MVC

Фреймворк

Фреймворк (framework) - это каркас, облегчающий разработку и объединение разных компонентов большого программного проекта.

Фреймворк

- готовые компоненты и решения
- основы безопасности
- стандартизация
- сообщество

Фреймворки

- Symfony
- Zend
- Yii
- Laravel
- Spring MVC
- JSF
- Wicket

Фреймворк

- Когда использовать?
- “Простые” и “сложные” фреймворки
- CMF и CMS

MVC

MVC

- Модель (Model)
- Представление (View)
- Контроллер (Controller)

MVC

MVC предоставляет возможность отделения друг от друга слоя данных, представления и логики поведения.

Цель

Основная цель - повышение возможности повторного использования кода.

Какие задачи решаются:

- Использование нескольких представлений у одной модели.
- Изменение реакции без изменения представления.
- Разделение труда специалистов по областям.

Модель

Модель

- данные и методы работы с этими данными
- реагирует на запросы, изменяя своё состояние
- не содержит информации о том, как можно визуализировать данные

Модель

- описание структуры данных
- взаимодействие с хранилищем (СУБД)
- бизнес-логика*

Модель

```
public class Paper extends BaseEntity {

    private String title;

    private PaperStatus status;

    private Date createDate;

    private Date updateDate;

    private String comment;

    private Boolean locked;

    public PaperStatus getStatus() {
        return status;
    }
    public void setStatus(PaperStatus status) {
        this.status = status;
    }
...
```

Модель

```
public class EmailForm {
    private String subject;
    private String to;
    private String message;

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

...

}
```

Представление

Представление

Отвечает за отображение информации (визуализацию)

Представление

- Каркас формата (html, xml, json, jsp)
- Вывод данных
- Минимальный набор операторов (шаблонизатор)

Представление

```
<p:accordionPanel activeIndex="-1" style="height:100%;">
   <c:forEach items="#{reportsBacking.reports}" var = "report" varStatus="iter">
       <ui:include src="#{report}">
           <ui:param name="num" value="#{iter.index+1}" />
       </ui:include>
   </c:forEach>
</p:accordionPanel>
```

Представление

```
<h1>Результат обработки формы</h1>
  <table>
      <tr>
          <td>Тема:</td>
          <td><p th:text="${emailForm.subject}" /></td>
      </tr>
      <tr>
          <td>Кому:</td>
          <td> <p th:text="${emailForm.to}" /></td>
      </tr>
      <tr>
          <td>Сообщение:</td>
          <td><p th:text="${emailForm.message}" /></td>
      </tr>
  </table>
  <a href="/">Отправить другое сообщение</a>
```

Контроллер

Контроллер

- контролирует ввод данных пользователем
- использует модель и представление для реализации необходимой реакции

