProduct(id, prodId);
    }

    @DELETE
    @Path("{id}")
    @Consumes("application/xml")
    public void deleteBasket(@PathParam("id") Long id) {
        rep.deleteBasket(id);
    }
}

```

Вариант 2

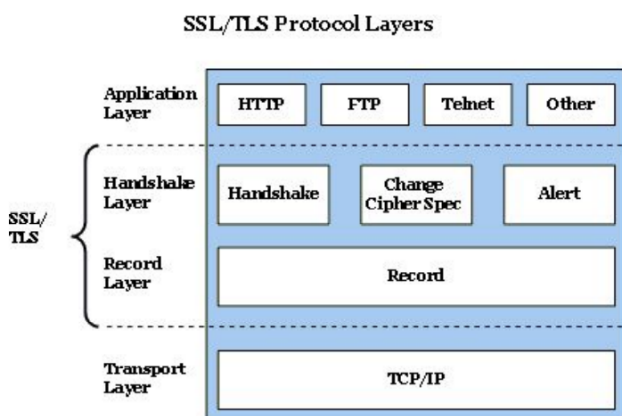
1. Правила именования RESTful ресурсов

- URL формируются иерархически.
- Управляемые сущности именуются во множественном числе.
- Обращение без параметра возвращает массив объектов.
- Обращение с ИД возвращает конкретный объект.
- Сложносоставные слова рекомендуется заменять иерархией.
- Семантика осуществляемого действия располагается в методе, а не в URL.

<http://www.example.com/customers/12345/invoices>

- Получить список поставщиков:
GET <http://www.example.com/customers>
- Добавить нового поставщика:
POST <http://www.example.com/customers>
- Получить поставщика с ИД=12345:
GET <http://www.example.com/customers/12345>
- Обновить данные о поставщике с ИД=12345:
PUT <http://www.example.com/customers/12345>
- Получить все заказы поставщика с ИД=12345:
GET <http://www.example.com/customers/12345/orders>

2. TLS/SSL в иерархии протоколов



SSL (Secure Sockets Layer) и TLS (Transport Level Security)

Протокол SSL размещается между двумя протоколами (работает фильтром, защищая данные): 1) протоколом, который использует программа-клиент (напр., HTTP) и 2) транспортным протоколом TCP/IP

Работу протокола SSL можно разделить на два уровня:

- Слой протокола подтверждения подключения (Handshake Protocol Layer), который состоит из трех подпротоколов:
 - Протокол подтверждения подключения (Handshake Protocol) - цепочка обмена данными, для начала аутентификации сторон и согласования шифрования
 - Протокол изменения параметров шифра (Cipher Spec Protocol) - для изменения данных ключа
 - Предупредительный протокол (Alert Protocol) - содержит сообщение, которое показывает сторонам изменение статуса или сообщает о возможной ошибке
- Слой протокола записи (Record Protocol Layer) - протокол:
 - принимает сообщения, которые нужно передать,
 - фрагментирует данные в управляемые блоки,
 - разумно сжимает данные, применяя MAC (message authentication code),
 - Шифрует

- передаёт результат.

3. Открыть из репозитория спринг дата рест все не публичные методы

```
/*
 * spring.data.rest.detection-strategy=annotation
 */
@Repository
public interface UserRepository extends Repository<User, Long> {
    @RestResource(exported = false)
    public User save(User user);
    @RestResource(exported = false)
    public Optional<User> findById(Long id);
    @RestResource
    Optional<User> findByUsername(String username);
    @RestResource
    void deleteByCityName(String cityName);
}
```

Вариант 3

1. Монолитная архитектура: особенности, достоинства, недостатки

Монолитное приложение состоит из базы данных, клиентского пользовательского интерфейса, серверного приложения

Плюсы:

Простое развертывание. Использование одного исполняемого файла или каталога упрощает развертывание.

Разработка. Приложение легче разрабатывать, когда оно создано с использованием одной базы кода.

Производительность. В централизованной базе кода и репозитории один интерфейс API часто может выполнять ту функцию, которую при работе с микросервисами выполняют многочисленные API.

Упрощенное тестирование. Монолитное приложение представляет собой единый централизованный модуль, поэтому сквозное тестирование можно проводить быстрее, чем при использовании распределенного приложения.

Удобная отладка. Весь код находится в одном месте, благодаря чему становится легче выполнять запросы и находить проблемы.

Минусы:

- Кодовая база со временем становится громоздкой

С течением времени большинство продуктов продолжают разрабатываться и увеличиваются в объеме, а их структура становится размытой. В этот момент и масштабирование становится сложным (так как нельзя масштабировать отдельные части вашей системы)

- Сложно внедрять новые технологии

Добавление новой технологии означает переписывание всего приложения, что является дорогостоящим и требует много времени.

+/- по SOA по сравнению с монолитной архитектурой:

Достоинства:

- Декомпозиция модулей.
- Можно использовать в разных модулях разные технологии.
- Можно модернизировать модули независимо друг от друга.
- (Теоретически) лучшая масштабируемость.
- Удобная интеграция “из коробки”.

Недостатки:

- Усложнение архитектуры.
- Система теряет целостность.
- Сложнее тестировать.
- Сложнее поддерживать

2. Стратегии экспорта в репозиториях Spring Data Rest.

Наименование	Описание
DEFAULT	Открывает наружу все публичные интерфейсы репозитория, но учитывает флаг <code>exported</code> в аннотациях <code>@(Repository) RestResource</code> .
ALL	Открывает наружу все интерфейсы репозитория без учёта модификаторов доступа и аннотаций.
ANNOTATION	Открывает наружу только ресурсы, помеченные аннотациями <code>@(Repository) RestResource</code> с учётом значения флага <code>exported</code> .
VISIBILITY	Открывает наружу только публичные аннотированные ресурсы.

Для Spring Boot:

Настройка происходит путём добавления строки в `application.properties`:

`spring.data.rest.detection-strategy=visibility`

P.S. флаг `exported` ставится в аннотации `@RestResource` над методом репозитория:

```
@RestResource(exported = false)
```

```
void deleteById(Long aLong);
```

Или над самим репозиторием, если необходимо скрыть все его методы:

```
@RepositoryRestResource(exported = false) was
```

```
interface PersonRepository extends CrudRepository<Person, Long> {}
```

3. Дан Spring Data репозиторий с одним методом. Превратить его в Spring Data Rest и сделать так, чтобы в интернету торчал только этот метод

Добавляем в `application.properties`:

```
spring.data.rest.detection-strategy=visibility
```

Далее пишем репозиторий:

