СБЕРБАНК ТЕХНОЛОГИИ

Сервлеты
Spring Web MVC framework

ПЛАН ЗАНЯТИЯ

СБЕРБАНК ТЕХНОЛОГИИ

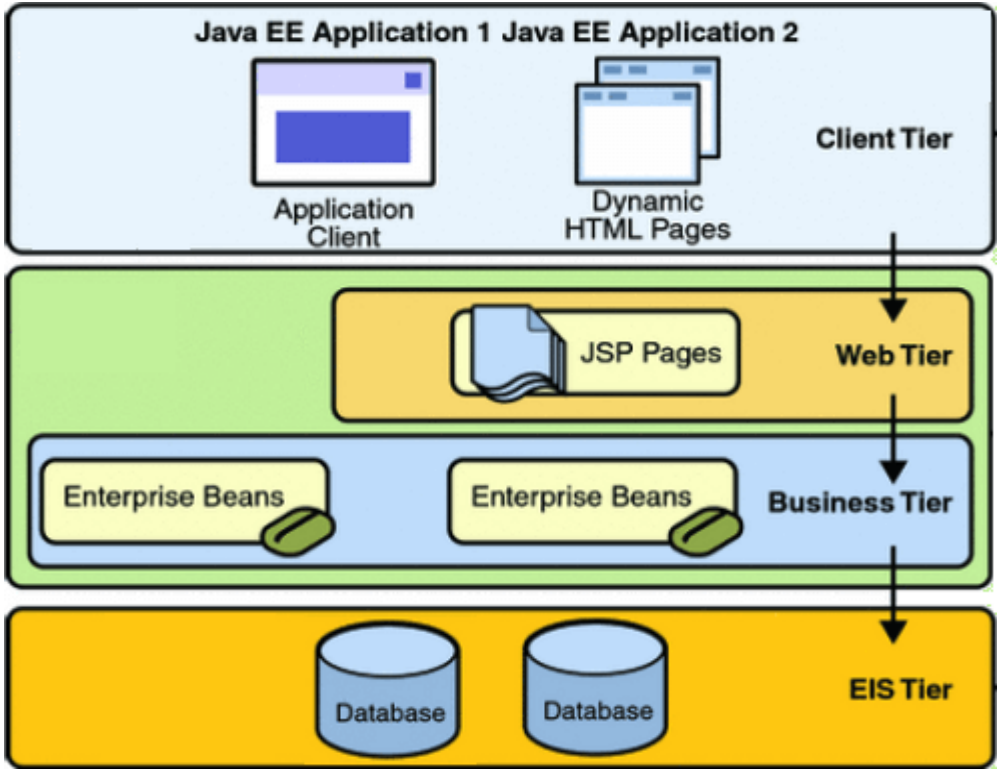
- Узнаем что такое сервер приложений.
- А ещё что такое сервлеты, фильтры и слушатели.
- Напишем свой сервлет. (если не забудем)
- Реализуем небольшое приложение, обрабатывающее HTTP запросы с использованием Spring Web MVC framework.
- Скомпонуем Spring MVC и Hibernate (Spring Data)

СЕРВЕР ПРИЛОЖЕНИЙ

СБЕРБАНК ТЕХНОЛОГИИ

Сервер приложений – компонентная программная платформа, предоставляющая среду для исполнения приложений.

Обычно это средний слой **трёхуровневой архитектуры**.



СЕРВЕР ПРИЛОЖЕНИЙ: ПРЕИМУЩЕСТВА

СБЕРБАНК ТЕХНОЛОГИИ

- **Целостность данных и кода.**

  Бизнес логика на отдельном сервере/серверах => обновление приложений для всех пользователей одновременно.

- **Централизованная настройка и управление**

- **Безопасность**

  Действует как центральная точка, используя которую, поставщики сервисов могут управлять доступом к данным и частям самих приложений.

- **Поддержка транзакций**

СЕРВЛЕТЫ

СБЕРБАНК ТЕХНОЛОГИИ

Сервлеты расширяют возможности серверов с приложениями, работающими по принципу запрос-ответ.

