**Возможные стратегии загрузки коллекций в Hibernate**

В [предыдущей статье](https://dou.ua/lenta/articles/jpa-fetch-types/) я рассматривал детали различных типов и стратегий загрузки коллекций в JPA. В данной статье будут рассмотрены режимы загрузки коллекций в Hibernate.

Для тех, кто не читал, повторюсь: отношениям один-ко-многим или многие-ко-многим между таблицами реляционной базы данных в объектном виде соответствуют свойства сущности типа List или Set, размеченные аннотациями @OneToMany или @ManyToMany. При работе с сущностями, которые содержат коллекции других сущностей, возникает проблема известная как «N+1 selects». Первый запрос выберет только корневые сущности, а каждая связанная коллекция будет загружена отдельным запросом. Таким образом ORM выполняет N+1 SQL запросов, где N — количество корневых сущностей в результирующей выборке запроса.

Итак, самая популярная реализация JPA, Hibernate предоставляет множество способов управления загрузкой отношений один-ко-многим и многие-ко-многим.

FetchMode в Hibernate говорит как мы хотим, чтоб связанные сущности или коллекции были загружены:  
- SELECT — используя по дополнительному SQL запросу на коллекцию,

- SELECT + аннотация @BatchSize (или атрибут batch-size в XML) - устанавливается количество коллекций, которые будут загружаться в одном запросе,

- SUBSELECT— в дополнительном запросе, используя SUBSELECT,  
- JOIN — в одном запросе с корневой сущностью, используя SQL оператор JOIN.

Пример

Продолжим рассматривать пример из предыдущей части статьи, в котором сущность Book владеет отношениями многие-ко-многим с сущностями Author и Category. Дополним этот пример, добавив аннотации из пакета org.hibernate.annotations, которые не являются частью JPA. Пример целиком доступен на [Github](https://github.com/evgeniy-khist/examples/tree/master/jpa-hibernate-fetching-strategies/hibernate-fetch-mode" \t "_blank).

@Entity

public **class** **Book** **extends** **AbstractBook** {

@ManyToMany(fetch = FetchType.EAGER)

**private** List<Author> authors = **new** ArrayList<>();

@ManyToMany

**private** List<Category> categories = **new** ArrayList<>();

/\*...\*/

}

@Entity

public **class** **BookFetchModeSelect** **extends** **AbstractBook** {

@ManyToMany(fetch = FetchType.EAGER)

@Fetch(FetchMode.SELECT)

**private** List<Author> authors = **new** ArrayList<>();

@ManyToMany

@Fetch(FetchMode.SELECT)

**private** List<Category> categories = **new** ArrayList<>();

/\*...\*/

}

@Entity

public **class** **BookFetchModeJoin** **extends** **AbstractBook** {

@ManyToMany(fetch = FetchType.EAGER)

@Fetch(FetchMode.JOIN)

**private** List<Author> authors = **new** ArrayList<>();

@ManyToMany

@Fetch(FetchMode.JOIN)

**private** List<Category> categories = **new** ArrayList<>();

/\*...\*/

}

@Entity

public **class** **BookFetchModeSubselect** **extends** **AbstractBook** {

@ManyToMany(fetch = FetchType.EAGER)

@Fetch(FetchMode.SUBSELECT)

**private** List<Author> authors = **new** ArrayList<>();

@ManyToMany

@Fetch(FetchMode.SUBSELECT)

**private** List<Category> categories = **new** ArrayList<>();

/\*...\*/

}

@Entity

public **class** **BookBatchSize** **extends** **AbstractBook** {

@ManyToMany(fetch = FetchType.EAGER)

// Явное указание FetchMode.SELECT необходимо

// так как в Criteria API EAGER ассоциации по умолчанию загружаются с

@Fetch(FetchMode.SELECT)

@BatchSize(size = 2)

**private** List<Author> authors = **new** ArrayList<>();

@ManyToMany

@Fetch(FetchMode.SELECT)

@BatchSize(size = 2)