```
@RepositoryRestResource(collectionResourceRel = "car", path = "cars")
```

```
public interface CarRepository extends Repository<Car, Long> {  
    public Car save(Car car);  
}
```

Вариант 4

1. Особенности разработки RESTful на Spring

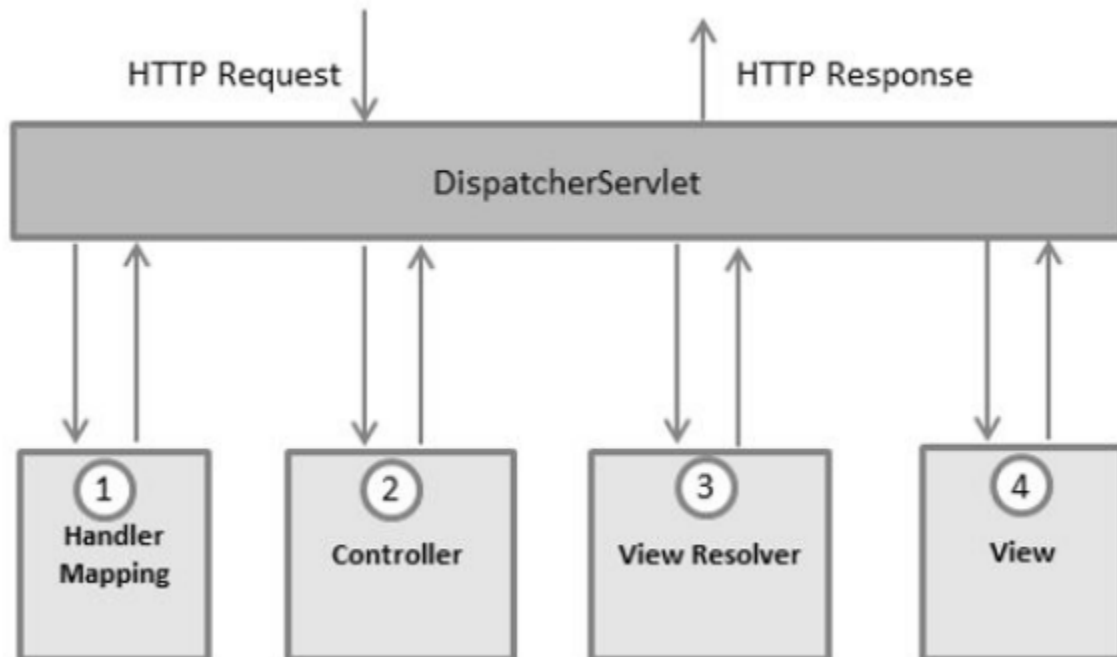
2 ключевых фреймворка:

- Spring Web MVC
- Spring Data REST

Spring Web MVC:

- “базовый” фреймворк в составе Spring для разработки веб приложений.
- Универсальный, на клиентской стороне интегрируется с популярными JS фреймворками.
- Удобен для разработки веб сервисов

Архитектура:



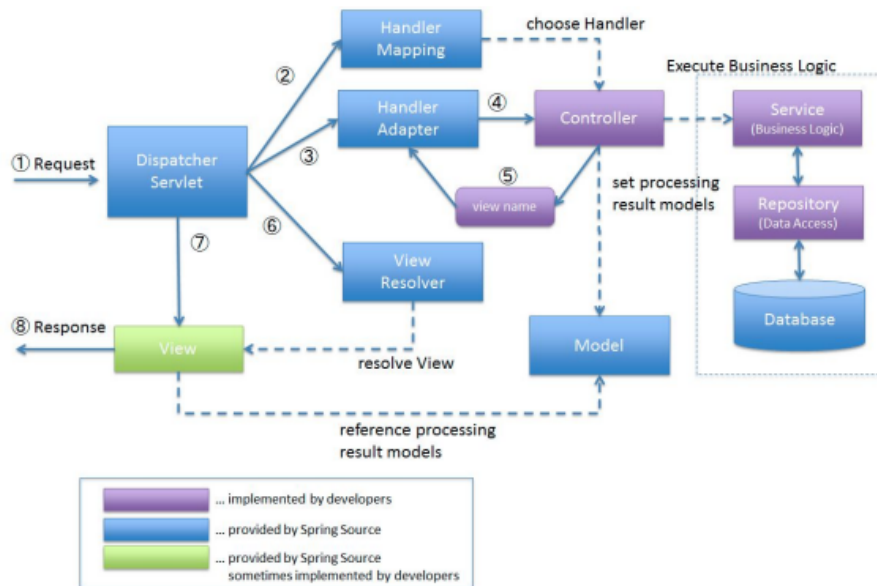
Состав приложения:

- Model -- инкапсулирует данные приложения (состоят из POJO или бинов).
- View -- отвечает за отображение данных модели.
- Controller -- обрабатывает запрос пользователя, создаёт соответствующую модель и передаёт её для отображения в представление.

Dispatcher Servlet:

- Обрабатывает все запросы и формирует ответы на них.
- Связывает между собой все элементы архитектуры Spring MVC.
- Обычный сервлет -- конфигурируется в web.xml.

Обработка запроса:



1. DispatcherServlet получает запрос.
2. DispatcherServlet отправляет задачу выбора подходящего контроллера в HandlerMapping. HandlerMapping выбирает контроллер, который сопоставляется с URL-адресом входящего запроса, и возвращает (выбранный обработчик) и контроллер в DispatcherServlet.
3. DispatcherServlet отправляет задачу выполнения бизнес-логики Controller на HandlerAdapter.
4. HandlerAdapter вызывает процесс бизнес-логики контроллера.
5. Controller выполняет бизнес-логику, устанавливает результат обработки в Модель и возвращает логическое имя представления в HandlerAdapter.
6. DispatcherServlet отправляет задачу разрешения представления, соответствующего имени представления, в ViewResolver. ViewResolver возвращает представление, сопоставленное с именем представления.
7. DispatcherServlet отправляет процесс рендеринга в возвращенное представление.
8. Представление отображает данные модели и возвращает ответ.

Контроллер:

```

@Controller
@RequestMapping("/hello")
public class HelloController {
    @RequestMapping(method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}

```

Возвращаемое
представление

Атрибуты модели

65

```

1  @Controller
2  @RequestMapping("/post")
3  public class ExamplePostController {
4
5      @Autowired
6      ExampleService exampleService;
7
8      @PostMapping("/response")
9      @ResponseBody
10     public ResponseTransfer postResponseController(
11         @RequestBody LoginForm loginForm) {
12         return new ResponseTransfer("Thanks For Posting!!!");
13     }
14 }

```

LoginForm будет
десериализован из
JSON

ResponseTransfer
будет сериализован в
JSON

```

@GetMapping("/books")
public void book() {
    //
}

/* these two mappings are identical */

@RequestMapping(value = "/books", method = RequestMethod.GET)
public void book2() {
}

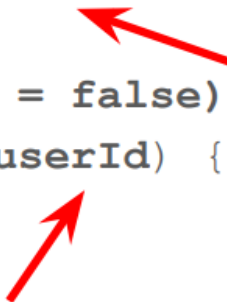
```

Есть аналогичные аннотации для Post, Put, Delete и Patch

```
@PostMapping("/users")
/* First Param is optional */
public User createUser(
    @RequestParam(required = false)
    Integer age,
    @RequestParam String name) {
    // does not matter
}

@PostMapping("/users")
/* Spring преобразует userDto
автоматически, если в классе есть
getters and setters */
public User createUser(
    UserDto userDto) {
    //
}

@GetMapping("/users/{userId}")
public User getUser(
    @PathVariable(required = false)
    String userId) {
    //...
    return user;
}
```




```

@GetMapping(
    "/users/{userId}/{userName}")
public User getUser(UserDto
                    userDto) {

    /* Will set "userId" &
       "userName" properties
       Automatically */
    return user;
}

```



Отображение методов на URL

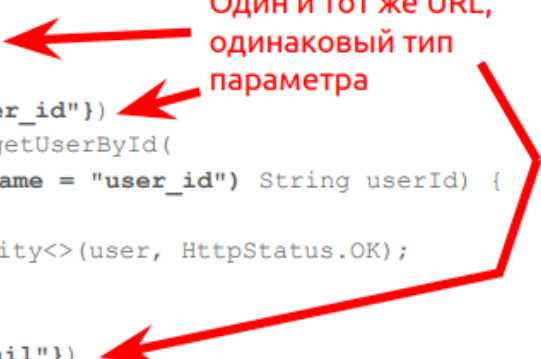
```