Обычно используются для обработки HTTP запросов.

СЕРВЛЕТЫ

СБЕРБАНК ТЕХНОЛОГИИ

Все сервлеты имплементируют интерфейс javax.servlet.Servlet.

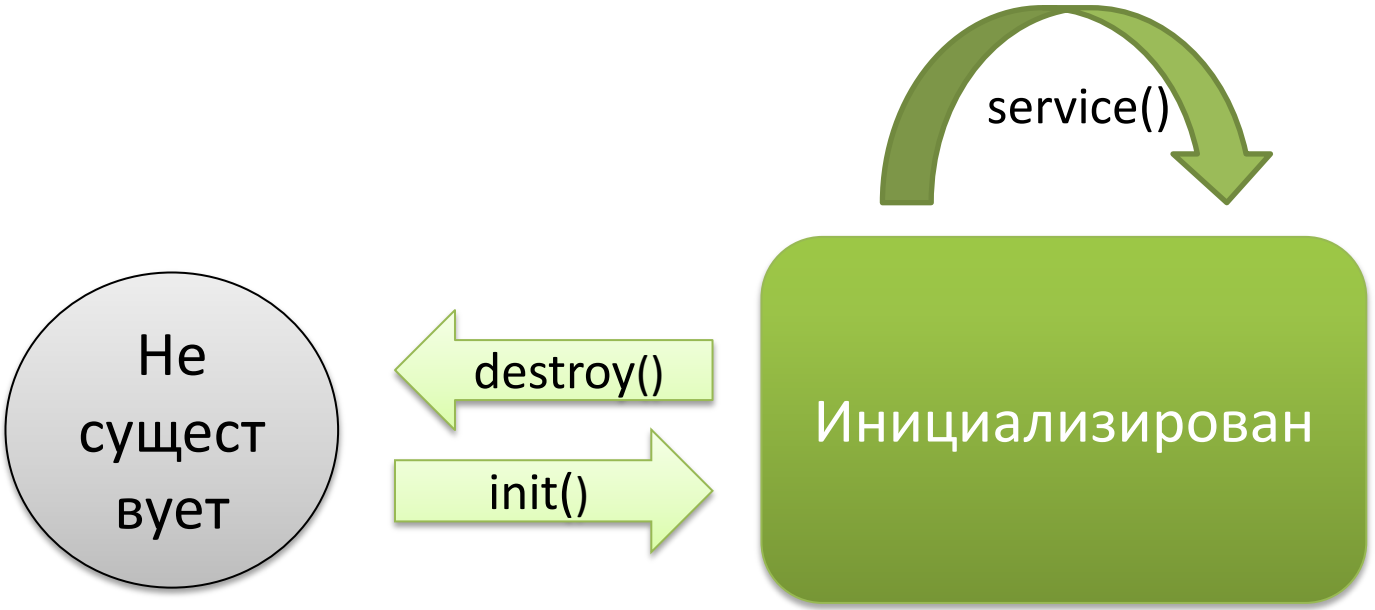
```
public interface Servlet {
    void init(ServletConfig var1) throws
ServletException;

    void service(ServletRequest request, ServletResponse
response) throws ServletException, IOException;

    void destroy();
    …
}
```