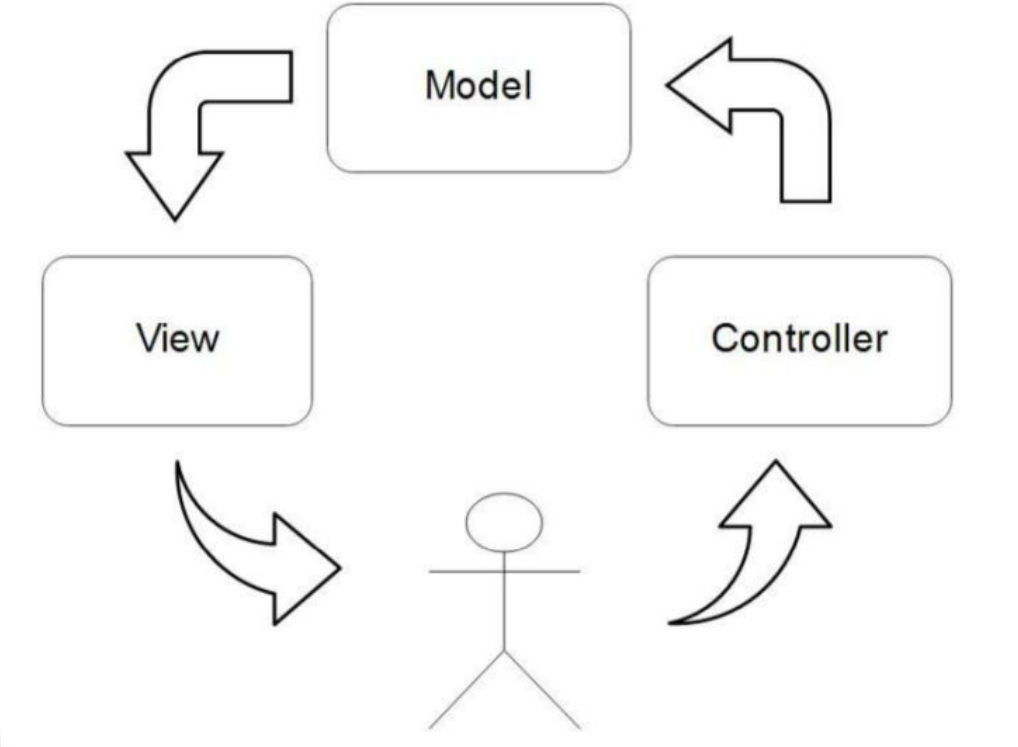
Контроллер

- получение данных от пользователя
- передача данных в модель
- получение представления на основе текущего состояния модели, вывод результатов пользователю

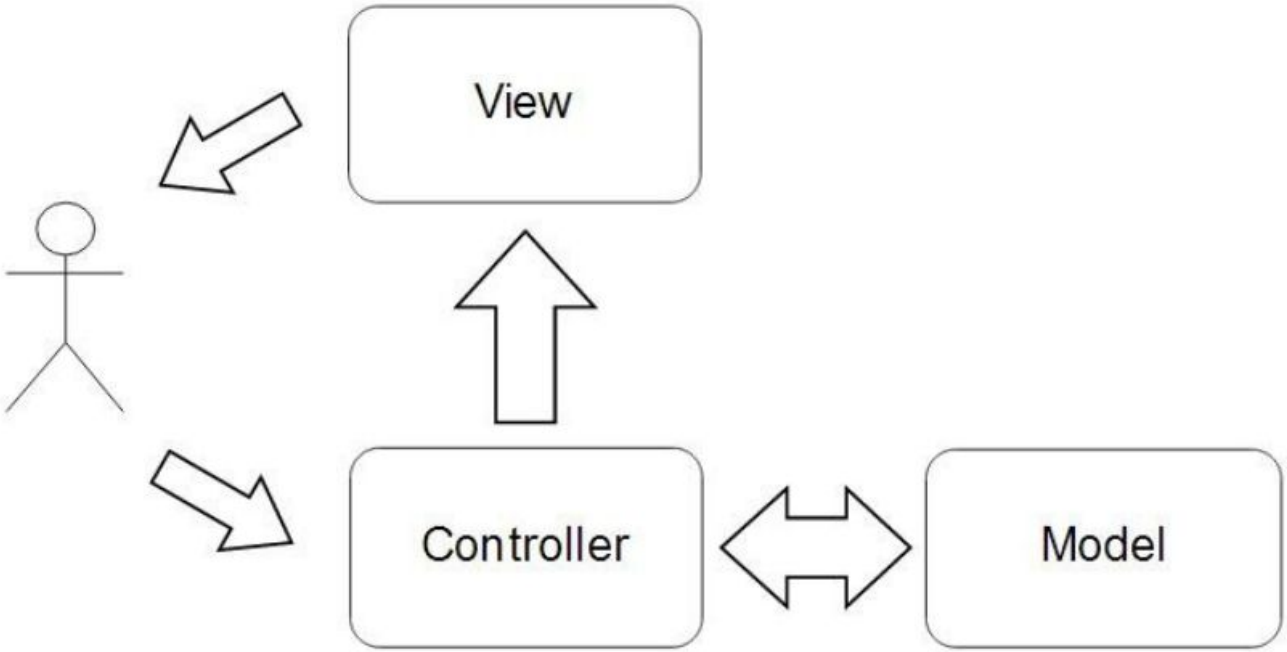
Контроллер

```
@Controller
public class EmailController {

    @GetMapping("/")
    public String indexForm(Model model) {
        model.addAttribute("emailForm", new EmailForm());
        return "index";
    }

    @PostMapping("/sendEmail")
    public String sendEmail(@ModelAttribute EmailForm emailForm, Model model) {
        if (emailForm.getTo().isEmpty()) {
            model.addAttribute("error", "'Кому' не должно быть пустым");
            return "index";
        }
        return "result";
    }
}
```