@RestController
@RequestMapping("/api/users")
public class UserController {
    @GetMapping(params = {"user_id"})
    public ResponseEntity<?> getUserById(
        @RequestParam(name = "user_id") String userId) {
        // Doesn't matter
        return new ResponseEntity<>(user, HttpStatus.OK);
    }

    @GetMapping(params = {"email"})
    public ResponseEntity<?> getUserByEmail(
        @RequestParam(name = "email") String email) {
        // Doesn't matter
        return new ResponseEntity<>(dtos, HttpStatus.OK);
    }
}

```

Один и тот же URL, одинаковый тип параметра



<pre> 1 @Controller 2 @RequestMapping("books") 3 public class SimpleBookController { 4 5 @GetMapping("/{id}", produces = "application/json") 6 public @ResponseBody Book getBook(@PathVariable int id) { 7 return findBookById(id); 8 } 9 10 private Book findBookById(int id) { 11 // ... 12 } 13 } </pre>	<p>@Controller</p> <p>+</p> <p>@ResponseBody</p> <p>=</p> <p>@RestController</p>	<pre> 1 @RestController 2 @RequestMapping("books-rest") 3 public class SimpleBookRestController { 4 5 @GetMapping("/{id}", produces = "application/json") 6 public Book getBook(@PathVariable int id) { 7 return findBookById(id); 8 } 9 10 private Book findBookById(int id) { 11 // ... 12 } 13 } </pre>
--	---	---



HttpMessageConverter:

- Spring сам не умеет в сериализацию / маршалинг.
- Сериализация / маршалинг реализуются сторонними библиотеками.
- HttpMessageConverter -- адаптер для сторонних библиотек.
- Содержит 4 метода -- canRead(MediaType), canWrite(MediaType), read(Object, InputStream, MediaType) и write(Object, OutputStream, MediaType).
- Есть готовые конвертеры "из коробки"

Spring Data REST:

Автоматически создаёт контроллеры, "открывая наружу" методы определённых репозиторий (учитывая стратегии экспорта).

2. keytool, основные команды

Keytool - это утилита командной строки, для управления ключами или сертификатами, а также хранилищами ключей.

Создать ключи вместе с keystore:

```
keytool -genkey -alias example.com -keyalg RSA -keystore keystore.jks -keysize 2048
```

Создать запрос сертификата (CSR) для существующего Java keystore:

```
keytool -certreq -alias example.com -keystore keystore.jks -file example.com.csr
```

Загрузить корневой или промежуточный CA сертификат:

```
keytool -import -trustcacerts -alias root -file Thawte.crt -keystore keystore.jks
```

Импортировать доверенный сертификат:

```
keytool -import -trustcacerts -alias example.com -file example.com.crt -keystore keystore.jks
```

Экспортировать сертификат из keystore: keytool -export -alias example.com -file example.com.crt -keystore keystore.jks

Сгенерировать самоподписанный сертификат и keystore:

keytool -genkey -alias selfsigned -keyalg RSA -keystore keystore.jks -storepass password
-validity 360 -keysize 2048

Посмотреть сертификат: keytool -printcert -v -file example.com.crt

Посмотреть список сертификатов: keytool -list -v -keystore keystore.jks

Проверить конкретный сертификат по алиасу: keytool -list -v -keystore keystore.jks -alias example.com

Удалить сертификат: keytool -delete -alias example.com -keystore keystore.jks

Изменить пароль keystore: keytool -storepasswd -new new_storepass -keystore keystore.jks

3. На основе сервлетов сервис для создания резиновых уточек по 3д модели с управлением заказами, возможностью указать размеры утки и партии

```
@WebServlet(name = "DuckServlet", urlPatterns = "/duck")
public class DuckServlet extends HttpServlet {

    private DuckService duckService = new DuckService();

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) {
        List<Duck> ducks = duckService.getDucks();

    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) {
        long width = request.getParameter("width");
        long height =
        .getParameter("height");
        long length = request.getParameter("length");
        duckService.addDuck(width, height, length);
    }

    @Override
    protected void doPut(HttpServletRequest request, HttpServletResponse
response) {
        long id = request.getParameter("id");
        duckService.updateDuck(id);
    }
}
```



```

@Override
protected void doDelete(HttpServletRequest request,
HttpServletResponse response) {
    long id = request.getParameter("id");
    duckService.deleteDuck(id);
}
}

```

Вариант 5

1. основные принципы soa

- Standartized Contract – интерфейсы взаимодействия должны быть четко специфицированы.
- Reference Autonomy – взаимосвязи между сервисами должны быть сведены к минимуму.
- Location Transparency – то, где физически располагается сервис, не должно иметь значения при взаимодействии с ним.
- Longevity – сервисы должны разрабатываться с учётом возможности их длительного использования.
- Abstraction – внутренняя логика сервиса должна быть скрыта от клиента.
- Autonomy – сервисы должны самостоятельно контролировать собственную функциональность.
- Statelessness – сервис не должен сохранять состояние между обращениями к нему.
- Granularity – сервис должен реализовывать чётко специфицированный и логически обоснованный набор функций.
- Normalization – сервисы должны быть декомпозированы и нормализованы, чтобы минимизировать избыточность.
- Composability – функциональность сервиса может строиться на базе функциональности других сервисов.
- Discovery – сервисы должны сопровождаться метаданными, позволяющими эффективно идентифицировать и использовать их.
- Reusability – логика приложения разбивается на локальные сервисы, что позволяет повторно использовать код.
- Encapsulation – в сервисы можно “оборачивать” функциональность приложений, построенных по принципам, отличным от SOA

2. spring data rest - конфигурация, основные аннотации

1. Добавляем зависимость в Maven / Gradle.

- Зависимость в Maven:

```
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-rest-webmvc</artifactId>
  <version>3.3.4.RELEASE</version>
</dependency>
```

2. Конфигурируем.

- Не требуется, если используем Spring Boot.
- Задается в классе RepositoryRestMvcConfiguration, который необходимо импортировать в конфигурацию приложения.
- Изменяется путем регистрации своего конфигуратора RepositoryRestConfigurer или наследования от класса адаптера RepositoryRestConfigurerAdapter.

3. Выбираем стратегию экспорта репозитория.

Добавляем в application.properties:

```
spring.data.rest.detection-strategy=visibility/all/default/annotation
```

4. Выбираем базовый URI.

- Задаётся в application.properties: spring.data.rest.basePath=/api
- Может быть задано в RepositoryRestConfigurer:

@Component

```
public class CustomizedRestMvcConfiguration extends RepositoryRestConfigurerAdapter {
    @Override
    public void configureRepositoryRestConfiguration( RepositoryRestConfiguration config) {
        config.setBasePath("/api");
    }
}
```

5. Запускаем приложение.

Основные аннотации:

@RepositoryRestResource - ставится перед репозиторием (необязательный и используется для настройки конечной точки REST: для кастомизации relationships и экспорта)

@RestResource - same, но можно использовать для методов

Пример:

```
@RepositoryRestResource(collectionResourceRel = "people", path = "people")
public interface PersonRepository extends MongoRepository<Person, String> {

    // Prevents GET /people/:id
    @Override
    @RestResource(exported = false)
    public Person findOne(String id);
}
```

3. спецификация (url'ы) веб-сервиса, реализующего домофон

```
paths:
  /code:
    get:
```

```

    description: Get current code
    responses:
      '200':
        description: Current code
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/code'
      '500':
        description: Server error, try again later.
  put:
    description: update current code (add new digit or execute
symbol)
    parameters:
      - name: newDigit
        in: query
        description: new digit
        schema:
          type: integer
          format: int64
      - name: executeCode
        in: query
        description: symbol to start code execution
        schema:
          type: char
    responses:
      '201':
        description: Successful command execution
      '404':
        description: Unsuccessful command execution
(command does not exist)
      '500':
        description: Server error, try again later.
  delete:
    description: delete last digit in code
    responses:
      '200':
        description: Successful deletion
      '500':
        description: Server error, try again later.

```

Вариант 6

1. что-то про soa

SOA— это стиль архитектуры программного обеспечения, который предполагает модульное приложение, состоящее из дискретных и слабосвязанных программных агентов, которые выполняют конкретные функции.

Концепция SOA заключается в следующем: приложение может быть спроектировано и построено таким образом, что его модули легко интегрируются и могут быть легко использованы повторно.

Сервис-ориентированная архитектура (SOA) – это метод разработки программного обеспечения, который использует программные компоненты, называемые сервисами, для создания бизнес-приложений. Каждый сервис предоставляет бизнес-возможности, и сервисы также могут взаимодействовать друг с другом на разных платформах и языках. Разработчики применяют SOA для многократного использования сервисов в различных системах или объединения нескольких независимых сервисов для выполнения сложных задач.

Плюсы SOA

- Повторное использование сервисов
- Эффективное обслуживание

Легче создавать, обновлять и отлаживать небольшие сервисы, чем большие блоки кода в монолитных приложениях

- Более высокая надежность

Службы легче отлаживать и тестировать, чем огромные куски кода, как в монолитах. Это, в свою очередь, делает продукты на основе SOA более надежными.

Минусы:

- Сложность в управлении

Каждый сервис должен обеспечивать своевременную доставку сообщений. Количество этих сообщений может превышать миллион за один раз, что затрудняет управление всеми службами.

- сложности с реализацией асинхронной связи между приложениями;
- большое время отклика, трудности организации обмена большими объемами данных, обусловленные тем, что XML дает надежность, но не скорость (существуют альтернативы XML — в частности, JSON);

2. Особенности, схожесть и отличие ssl и tls

SSL:

- Криптографический протокол.
- Использует асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений.
- Со временем должен быть исключен в пользу TLS.

TLS:

- Основан на SSL 3.0.
- Актуальная версия -- 1.3 (2018 г.).
- Обратно совместим с SSL v3.

Безопасность транспортного уровня (TLS) является преемником протокола SSL. TLS — это улучшенная версия SSL. Он работает почти так же, как SSL, используя шифрование для защиты передачи данных и информации. Эти два термина часто взаимозаменяемы в отрасли, хотя SSL по-прежнему широко используется.

Самое фундаментальное различие между этими протоколами заключается в том, как они устанавливают соединения.

TLS-сертификат	SSL-сертификат
Сертификаты TLS также известны как «безопасность уровня передачи».	SSL-сертификаты также известны как Secure Sockets Layer.
По сравнению с SSL, TLS — более простой протокол.	SSL — более сложный для реализации протокол, чем TLS.
TLS имеет четыре версии, из которых версия TLS 1.3 является последней.	Принимая во внимание, что SSL имеет три версии, из которых SSL 3.0 является последней.
Протокол TLS обеспечивает более высокий уровень безопасности, чем SSL.	Все версии протокола SSL сравнительно подвержены уязвимостям.
Протокол TLS был выпущен в 1999 году.	Между тем, SSL v2.0 был выпущен в 1995 году и v3.0 в 1996 году.
TLS поддерживает Fortezza (алгоритм)	SSL не поддерживает алгоритм Fortezza.
Сертификаты TLS имеют сложный процесс проверки	Сертификаты SSL предлагают простой процесс проверки.

TLS лучше, чем SSL.

В значительной степени из-за известных уязвимостей безопасности протокол SSL устарел.

Самые последние версии TLS также обеспечивают повышение производительности и другие улучшения.

TLS не только более безопасен и производителен, но и поддерживается большинством современных веб-браузеров. Например, Google Chrome давно прекратил поддержку SSL 3.0, а большинство основных браузеров планируют прекратить поддержку TLS 1.0 и TLS 1.1 к 2020 году.

3. spring data rest сервис к репозиторию

```
@Entity
@Data
public class WebsiteUser {
```

```

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    private String name;
    private String email;
}

```

Репозиторий для доступа к бд:

```

@RepositoryRestResource(collectionResourceRel = "users", path = "users")
public interface UserRepository extends
PagingAndSortingRepository<WebsiteUser, Long> {
    List<WebsiteUser> findByName(@Param("name") String name);
}

```

Класс запуска приложения:

```

@SpringBootApplication
public class SpringDataRestApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringDataRestApplication.class,
args);
    }
}

```

4. (Alternative) Дан Spring Data Rest репозиторий с 1 методом public, который наследует crudrepository. Сделать так, чтобы все методы паблик стали доступны

Добавляем в application.properties:

```
spring.data.rest.detected-strategy=annotation
```

Далее навешиваем на все public методы аннотацию @RestResource

```

@Repository
public interface CarRepository extends Repository<Car, Long> {
    @RestResource(exported = false)
    public Car save(Car car);
}

```

Или же можно просто в application.properties указать:

```
spring.data.rest.detected-strategy=default
```

Вариант 7

1. Создание контроллера в спринг

См. [Вариант 4 Особенности разработки RESTful на Spring](#)

2. Криптография в веб приложениях

Сертификаты - Используются для проверки принадлежности открытого ключа его реальному владельцу.

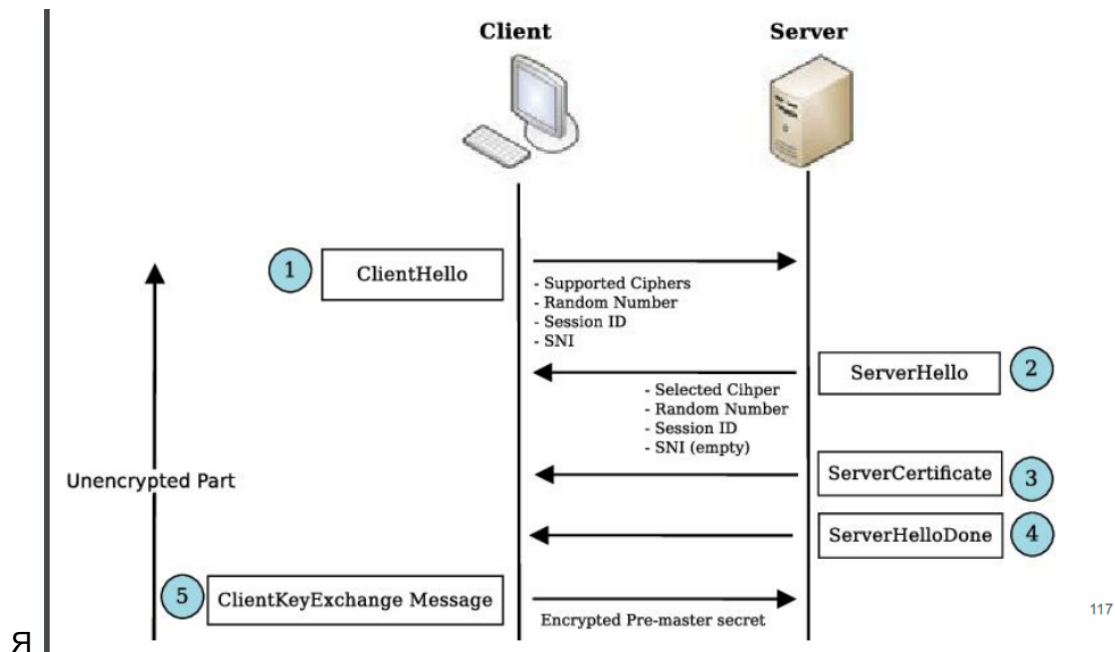
Сначала клиент устанавливает **соединение с сервером** и **настраивает ключи для шифрования** передаваемых данных. Он отправляет информацию о себе, какие поддерживает шифрования и случайное число.

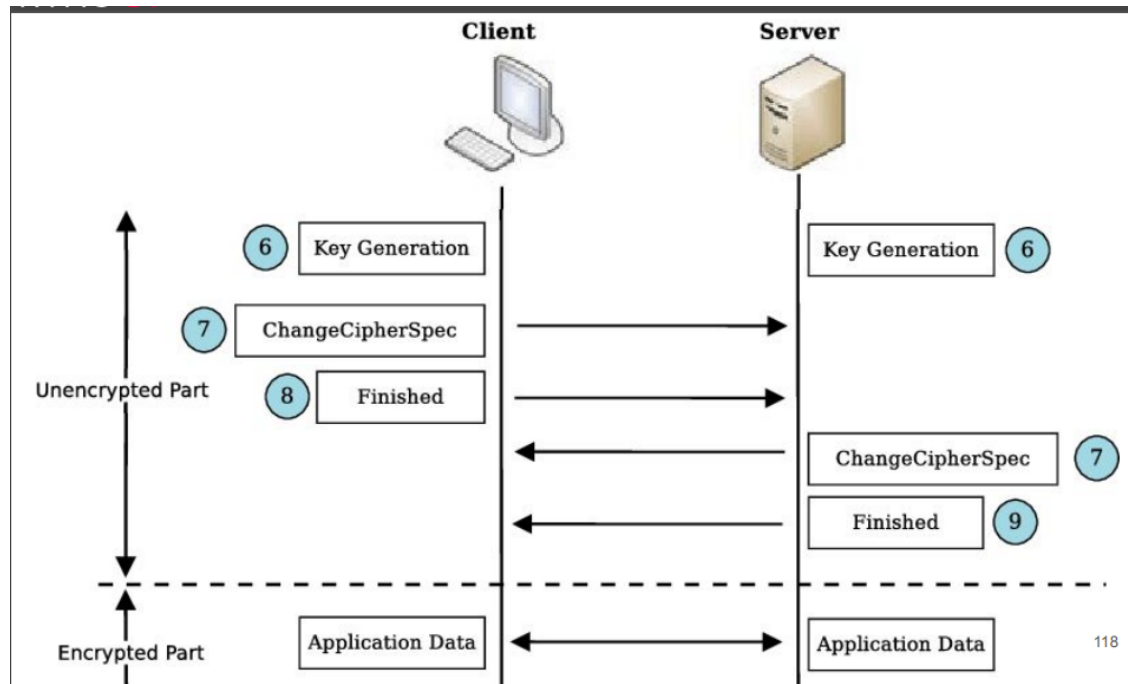
После чего **сервер делает то же самое**, только выбирает какое шифрование использовать.

Вся суть в том что это работает на открытом и закрытом ключе. Закрытый ключ хранится у клиента и сервера. А открытый доступен всем, даже злоумышленнику, если он его украдет. **Через открытый ключ происходит шифрование, а через закрытый дешифровка.**

То есть, как только Клиент и Сервер обмениваются своими ключами - это происходит не зашифровано. А уже данные, когда начинают обмениваться - уже происходит шифрование их.

Так работает SSL/TSL поверх обычно http протокола. -> https





3. Jax-rs казино

```
@Data
public class SlotMachine {
    private Long id;
    private Double winChanse;
    private Double getMany;
    private Double priceOneGame;
    private Double jackpot;
}

@Data
public class ResultGame {
    private Long id;
    private Long slotMachineId;
    private Boolean win;
    private Boolean jackpot;
    private Double bet;
    private Double winSum;
}

@Path("/slots")
public class SlotMachineService {

    private Repository rep = new Repository();

    private SlotMachineGameService game = new SlotMachineGameService();

    @POST
    @Consumes("application/json")
    public SlotMachine createSlotMachine(SlotMachine slotMachine) {
        return
Mapper.mapSlotMachine(rep.createSlotMachine(Mapper.mapSlotMachineEntity(sl
otMachine)));
    }

    @GET
    @Produces("application/json")
    public List<SlotMachine> getSlotMachines() {
```

```

        return
rep.getSlotMachines().stream().map(Mapper::mapSlotMachine).collect(Collection.toList());
    }

    @GET
    @Path("{id}")
    @Produces("application/json")
    public SlotMachine getSlotMachine(@PathParam("id") Long id) {
        return Mapper.mapSlotMachine(rep.getSlotMachine(id));
    }

    @PUT
    @Path("{id}")
    @Consumes("application/json")
    public ResultGame betSlotMachine(@PathParam("id") Long id,
@QueryParam("bet") Double bet) {
        return game.initGame(id, bet);
    }

    @GET
    @Path("{id}/games/{gameId}")
    @Produces("application/json")
    public ResultGame getSlotMachine(@PathParam("id") Long id,
@PathParam("gameId") Long gameId) {
        return game.getGameInSlotMachine(id, gameId);
    }