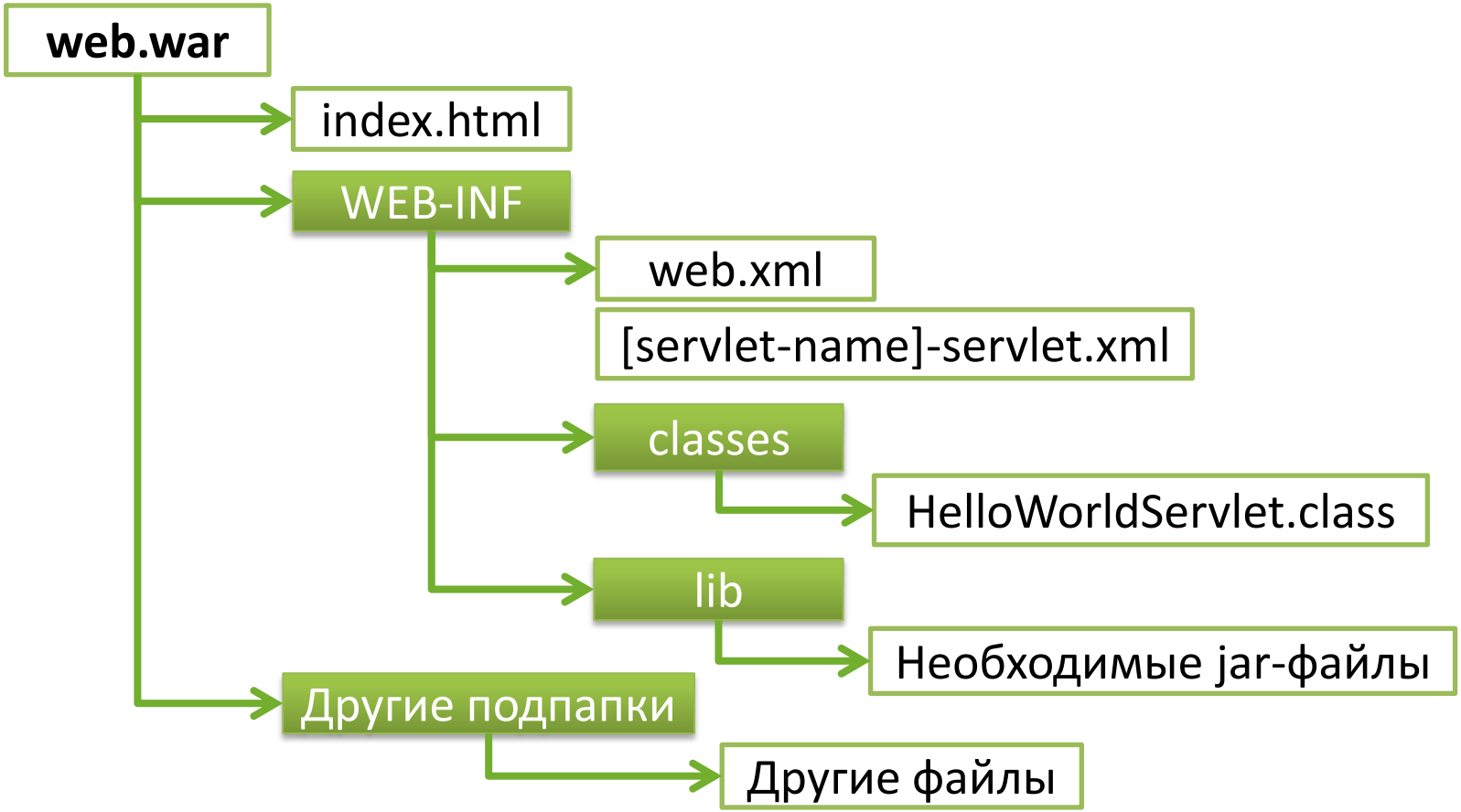
ЖИЗНЕННЫЙ ЦИКЛ СЕРВЛЕТА

СБЕРБАНК ТЕХНОЛОГИИ



СТРУКТУРА JEE WEB ПРИЛОЖЕНИЯ

СБЕРБАНК ТЕХНОЛОГИИ



HTTP СЕРВЛЕТЫ

СБЕРБАНК ТЕХНОЛОГИИ

Используются для обработки HTTP запросов. Наследуют абстрактный класс javax.servlet.http.HttpServlet, где определены:

- **doGet -** HTTP GET – получение ресурса по URI.
- **doPost -** HTTP POST – передача пользовательских данных (например, для добавления записей в БД, обработки данных клиентской формы).
- **doPut -** HTTP PUT – изменение ресурса с указанным URI если он существует, иначе – создание нового, доступного по этому URI.
- **doDelete -** HTTP DELETE – удаление ресурса с указанным URI.

HTTP STATUS CODES

СБЕРБАНК ТЕХНОЛОГИИ

Код состояния HTTP — часть первой строки ответа сервера при запросах по протоколу HTTP.