**private** List<Category> categories = **new** ArrayList<>();

/\*...\*/

}

Для более наглядной демонстрации отличий режимов загрузки необходимо добавить больше тестовых данных:

Session session = sessionFactory.getCurrentSession();

Category softwareDevelopment = **new** Category();

softwareDevelopment.setName("Software development");

session.persist(softwareDevelopment);

Category systemDesign = **new** Category();

systemDesign.setName("System design");

session.persist(systemDesign);

Author martinFowler = **new** Author();

martinFowler.setFullName("Martin Fowler");

session.persist(martinFowler);

AbstractBook poeaa = bookSupplier.get();

poeaa.setIsbn("007-6092019909");

poeaa.setTitle("Patterns of Enterprise Application Architecture");

poeaa.setPublicationDate(Date.from(Instant.parse("2002-11-15T00:00:00.00Z")));

poeaa.setAuthors(asList(martinFowler));

poeaa.setCategories(asList(softwareDevelopment, systemDesign));

session.persist(poeaa);

Author gregorHohpe = **new** Author();

gregorHohpe.setFullName("Gregor Hohpe");

session.persist(gregorHohpe);

Author bobbyWoolf = **new** Author();

bobbyWoolf.setFullName("Bobby Woolf");

session.persist(bobbyWoolf);

AbstractBook eip = bookSupplier.get();

eip.setIsbn("978-0321200686");

eip.setTitle("Enterprise Integration Patterns");

eip.setPublicationDate(Date.from(Instant.parse("2003-10-20T00:00:00.00Z")));

eip.setAuthors(asList(gregorHohpe, bobbyWoolf));

eip.setCategories(asList(softwareDevelopment, systemDesign));

session.persist(eip);

Category objectOrientedSoftwareDesign = **new** Category();

objectOrientedSoftwareDesign.setName("Object-Oriented Software Design");

session.persist(objectOrientedSoftwareDesign);

Author ericEvans = **new** Author();

ericEvans.setFullName("Eric Evans");

session.persist(ericEvans);

AbstractBook ddd = bookSupplier.get();

ddd.setIsbn("860-1404361814");

ddd.setTitle("Domain-Driven Design: Tackling Complexity in the Heart of Software");

ddd.setPublicationDate(Date.from(Instant.parse("2003-08-01T00:00:00.00Z")));

ddd.setAuthors(asList(ericEvans));

ddd.setCategories(asList(softwareDevelopment, systemDesign, objectOrientedSoftwareDesign));

session.persist(ddd);

Category networkingCloudComputing = **new** Category();

networkingCloudComputing.setName("Networking & Cloud Computing");

session.persist(networkingCloudComputing);

Category databasesBigData = **new** Category();

databasesBigData.setName("Databases & Big Data");

session.persist(databasesBigData);

Author pramodSadalage = **new** Author();

pramodSadalage.setFullName("Pramod J. Sadalage");

session.persist(pramodSadalage);

AbstractBook nosql = bookSupplier.get();

nosql.setIsbn("978-0321826626");

nosql.setTitle("NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence");

nosql.setPublicationDate(Date.from(Instant.parse("2012-08-18T00:00:00.00Z")));

nosql.setAuthors(asList(pramodSadalage, martinFowler));

nosql.setCategories(asList(networkingCloudComputing, databasesBigData));

session.persist(nosql);

В тестах используется Hibernate 5.2.0.Final.

HQL запрос и FetchMode по умолчанию

Когда аннотация @Fetch не добавлена, HQL и Hibernate Criteria запросы ведут себя по-разному. В случае использования HQL запроса, по умолчанию используется FetchMode.SELECT для связанных коллекций с любым типом загрузки (EAGER и LAZY).

**List** books = getCurrentSession().createQuery("select b from Book b").**list**();

assertEquals(4, books.size());

Сгенерированный SQL:

**select**

book0\_.id **as** id1\_1\_,

book0\_.isbn **as** isbn2\_1\_,

book0\_.publicationDate **as** publicat3\_1\_,

book0\_.title **as** title4\_1\_

**from**

Book book0\_

**select**

authors0\_.Book\_id **as** Book\_id1\_2\_0\_,

authors0\_.authors\_id **as** authors\_2\_2\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

Book\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.Book\_id=?