    @DELETE
    @Path("{id}")
    @Consumes("application/json")
    public void deleteSlotMachine(@PathParam("id") Long id) {
        rep.deleteSlotMachine(id);
    }
}

```

Вариант 8

1. Jax-rs ключевые моменты

Jax-rs - Спецификация API для разработки веб сервисов

Особенности:

- Позволяет создавать REST API к компонентам с помощью аннотаций
- Часть Java EE – работает на любом сервере приложений.
- Внутри сервисов доступны API всех компонентов Java EE.
- Аннотации можно применять внутри любых компонентов

Основные аннотации:

- `@Path` – путь (URL) к ресурсу или методу
- `@GET`, `@PUT`, `@POST`, `@DELETE` и `@HEAD` – метод HTTP-запроса, который будет обработан ресурсом.
- `@Produces` – тип возвращаемого контента (text/html etc)
- `@Consumes` – тип обрабатываемого контента (text/json etc).

Вспомогательные аннотации:

- `@PathParam` – отображает элемент иерархии URL на параметр метода.
- `@QueryParam` – отображает параметр из URL на параметр метода.
- `@MatrixParam` – отображает матричный параметр HTTP-запроса на параметр метода.
- `@HeaderParam` – отображает заголовок HTTP-запроса на параметр метода.
- `@CookieParam` – отображает cookie на параметр метода.
- `@FormParam` – отображает параметр POST-запроса на параметр метода.
- `@DefaultValue` – определяет значение по умолчанию для параметра метода.
- `@Context` – позволяет получить контекстно связанный объект (например, `@Context HttpServletRequest request`)

Порядок разработки сервиса на JAX-RS:

- Создаём проект (в случае Maven можно использовать архетип `maven-archetype-webapp`).
- Добавляем зависимости JAX-RS (если версия JDK < 7).
- Создаём описание представления ресурса (например, с помощью JAXB).
- Создаём ресурс REST (сам веб-сервис).
- Регистрируем ресурс.

Описание представления ресурса.

- Опционально - в принципе, может передавать что угодно
- В каноническом варианте реализуется с помощью аннотаций JAXB – `@XmlRootElement`, `@XmlAttribute`, `@XmlElement` и т.д

2. Spring data rest особенности отличия от spring mvc rest

Основные особенности Spring Data Rest:

- Ресурсы описываются в формате HAL.
 - Hypertext Application Language (HAL)
 - "Work-in-progress" стандарт для описания hypermedia-resources.
 - Гипермедиа -- расширение гипертекста (+ графика, видео, звук и т.д.).
 - Две нотации -- JSON и XML.
- 3 основных вида ресурсов -- коллекция (collection), элемент (item - отдельные элементы коллекции) и ассоциация (association - взаимодействие с ресурсами вложенными в свойства основного).
- Поддерживается постраничный вывод.
- Для коллекций поддерживается динамическая фильтрация.
- Специальный вид ресурсов -- поисковый (search resources) для вызова методов, формирующих поисковые запросы
- Поддерживаются JPA, MongoDB, Neo4j, GemFire и Cassandra.

Отличия от Spring MVC REST:

- В отличие от Spring MVC Rest, где контроллеры реализуются самостоятельно, в Spring Data Rest достаточно описать ресурсы с помощью аннотаций в репозиториях (учитывая стратегии экспорта).

3. Управление банкоматом на сервлетах

```
@WebServlet(name = "AtmServlet", urlPatterns = "/atm")
public class AtmServlet extends HttpServlet {

    private AtmService atmService = new AtmService();

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) {
        long user_id = request.getParameter("user_id");
        long balance = atmService.getBalance(user_id); // узнать баланс
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) {
        long user_id = request.getParameter("user_id");
        long sum = request.getParameter("sum");
        atmService.addBanknotes(user_id, sum); // положить наличные
    }
}
```

```

    }

    @Override
    protected void doDelete(HttpServletRequest request,
        HttpServletResponse response) {
        long user_id = request.getParameter("user_id");
        long sum = request.getParameter("sum");
        atmService.getCash(user_id); // снять наличные
    }
}

```

Вариант 9

1. Понятие ресурса в RESTful. Виды ресурсов

Ресурс — это ключевая абстракция, на которой концентрируется протокол HTTP: это объект с типом, связанными данными, отношениями с другими ресурсами и набором методов, которые с ним работают.

Он похож на экземпляр объекта в ООП языке, с той разницей, что для ресурса определено только несколько стандартных методов (соответствующих стандартным методам HTTP: GET, POST, PUT и DELETE), а экземпляр объекта обычно имеет много методов.

На ресурс указывает URI.

Два вида ресурсов:

- Осуществляющие манипуляции с данными.
- Выполняющие какие-либо операции.

Метод	Ресурс, манипулирующий данными https://api.example.com/collection	Ресурс, выполняющий операции https://api.example.com/clusters/1234/create-vm
POST		Вызывает операцию, интерфейс к которой предоставляет ресурс
GET	Возвращает объект в теле ответа	Возвращает статус асинхронной операции в теле ответа
PUT	Загружает объект из тела запроса на ресурс	
PATCH	Обновляет какую-либо часть содержимого ресурса в соответствии с данными в теле запроса	
DELETE	Удаляет содержимое ресурса. Последующий запрос GET вернёт HTTP 404.	Отменяет асинхронную операцию

2. KeyStore и TrustStore в Java. Утилита keytool

<i>Keystore</i>	<i>TrustStore</i>
Хранятся приватные ключи и сертификаты (клиентские или серверные)	Хранятся доверенные сертификаты (корневые самоподписанные CA root)
Необходим для настройки SSL на сервере	Необходим для успешного подключения к серверу на клиентской стороне
Клиент будет хранить свой приватный ключ и сертификат в keystore	Сервер будет валидировать клиента при двусторонней аутентификации на основании сертификатов в trustStore
Используется API <code>javax.net.ssl.keyStore</code>	Используется API <code>javax.net.ssl.trustStore</code>

131

- В JDK/JRE есть truststore “по умолчанию” -- `$JAVA_HOME/lib/security/cacerts`.
- Пароль -- `changeit`.
- Сервер приложений обычно идёт в комплекте со своими keystore и truststore.

Keytool - утилита предназначена для работы с хранилищами JKS (Java KeyStore): может генерировать пары открытый ключ / закрытый ключ и сохранять их в хранилище ключей. Позволяет:

- Создавать ключи (`keytool -genkey ...`)
- Создать запрос сертификата (CSR) для существующего Java keystore (`keytool -certreq ...`)
- Загрузить корневой или промежуточный CA сертификат (`keytool -import -trustcacerts ...`)
- Импортировать доверенный сертификат (`keytool -import -trustcacerts ...`)
- Сгенерировать самоподписанный сертификат и keystore (`keytool -genkey ...`)
- Посмотреть список сертификатов (`keytool -list ...`) или конкретный сертификат (`keytool -printcert ...`)
- Удалить сертификат (`keytool -delete ...`)

И многое другое (см `-help`)

3. Написать веб-сервис на JAX-RS, который управляет шлагбаумом

```
@Data
public class Barrier {
    private String address;
    private Boolean status;
}

@Path("/barriers")
public class BarrierService {

    private Repository rep = new Repository();

    @POST
    @Consumes("application/json")
    public Barrier createBarrier(Barrier barrier) {
        return
Mapper.mapBarrier(rep.createBarrier(Mapper.mapBarrierEntity(barrier)));
    }

    @GET
    @Produces("application/json")
    public List<Barrier> getBarriers() {
        return
rep.getBarriers().stream().map(Mapper::mapBarrier).collect(Collection.toList());
    }

    @GET
    @Path("{id}")
    @Produces("application/json")
    public Barrier getBarrier(@PathParam("id") Long id) {
        return Mapper.mapBarrier(rep.getBarrier(id));
    }

    @PUT
    @Path("{id}/open")
    @Consumes("application/json")
    public void open(@PathParam("id") Long id) {
        rep.openBarrier(id);
    }
}
```



```

    }

    @PUT
    @Path("{id}/close")
    @Consumes("application/json")
    public void close(@PathParam("id") Long id) {
        rep.closeBarrier(id);
    }