Представляет собой 3 десятичных цифры, первая из которых определяет класс состояния:

- 1xx – информационные;
- 2xx – успешные;
- 3xx – перенаправление;
- 4xx – ошибка клиента;
- 5xx – ошибка сервера.

HTTP STATUS CODES: ИНФОРМАЦИОННЫЕ

СБЕРБАНК ТЕХНОЛОГИИ

Информируют о процессе передачи запроса.

- **100** Continue

  Сервер удовлетворён начальными сведениями о запросе, клиент может продолжать передавать запрос.

- **101** Switching Protocols

  Сервер предлагает перейти на более подходящий для указанного ресурса протокол.

HTTP STATUS CODES: УСПЕШНЫЕ

СБЕРБАНК ТЕХНОЛОГИИ

Информируют об успешном получении и обработке запроса.

- **200** OK

  Выполнено успешно.
- **201** Created

  Был создан новый ресурс.
- **202** Accepted

  Запрос принят, но обработка не завершена (она может занять много времени, клиенту не обязательно дожидаться окончания).
- **204** No Content

  Сервер успешно обработал запрос (в ответе могут быть только заголовки, тело сообщения отсутствует).

HTTP STATUS CODES: ПЕРЕНАПРАВЛЕНИЕ

СБЕРБАНК ТЕХНОЛОГИИ

Сообщают клиенту, что для успешного выполнения операции необходимо выполнить другой запрос, как правило, по другому URI (его адрес сервер указывает в заголовке Location).

- **301** Moved Permanently - Запрошенный документ был окончательно перенесен на новый URI.
- **304** Not Modified - Клиент запросил ресурс методом GET с заголовком If-Modified-Since или If-None-Match и ресурс не изменился с указанного момента (при этом ответ сервера не должен содержать тела).

HTTP STATUS CODES: ОШИБКА КЛИЕНТА

СБЕРБАНК ТЕХНОЛОГИИ

Сервер сообщает об ошибке со стороны клиента.

- **400** Bad Request - Синтаксическая ошибка в запросе клиента.
- **401** Unauthorized - Требуется аутентификация клиента.
- **403** Forbidden - Сервер понял запрос, но отказывается его выполнять из-за ограничений в доступе для клиента.
- **404** Not Found - Сервер понял запрос, но не нашёл ресурса по указанному URI.

HTTP STATUS CODES: ОШИБКА СЕРВЕРА

СБЕРБАНК ТЕХНОЛОГИИ

Неудачное выполнение операции по вине сервера.

- **500** Internal Server Error - Внутренняя ошибка сервера.
- **501** Not Implemented - Сервер не поддерживает функциональность, необходимую для обработки запроса.
- **503** Service Unavailable - Сервер временно не имеет возможности обрабатывать запросы по техническим причинам (например, обслуживание или перегрузка).

СЕРВЛЕТНЫЕ ФИЛЬТРЫ

СБЕРБАНК ТЕХНОЛОГИИ

Позволяют перехватывать запросы и ответы с целью:

- их изменения;
- использования информации, содержащейся в них.

Примеры:

- блокирование запросов определённых пользователей;
- логирование и аудит действий пользователей;
- сжатие данных;
- шифрование данных;
- языковая локализация.
- управление транзакциями