**select**

authors0\_.Book\_id **as** Book\_id1\_2\_0\_,

authors0\_.authors\_id **as** authors\_2\_2\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

Book\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.Book\_id=?

**select**

authors0\_.Book\_id **as** Book\_id1\_2\_0\_,

authors0\_.authors\_id **as** authors\_2\_2\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

Book\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.Book\_id=?

**select**

authors0\_.Book\_id **as** Book\_id1\_2\_0\_,

authors0\_.authors\_id **as** authors\_2\_2\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

Book\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.Book\_id=?

Hibernate Criteria запрос и FetchMode по умолчанию

Когда аннотация @Fetch не добавлена, в Hibernate Criteria запросах:

- для связанных коллекций с типом загрузки EAGER по умолчанию используется FetchMode.JOIN,

- для связанных коллекций с типом загрузки LAZY по умолчанию используется FetchMode.SELECT.

Коллекции с типом загрузки EAGER будут загружены в одном SQL запросе с корневой сущностью. Результатом запроса будет декартово произведение (cartesian product). Вместо 4 элементов в результирующей выборке, запрос вернет 6, потому что книги «Enterprise Integration Patterns» и «NoSQL» имеют двух авторов и будут дважды встречаться в результатах запроса.

Стоит отметить, что начиная с версии Hibernate 5.2 метод createCriteria класса org.hibernate.Session, который создает экземпляр org.hibernate.Criteria, считает устаревшим и помечен аннотацией @Deprecated. Вместо этого в [Javadoc](https://docs.jboss.org/hibernate/orm/5.2/javadocs/org/hibernate/SharedSessionContract.html" \l "createCriteria-java.lang.Class-" \t "_blank) рекомендуется использовать JPA Criteria.

List books = getCurrentSession().createCriteria(Book.**class**).list();

assertEquals(6, books.size());

Сгенерированный SQL:

**select**

this\_.id **as** id1\_1\_1\_,

this\_.isbn **as** isbn2\_1\_1\_,

this\_.publicationDate **as** publicat3\_1\_1\_,

this\_.title **as** title4\_1\_1\_,

authors2\_.Book\_id **as** Book\_id1\_2\_3\_,

author3\_.id **as** authors\_2\_2\_3\_,

author3\_.id **as** id1\_0\_0\_,

author3\_.fullName **as** fullName2\_0\_0\_

**from**

Book this\_

**left** **outer** **join**

Book\_Author authors2\_

**on** this\_.id=authors2\_.Book\_id

**left** **outer** **join**

Author author3\_

**on** authors2\_.authors\_id=author3\_.id

HQL и Hibernate Criteria запросы и FetchMode.SELECT

С режимом загрузки FetchMode.SELECT первый запрос выберет только корневые сущности, а каждая связанная коллекция будет загружена отдельным запросом. Так как для данного примера мы сохранили 4 сущности, то запросов будет 5. Один для загрузки сущностей Book и по одному для каждой из 4 сущностей Book для загрузки списка сущностей Author. Список сущностей Category имеет тип загрузки по умолчанию (LAZY для коллекций) и будет загружен отдельным запросом при первом обращении в коде.

**List** books = getCurrentSession().createQuery("select b from BookFetchModeSelect b").**list**();

assertEquals(4, books.size());

При режиме загрузки FetchMode.SELECT HQL и Hibernate Criteria запросы ведут себя одинаково.

List books = getCurrentSession().createCriteria(BookFetchModeSelect.**class**).list();

assertEquals(4, books.size());

Сгенерированный SQL:

**select**

bookfetchm0\_.id **as** id1\_10\_,

bookfetchm0\_.isbn **as** isbn2\_10\_,

bookfetchm0\_.publicationDate **as** publicat3\_10\_,

bookfetchm0\_.title **as** title4\_10\_

**from**

BookFetchModeSelect bookfetchm0\_

**select**

authors0\_.BookFetchModeSelect\_id **as** BookFetc1\_11\_0\_,

authors0\_.authors\_id **as** authors\_2\_11\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

BookFetchModeSelect\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookFetchModeSelect\_id=?