    @DELETE
    @Path("{id}")
    @Consumes("application/json")
    public void deleteBarrier(@PathParam("id") Long id) {
        rep.deleteBarrier(id);
    }
}

```

Вариант 10

1. Языки спецификации веб-сервисов

- Их много.
- Позволяют декларативно описать, что “умеет” веб-сервис.
- Существуют как для RESTful, так и для SOAP.
- Могут быть использованы для построения реестров веб-сервисов.
- Могут быть использованы для автогенерации кода сервиса и/или клиента.

WSDL (англ. Web Services Description Language) — язык описания веб-сервисов и доступа к ним, основанный на языке XML.

Каждый документ WSDL 1.1 можно разбить на следующие логические части:

1. определение типов данных (types) — определение вида отправляемых и получаемых сервисом XML-сообщений
2. элементы данных (message) — сообщения, используемые web-сервисом
3. абстрактные операции (portType) — список операций, которые могут быть выполнены с сообщениями
4. связывание сервисов (binding) — способ, которым сообщение будет доставлено

Пример:

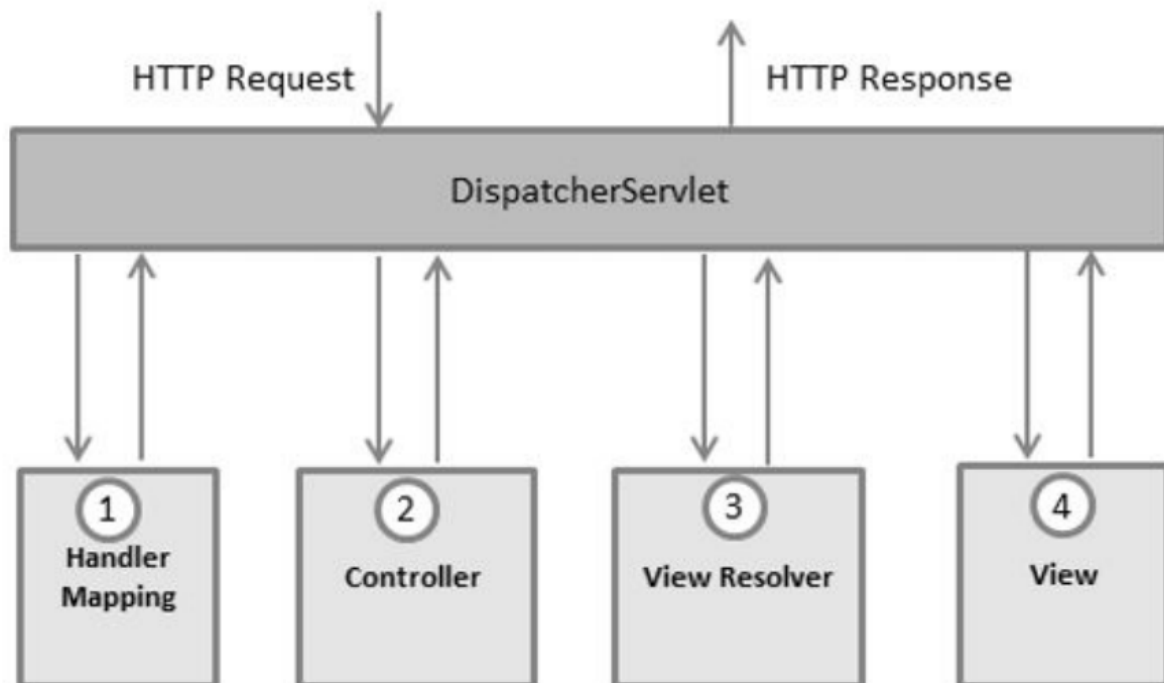
```

▼<message name="ExecuteDeliveryRequestResponse">
  <part name="parameters" element="tns:ExecuteDeliveryRequestResponse"/>
</message>
▼<portType name="TurtlebotPublishersInterface">
  ▶<operation name="ExecuteCoffeeRequest">...</operation>
  ▼<operation name="ExecuteDeliveryRequest">
    <input
      wsam:Action="http://soap.turtlebot.mybot.org/TurtlebotPublishersInterface/Ex
      message="tns:ExecuteDeliveryRequest"/>
    <output
      wsam:Action="http://soap.turtlebot.mybot.org/TurtlebotPublishersInterface/Ex
      message="tns:ExecuteDeliveryRequestResponse"/>
    </operation>
  </portType>
▶<binding name="TurtlebotPublishersWebServicePortBinding"
  type="tns:TurtlebotPublishersInterface">...</binding>
▼<service name="TurtlebotPublishersWebServiceService">
  ▼<port name="TurtlebotPublishersWebServicePort"
    binding="tns:TurtlebotPublishersWebServicePortBinding">
    <soap:address location="http://192.168.100.11:5555/turtlesim_publisher_ws"/>
  </port>
</service>

```

Также см. [2 часть Варианта 1. Всё про SOAP](#)

2. Архитектура spring web mvc



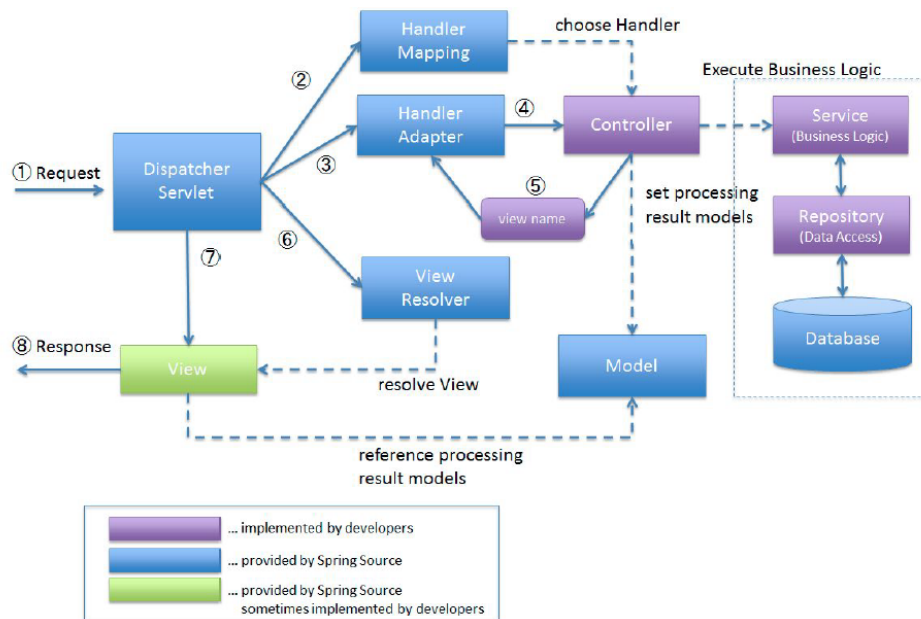
- Model -- инкапсулирует данные приложения (состоят из POJO или бинов).
- View -- отвечает за отображение данных модели.

- Controller -- обрабатывает запрос пользователя, создаёт соответствующую модель и передаёт её для отображения в представление.

По определению HandlerMapping — интерфейс, который реализуется объектами, которые определяют отображение между запросами и объектами обработчиков.

Dispatcher Servlet - Обрабатывает все запросы и формирует ответы на них.

- Связывает между собой все элементы архитектуры Spring MVC.
- Обычный сервлет -- конфигурируется в web.xml.



1. DispatcherServlet получает запрос.
2. DispatcherServlet отправляет задачу выбора подходящего контроллера в HandlerMapping. HandlerMapping выбирает контроллер, который сопоставляется с URL-адресом входящего запроса, и возвращает (выбранный обработчик) и контроллер в DispatcherServlet.
3. DispatcherServlet отправляет задачу выполнения бизнес-логики Controller на HandlerAdapter.
4. HandlerAdapter вызывает процесс бизнес-логики контроллера.
5. Controller выполняет бизнес-логику, устанавливает результат обработки в Модель и возвращает логическое имя представления в HandlerAdapter.
6. DispatcherServlet отправляет задачу разрешения представления, соответствующего имени представления, в ViewResolver. ViewResolver возвращает представление, сопоставленное с именем представления.
7. DispatcherServlet отправляет процесс рендеринга в возвращенное представление.
8. Представление отображает данные модели и возвращает ответ.

3. Последовательность команд для конфигурации двунаправленного взаимодействия двух серверов приложений WildFly путём взаимного вызова Restful веб-сервисов

- **Генерируем серверный сертификат:**