СЛУШАТЕЛИ СОБЫТИЙ СЕРВЛЕТА

СБЕРБАНК ТЕХНОЛОГИИ

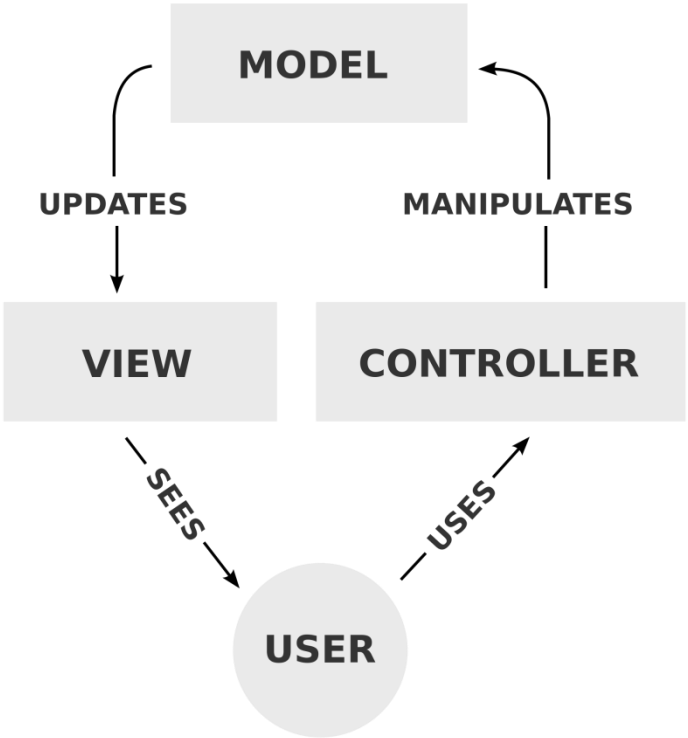
Позволяют отслеживать ключевые события web-приложения.

- События контекста сервлета (уровень приложения):
  - контекст создан / удаляется;
  - добавлен / удалён / заменён атрибут контекста.
- События HTTP сессии:
  - сессия создана / разрушена;
  - добавлен / удалён / заменён атрибут сессии.

SPRING WEB MVC FRAMEWORK

СБЕРБАНК ТЕХНОЛОГИИ

Spring **Web MVC** framework предоставляет model-view-controller решение, которое используется для написания гибких и слабосвязанных web приложений.



