Pro Java №37

Логирование — это процесс записи информации о событиях, происходящих в приложении, системе или сервисе. Оно используется для мониторинга, анализа, диагностики и улучшения производительности приложения. Логирование помогает разработчикам выявлять и устранять ошибки, а также лучше понимать поведение системы.

Уровни логирования:

1. **TRACE**:
    - Самый детализированный уровень, используется для отслеживания шагов выполнения программы.
    - Пример: информация о входе и выходе из методов.
2. **DEBUG**:
    - Используется для детального анализа работы программы.
    - Пример: отладочная информация о состоянии переменных или объектах.
3. **INFO**:
    - Уровень для записи общей информации о работе приложения.
    - Пример: запуск приложения, успешное завершение операций.
4. **WARN**:
    - Указывает на возможные проблемы, которые пока не критичны, но требуют внимания.
    - Пример: превышение лимита запросов или использование устаревших библиотек.
5. **ERROR**:
    - Записываются ошибки, которые могут привести к сбоям в приложении.
    - Пример: невозможность подключения к базе данных.
6. **FATAL**:
    - Уровень для критических ошибок, которые требуют немедленного вмешательства, так как приводят к остановке работы приложения.
    - Пример: полная потеря доступа к важным ресурсам.

Вы можете настроить уровень логирования для каждого пакета в вашем приложении, чтобы более точно контролировать, какие сообщения записываются в логи. Это особенно полезно для управления детализацией информации от различных частей системы.

Аннотация `@Slf4j` из проекта Lombok автоматически создаёт экземпляр логгера для вашего класса, упрощая процесс логирования. Она интегрируется с библиотекой SLF4J и позволяет избежать необходимости вручную объявлять логгер.

Аннотация `@JoinColumn` в JPA используется для указания столбца, который соединяет две таблицы в базе данных. Она применяется для определения связи между сущностями, таких как `@OneToOne`, `@ManyToOne`, `@OneToMany`, и устанавливает, какой столбец будет выступать в роли "соединительной" колонки в реляционной базе данных.

Основные каскадные типы:

1. `CascadeType.PERSIST`:
    - Связанная сущность сохраняется автоматически, если сохраняется родительская сущность.
    - Пример: Если вы сохраните `Order`, связанный с ним `User` автоматически сохранится.
2. `CascadeType.MERGE`:
    - Связанная сущность обновляется автоматически, если обновляется родительская сущность.
3. `CascadeType.REMOVE`:
    - Удаляет связанную сущность, если удаляется родительская сущность.
4. `CascadeType.REFRESH`:
    - Обновляет (перезагружает) состояние связанной сущности из базы данных.
5. `CascadeType.DETACH`:
    - Указывает, что связанная сущность отделяется от EntityManager, когда родительская сущность отделяется.
6. `CascadeType.ALL`:
    - Включает все вышеуказанные типы каскадного поведения.