```
keytool -genkeypair -alias localhost -keyalg RSA -keysize 2048 -validity 365 -keystore server.keystore -dname "cn=Server Administrator,o=Acme,c=GB" -keypass secret -storepass secret
```

- **Копируем keystore на сервер приложений:**

```
cp server.keystore $JBOSS_HOME/standalone/configuration
```

- **Генерируем клиентский сертификат:**

```
keytool -genkeypair -alias client -keyalg RSA -keysize 2048 -validity 365 -keystore client.keystore -dname "CN=client" -keypass secret -storepass secret
```

- **Экспортируем содержимое клиентского и серверного keystore в файлы сертификатов:**

```
keytool -exportcert -keystore server.keystore -alias localhost -keypass secret -storepass secret -file server.crt
```

```
keytool -exportcert -keystore client.keystore -alias client -keypass secret -storepass secret -file client.crt
```

- **Импортируем сертификаты в клиентский и серверный truststore:**

```
keytool -importcert -keystore server.truststore -storepass secret -alias client -trustcacerts -file client.crt -noprompt
```

```
keytool -importcert -keystore client.truststore -storepass secret -alias localhost -trustcacerts -file server.crt -noprompt
```

- **Копируем клиентский truststore в конфигурацию сервера приложений:**

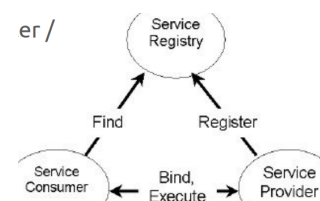
```
cp client.truststore $JBOSS_HOME/standalone/configuration
```

Вариант 11

1. Структура SOA приложения

Любая SOA-система состоит из трёх видов “блоков”:

- **Поставщик (service provider):**
 - создает веб-службу и предоставляет информацию о ней в реестр службы.
 - Решает, какой сервис предоставлять



- Решает вопросы безопасности и доступности
- Брокер (broker) / реестр (registry) / репозиторий (repository).
 - Делает информацию о веб-сервисе доступной для любого потенциального запрашивающего.
 - Есть публичные брокеры (доступны всем), и частные (доступные узкому кругу лиц)
- Потребитель (requester / consumer).
 - находит записи в реестре брокера, используя различные операции поиска, а затем привязывается к поставщику услуг, чтобы вызвать одну из его веб-служб.
 - Должен передать сервис, необходимый потребителям, брокерам, связать с соответствующим сервисом и затем использовать.

2. Виды ресурсов в Spring Data REST

Основные особенности Spring Data Rest:

- Ресурсы описываются в формате HAL.
 - Hypertext Application Language (HAL)
 - “Work-in-progress” стандарт для описания hypermedia-resources.
 - Гипермедиа -- расширение гипертекста (+ графика, видео, звук и т.д.).
 - Две нотации -- JSON и XML.
- 3 основных вида ресурсов -- коллекция (collection), элемент (item - отдельные элементы коллекции) и ассоциация (association - взаимодействие с ресурсами вложенными в свойства основного).
- Поддерживается постраничный вывод.
- Для коллекций поддерживается динамическая фильтрация.
- Специальный вид ресурсов -- поисковый (search resources) для вызова методов, формирующих поисковые запросы
- Поддерживаются JPA, MongoDB, Neo4j, GemFire и Cassandra.

3. Написать спецификацию (url'ы) сервиса по записи студентов на курсы.

```
paths:
  /students/{student-id}/courses:
    get:
      description: Get all available courses for student with id =
{student-id}
      responses:
        '200':
          description: List of courses
          content:
            application/json:
              schema:
```

```

$ref: '#/components/schemas/courses'

'500':
  description: Server error, try again later.

/students/{student-id}/courses/{course-id}:
  post:
    description: enroll a student with id = {student-id} in the
course with id = {course-id}
    responses:
      '201':
        description: Successful enrollment
      '404':
        description: Unsuccessful enrollment (bad id)
      '500':
        description: Server error, try again later.

  delete:
    Description: remove a student with id = {student-id} from the
course with id = {course-id}
    parameters:
      - name: reason
        in: query
        description: reason for unsubscription
        schema:
          type: string
    responses:
      '200':
        description: Successful unsubscription
      '404':
        description: Unsuccessful unsubscription (bad id)
      '500':
        description: Server error, try again later.

```

Вариант 12

1. Описание представления ресурса JAX-RS



Путь (URL) к ресурсу (или к методу) задается с помощью аннотации `@Path`

- `@GET`, `@PUT`, `@POST`, `@DELETE` и `@HEAD` – метод HTTP-запроса, который будет обработан ресурсом.

Описание представления ресурса:

- Опционально - в принципе, может передавать что угодно
- В каноническом варианте реализуется с помощью аннотаций JAXB – `@XmlRootElement`, `@XmlAttribute`, `@XmlElement` и т.д

Например:

 Описание представления ресурса	 Описание представления коллекции
<pre>(...) @XmlRootElement(name = "student") @XmlAccessorType(XmlAccessType.FIELD) public class Student { @XmlAttribute private Integer id; @XmlElement private String name; (...) }</pre>	<pre>(...) @XmlRootElement(name = "student") @XmlAccessorType(XmlAccessType.FIELD) public class Configurations { @XmlAttribute private Integer size; @XmlElement private List<Student> students; (...) }</pre>

2. Криптография в приложениях на Java: особенности, стандарты, протоколы

Есть спецификация Java Cryptography Architecture (JCA, не путать с Java Connector Architecture!).

Протоколы -- TLS / SSL.

- Клиентом и сервером являются веб-сервисы.
- Используется инфраструктура JRE / JDK и сервера приложений.

См. [9.2](#)

3. Restful-сервис на базе сервлета, реализующий механизм управления номеронабирателем телефонного аппарата. Номеронабиратель должен поддерживать функции локальных, междугородних и международных звонков, а также автодозвон

```
public class PhoneServlet extends HttpServlet {  
    static final long serialVersionUID = 1L;  
}
```

```

    private Phones phones; // back-end bean

    // Executed when servlet is first loaded into container.
    @Override
    public void init() {
        this.phones = new Phones();
        novels.setServletContext(this.getServletContext());
    }

    // GET /phones
    // GET /phones?id=1
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) {
        .....
        sendResponse(response, phones.toXml(novel));
    }

    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse
response) {
        String phone = request.getParameter("phone");
        .....
    }
}

```

Вариант 13

1. COA - особенности, плюсы и минусы

См [вопрос 6.1](#)

2. SSL/TLS - особенности, отличия и сходства

См [вопрос 6.2](#)

3. Дан Spring Data Rest репозиторий с 1 методом public, который наследует crudrepository. Сделать так, чтобы все методы паблик стали доступны

См [вопрос 6.4](#)