Варианты реализации MVC



Варианты реализации MVC



Вспомогательные компоненты

- Маршрутизатор
- Сервисы

Маршрутизатор

Маршрутизатор

Основное назначение - определение контроллера, которому будет передано управление.

Маршрутизатор

```
@Controller(“some-url”)
public class EmailController {

    @GetMapping("/")
    public String indexForm(Model model) {
        model.addAttribute("emailForm", new EmailForm());
        return "index";
```

Сервисы

Сервисы

- выполняют “глобальные” задачи
- как правило, доступны внутри контроллеров

Сервисы

```
class UserController extends BaseController
{
  public function sendAction($id, $message) {
    $user = $this->getRepository('user')->find($id);

    if (!$user || $user->isDeleted()) {
      return $this->generateError('user.notFound');
    }

    $mailer = $this->get('mailer');

    return $mailer->send($user->getEmail(), $message);
  }
  ...
}
```

Типы контроллеров

- frontend
- command (CLI)
- API

Frontend

Типовые задачи:

- вывод html-страниц (или отдельных блоков)
- обработка запросов отправки форм

Frontend

```
@Controller
public class EmailController {

    @GetMapping("/")
    public String indexForm(Model model) {
        model.addAttribute("emailForm", new EmailForm());
        return "index";
    }

    @PostMapping("/sendEmail")
    public String sendEmail(@ModelAttribute EmailForm emailForm, Model model) {
        if (emailForm.getTo().isEmpty()) {
            model.addAttribute("error", "'Кому' не должно быть пустым");
            return "index";
        }
        return "result";
    }
}
```

Command

Типовые задачи:

- Импорт/экспорт данных
- Фоновые службы

Command

```
@Service
public class TimeSeriesCollector {

    @Scheduled(cron = "0 0 * * * ?")
    public void collectTimeSeries() {
        timeSeriesEmployeeService.collectEmployeeTimeSeries();
        timeSeriesUnitService.collectUnitTimeSeries();
        timeSeriesToolUsageService.collectToolUsageTimeSeries();
        timeSeriesToolWorkTimeFundService.collectToolWorkTimeFundTimeSeries();
        log.warn("time series collected");
    }

}
```

API

Типовые задачи:

- Взаимодействие с внешними приложениями
- Интеграция с компонентами приложения через AJAX

API

```
public class TimeSeriesController {

    @GetMapping("employee")
    public Response<List<TimeSeriesValueDto>> getEmployeeNum() {
        return new Response<>(timeSeriesService.getEmployeeTimeSeries(versionId));
    }
}
```

Варианты реализации модели

- ActiveRecord
- ORM

Шаблонизаторы

- PHP
- Twig
- Smarty
- XSLT
- Thymeleaf

Thymeleaf. Базовый шаблон

```
<!DOCTYPE html>
<html lang="ru"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
...
</head>

<body >
...
<div layout:fragment="content">
</div>
...
</body>
</html>
```

Thymeleaf. Базовый шаблон

```
<!DOCTYPE html>
<html lang="en"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout" layout:decorator="default">
<head>
</head>
<body>
<div class="container" layout:fragment="content">
   <section id="papers">
       <div class="container">
           <div class="row" id="paper-list">
               <div class="col-lg-12 text-center">
                   <h2 class="section-heading text-uppercase">Статьи</h2>
                   <a href="./dashboard"><h5>Перейти на тестовую страницу Алёны</h5></a>
                   <a href="./myPaper"><h4>Ссылка на пустую страницу</h4></a>
               </div>
           </div>
       </div>
   </section>
  </div>
</body>

</html>
```

Ссылки

https://gitlab.com/romanov73/spring-mvc-example