SPRING WEB MVC: ПРЕИМУЩЕСТВА

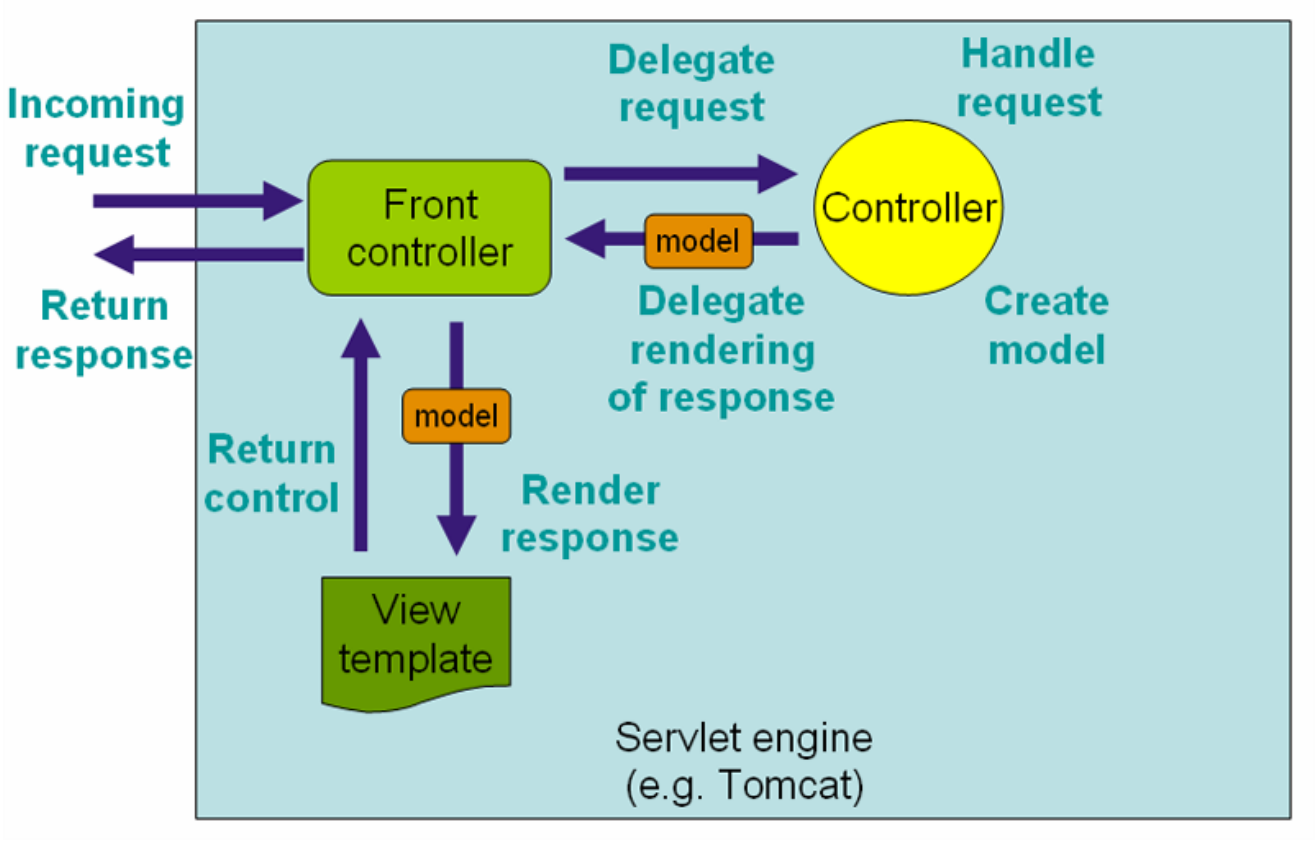
СБЕРБАНК ТЕХНОЛОГИИ

- Простая реализация **RESTful** приложений.
- Конфигурирование аннотациями и другие **возможности Spring** framework.
- Гибкость в поддержке различных **типов представлений** (например, JSP, velocity, XML, PDF и др.).

DispatcherServlet

СБЕРБАНК ТЕХНОЛОГИИ

Spring Web MVC framework построен вокруг **DispatcherServlet**.



[SERVLET-NAME]-SERVLET.XML

СБЕРБАНК ТЕХНОЛОГИИ

Во время инициализации DispatcherServlet Spring Web MVC ищет в папке **WEB-INF** файл с именем **[servlet-name]-servlet.xml** и создаёт бины определённые в нём.

ПАРАМЕТР contextConfigLocation

СБЕРБАНК ТЕХНОЛОГИИ

Параметром инициализации сервлета contextConfigLocation можно изменить путь до конфигурационного файла Spring.

```
<servlet>
  <servlet-name>mvc-users</servlet-name>
  <servlet-class>org.….DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/test.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

КОНТРОЛЛЕРЫ

СБЕРБАНК ТЕХНОЛОГИИ

Контроллеры предоставляют доступ к приложению через сервисный интерфейс.

Они интерпретируют пользовательский ввод и трансформируют его в модель, которая передаётся пользователю посредством представления.

ПРИМЕР КОНТРОЛЛЕРА

СБЕРБАНК ТЕХНОЛОГИИ

```
@Controller
public class HelloWorldController {

  @RequestMapping("/helloworld")
  public String helloWorld(Model model) {
    model.addAttribute("message", "Hello World!");

    return "helloWorld";
  }

}
```

@Controller

СБЕРБАНК ТЕХНОЛОГИИ

Аннотация **@Controller** указывается на классе. Является индикатором того, что класс выполняет роль контроллера.

Диспетчер (DispatcherServlet) сканирует такие классы на предмет маппинга методов и выявляет все аннотации **@RequestMapping**.

АННОТАЦИЯ @RequestMapping

СБЕРБАНК ТЕХНОЛОГИИ

Указывается на классе или методе (конкретизирует указанную на классе аннотацию, если она там есть). Сопоставляет запрос с обработчиком.