**select**

authors0\_.BookFetchModeSelect\_id **as** BookFetc1\_11\_0\_,

authors0\_.authors\_id **as** authors\_2\_11\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

BookFetchModeSelect\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookFetchModeSelect\_id=?

**select**

authors0\_.BookFetchModeSelect\_id **as** BookFetc1\_11\_0\_,

authors0\_.authors\_id **as** authors\_2\_11\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

BookFetchModeSelect\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookFetchModeSelect\_id=?

**select**

authors0\_.BookFetchModeSelect\_id **as** BookFetc1\_11\_0\_,

authors0\_.authors\_id **as** authors\_2\_11\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

BookFetchModeSelect\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookFetchModeSelect\_id=?<pre>

<h2>HQL и Hibernate Criteria запросы и FetchMode.SUBSELECT</h2>

При режиме загрузки <code>FetchMode.SUBSELECT</code> будет выполнено 2 **SQL** запроса. Первый загрузит корневые сущности, а второй - связанные коллекции для всех корневых сущностей из результатов первого **SQL** запроса, используя подзапрос.

<pre>**List** books = getCurrentSession().createQuery("select b from BookFetchModeSubselect b").list();

assertEquals(4, books.size());

HQL и Hibernate Criteria запросы ведут себя одинаково и при режиме загрузки FetchMode.SUBSELECT.

List books = getCurrentSession().createCriteria(BookFetchModeSubselect.**class**).list();

assertEquals(4, books.size());

Сгенерированный SQL:

**select**

bookfetchm0\_.id **as** id1\_13\_,

bookfetchm0\_.isbn **as** isbn2\_13\_,

bookfetchm0\_.publicationDate **as** publicat3\_13\_,

bookfetchm0\_.title **as** title4\_13\_

**from**

BookFetchModeSubselect bookfetchm0\_

**select**

authors0\_.BookFetchModeSubselect\_id **as** BookFetc1\_14\_1\_,

authors0\_.authors\_id **as** authors\_2\_14\_1\_,

author1\_.id **as** id1\_0\_0\_,

author1\_.fullName **as** fullName2\_0\_0\_

**from**

BookFetchModeSubselect\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookFetchModeSubselect\_id **in** (

**select**

bookfetchm0\_.id

**from**

BookFetchModeSubselect bookfetchm0\_

)

HQL запрос и FetchMode.JOIN

Поведение HQL запросов при режиме загрузке FetchMode.JOIN, на первый взгляд, немного неожиданное. Вместо того, чтобы загрузить связанные коллекции, помеченные аннотацией @Fetch(FetchMode.JOIN), в одном запросе с корневыми сущностями, используя SQL оператор JOIN, HQL запрос транслируется в несколько SQL запросов по типа FetchMode.SELECT. Но в отличии от FetchMode.SELECT, при FetchMode.JOIN будет игнорироваться указанный тип загрузки (LAZY и EAGER) и все коллекции будут загружены сразу, а не при первом обращении в коде (поведение соответствующее типу EAGER).

Таким образом в следующем примере будет выполнено 8 SQL запросов. Один запрос для загрузки корневых сущностей Book и для каждой из 4 корневых сущностей из результатов первого SQL запроса по одному запросу для загрузки списка сущностей Author и по одному запросу для загрузки списка сущностей Category.

**List** books = getCurrentSession().createQuery("select b from BookFetchModeJoin b").**list**();

assertEquals(4, books.size());

Сгенерированный SQL:

**select**

bookfetchm0\_.id **as** id1\_7\_,

bookfetchm0\_.isbn **as** isbn2\_7\_,

bookfetchm0\_.publicationDate **as** publicat3\_7\_,

bookfetchm0\_.title **as** title4\_7\_

**from**

BookFetchModeJoin bookfetchm0\_

**select**

categories0\_.BookFetchModeJoin\_id **as** BookFetc1\_9\_0\_,

categories0\_.categories\_id **as** categori2\_9\_0\_,

category1\_.id **as** id1\_16\_1\_,

category1\_.description **as** descript2\_16\_1\_,

category1\_.name **as** name3\_16\_1\_

**from**

BookFetchModeJoin\_Category categories0\_

**inner** **join**

**Category** category1\_

**on** categories0\_.categories\_id=category1\_.id

**where**

categories0\_.BookFetchModeJoin\_id=?