```
@Controller
@RequestMapping("/users")
public class UserRestController {
    @RequestMapping(path = "/{id}", method = GET)
    @ResponseBody
    public User getUser(@PathVariable long id) {
        return user;
    }
}
```

АТРИБУТЫ @RequestMapping

СБЕРБАНК ТЕХНОЛОГИИ

- **path** (value)

  Пути для сопоставления URI.
- **method**

  HTTP методы (GET, POST, PUT, DELETE, HEAD и др.).
- **params**

  Параметры сопоставленного запроса.
- **headers**

  HTTP заголовки сопоставленного запроса.
- **consumes**

  Типы содержимого запроса (HTTP заголовок Content-Type).
- **produces**

  Типы содержимого ответа (HTTP заголовок Accept).

@RequestMapping: consumes & produces

СБЕРБАНК ТЕХНОЛОГИИ

Служат для определения типа содержимого запроса/ответа.

Передаются в заголовках Content-Type и Accept.

Основные варианты переписаны в классе MediaType. Например:

- `ALL_VALUE = "*/*";`
- `APPLICATION_JSON_VALUE = "application/json";`
- `APPLICATION_OCTET_STREAM_VALUE = "application/octet-stream";`
- `APPLICATION_XML_VALUE = "application/xml";`
- `TEXT_HTML_VALUE = "text/html";`

СОСТАВНЫЕ ВАРИАНТЫ @RequestMapping

СБЕРБАНК ТЕХНОЛОГИИ

- @GetMapping

  Аналог `@RequestMapping(method = GET)`
- @PostMapping

  Аналог `@RequestMapping(method = POST)`
- @PutMapping

  Аналог `@RequestMapping(method = PUT)`
- @DeleteMapping

  Аналог `@RequestMapping(method = DELETE)`
- @PatchMapping

  Аналог `@RequestMapping(method = PATCH)`

АННОТАЦИЯ @ResponseBody

СБЕРБАНК ТЕХНОЛОГИИ

Указывается на методе или типе, означает, что возвращаемое значение пишется в ответ напрямую.

```
@GetMapping("/something")
@ResponseBody
public String helloWorld() {
    return "Hello World!";
}
```

Возвращаемый объект превращается в тело ответа путём использования специальных конвертеров (имплементации интерфейса HttpMessageConverter).

список ниже

ШАБЛОНЫ URI

СБЕРБАНК ТЕХНОЛОГИИ

URI шаблон – это URI подобная строка с именами переменных.

Пример: `/owners/{ownerId}/pets/{petId}`.
Для строки запроса http://www.example.ru/owners/ivanov/pets/12 значение ownerId будет ivanov, а petId – 12.

В шаблоне также можно использовать wildcard’ы.

Пример: `/owners/*/pets/{petId}`.

АННОТАЦИЯ @PathVariable

СБЕРБАНК ТЕХНОЛОГИИ

Указывается на аргументе метода обработчика для связи его со значением в шаблоне URI.

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {
    @RequestMapping("/pets/{petId}")
    public void findPet(@PathVariable String ownerId,
      @PathVariable String petId, Model model) {
        // Implementation omitted
    }
}
```

КОНКРЕТИЗАЦИЯ ПО ПАРАМЕТРАМ

СБЕРБАНК ТЕХНОЛОГИИ

Можно конкретизировать обработчик по параметрам запроса.

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {
    @GetMapping(path = "/pets/{petId}", params =
      "myParam=myValue")
    public void findPet(@PathVariable String ownerId,
      @PathVariable String petId, Model model) {
        // implementation omitted
    }
}
```

КОНКРЕТИЗАЦИЯ ПО ЗАГОЛОВКАМ

СБЕРБАНК ТЕХНОЛОГИИ

Также обработчик может быть конкретизирован по заголовку.

```
@Controller
@RequestMapping("/owners/{ownerId}")
public class RelativePathUriTemplateController {
    @GetMapping(path = "/pets/{petId}", headers =
      "myHeader=myValue")
    public void findPet(@PathVariable String ownerId,
      @PathVariable String petId, Model model) {
        // Implementation omitted
    }
}
```

АННОТАЦИЯ @RequestParam

СБЕРБАНК ТЕХНОЛОГИИ

Указывает связь между параметром запроса и атрибутом метода обработчика.

```
@GetMapping
@ResponseBody
public List<User> findUsersByNameAndAge(
  @RequestParam("name") String name,
  @RequestParam("age") int age) {
    return users;
}
```

АННОТАЦИЯ @RequestHeader

СБЕРБАНК ТЕХНОЛОГИИ

Указывает связь между заголовком запроса и атрибутом метода обработчика.

```
@RequestMapping("/displayHeaderInfo.do")
public void displayHeaderInfo(
  @RequestHeader("Accept-Encoding") String encoding,
  @RequestHeader("Keep-Alive") long keepAlive) {
    //...
}
```

АННОТАЦИЯ @RequestBody

СБЕРБАНК ТЕХНОЛОГИИ

Указывает связь между аргументом метода обработчика и содержимым запроса.

```
@PostMapping(path = "/pets", consumes =
  MediaType.APPLICATION_JSON_VALUE)
public void addPet(@RequestBody Pet pet, Model model) {
    // Implementation omitted
}

@PutMapping("/something")
public void handle(@RequestBody String body,
  Writer writer) throws IOException {
    writer.write(body);
}
```

ВХОДНЫЕ АТРИБУТЫ ОБРАБОТЧИКОВ

СБЕРБАНК ТЕХНОЛОГИИ

Допустимы также другие типы входных атрибутов обработчиков:

- **HttpServletRequest** / **ServletRequest**
- **HttpServletResponse** / **ServletResponse**
- **HttpSession**
- **java.io.InputStream** / **java.io.Reader**

  Для получения содержимого запроса.
- **java.io.OutputStream** / **java.io.Writer**

  Для формирования содержимого ответа.
- **@SessionAttribute**

  Для получения существующего атрибута сессии.
- **java.util.Map** / **org.springframework.ui.Model**

  Явное указание модели для использовании в представлении.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ ОБРАБОТЧИКОВ

СБЕРБАНК ТЕХНОЛОГИИ

Допустимы и другие типы возвращаемых значений обработчиков:

- **Model**, **Map**

  Модель или данные для модели. Представление определяется неявно.
- **View**

  Представление. Модель, например, через атрибут Model.
- **ModelAndView**
- **String**

  Логическое имя представления.
- **void**

  Метод сам сформировал ответ (например, через атрибут ServletResponse) или используется неявное определение представления.
- **ResponseEntity**

АННОТАЦИЯ @RestController

СБЕРБАНК ТЕХНОЛОГИИ

@RestController – аналог одновременного указания аннотаций **@Controller** и **@ResponseBody**.

Для удобства, так как часто контроллеры применяются именно для реализации REST API, т.е. обычно возвращают XML или JSON.

ОБРАБОТКА ИСКЛЮЧЕНИЙ

СБЕРБАНК ТЕХНОЛОГИИ

@ExceptionHandler указывает обработчик исключения.

```
@RestController @RequestMapping("/users")
public class UserRestController {
    @GetMapping("/{id}")
    public User getUser(@PathVariable long id) {
        return userController.getUser(id);
    }
    @ExceptionHandler(UserNotFoundException.class)
    public ResponseEntity<Void>
handleException(UserNotFoundException e) {
        return ResponseEntity.notFound().build();
    }
}
```

ОБРАБОТКА ИСКЛЮЧЕНИЙ

СБЕРБАНК ТЕХНОЛОГИИ

Аннотацией @ResponseStatus помечаются бизнес исключения. В случае возникновения такого исключения, будет возвращён код ошибки и причина, согласно заданным в параметрах аннотации значениям.

```
@ResponseStatus(code = NOT_FOUND, reason = "User not found")
public class UserNotFoundException extends Exception {
    …
}
```

ТЕСТИРОВАНИЕ

СБЕРБАНК ТЕХНОЛОГИИ

Это очень просто! Используйте Spring MVC Test Framework.

```
@RunWith(MockitoJUnitRunner.class)
public class UserRestControllerTest {
 private MockMvc mockMvc;

 @Before
 public void setUp() {
  mockMvc = standaloneSetup(new UserRestController(…)).build();
 }

 @Test
 public void testGetUserOk() throws Exception {
  mockMvc.perform(get("/users/{id}", 1)
    .accept(APPLICATION_XML)).andExpect(status().isOk())
    .andExpect(xpath("/User/name").string("Ivanov"));
 }}
```

ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА

СБЕРБАНК ТЕХНОЛОГИИ

- http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
- http://docs.spring.io/spring/docs/current/spring-framework-reference/html/integration-testing.html#spring-mvc-test-framework