**select**

authors0\_.BookFetchModeJoin\_id **as** BookFetc1\_8\_0\_,

authors0\_.authors\_id **as** authors\_2\_8\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

BookFetchModeJoin\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookFetchModeJoin\_id=?

**select**

categories0\_.BookFetchModeJoin\_id **as** BookFetc1\_9\_0\_,

categories0\_.categories\_id **as** categori2\_9\_0\_,

category1\_.id **as** id1\_16\_1\_,

category1\_.description **as** descript2\_16\_1\_,

category1\_.name **as** name3\_16\_1\_

**from**

BookFetchModeJoin\_Category categories0\_

**inner** **join**

**Category** category1\_

**on** categories0\_.categories\_id=category1\_.id

**where**

categories0\_.BookFetchModeJoin\_id=?

**select**

authors0\_.BookFetchModeJoin\_id **as** BookFetc1\_8\_0\_,

authors0\_.authors\_id **as** authors\_2\_8\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

BookFetchModeJoin\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookFetchModeJoin\_id=?

**select**

categories0\_.BookFetchModeJoin\_id **as** BookFetc1\_9\_0\_,

categories0\_.categories\_id **as** categori2\_9\_0\_,

category1\_.id **as** id1\_16\_1\_,

category1\_.description **as** descript2\_16\_1\_,

category1\_.name **as** name3\_16\_1\_

**from**

BookFetchModeJoin\_Category categories0\_

**inner** **join**

**Category** category1\_

**on** categories0\_.categories\_id=category1\_.id

**where**

categories0\_.BookFetchModeJoin\_id=?

**select**

authors0\_.BookFetchModeJoin\_id **as** BookFetc1\_8\_0\_,

authors0\_.authors\_id **as** authors\_2\_8\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

BookFetchModeJoin\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookFetchModeJoin\_id=?

**select**

categories0\_.BookFetchModeJoin\_id **as** BookFetc1\_9\_0\_,

categories0\_.categories\_id **as** categori2\_9\_0\_,

category1\_.id **as** id1\_16\_1\_,

category1\_.description **as** descript2\_16\_1\_,

category1\_.name **as** name3\_16\_1\_

**from**

BookFetchModeJoin\_Category categories0\_

**inner** **join**

**Category** category1\_

**on** categories0\_.categories\_id=category1\_.id

**where**

categories0\_.BookFetchModeJoin\_id=?

**select**

authors0\_.BookFetchModeJoin\_id **as** BookFetc1\_8\_0\_,

authors0\_.authors\_id **as** authors\_2\_8\_0\_,

author1\_.id **as** id1\_0\_1\_,

author1\_.fullName **as** fullName2\_0\_1\_

**from**

BookFetchModeJoin\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookFetchModeJoin\_id=?

HQL запрос с «join fetch»

Чтобы корневые сущности со связанными коллекциями были загружены в одном SQL запросе, используйте оператор JOIN FETCH. Результатом запроса будет декартово произведение. Вместо 4 элементов в результирующей выборке, запрос вернет 6, потому что книги «Enterprise Integration Patterns» и «NoSQL» имеют двух авторов и будут дважды встречаться в результатах запроса. Коллекции, которые не были присоединены оператором JOIN FETCH, будут загружены согласно типу и режиму загрузки.

В данном примере будет выполнено по дополнительному запросу на каждую корневую сущность из списка результатов для загрузки списка сущностей Category.

List books = getCurrentSession().createQuery("**select** b **from** BookFetchModeJoin b **join** **fetch** b.authors a").list();

assertEquals(6, books.size());

Сгенерированный SQL:

**select**

bookfetchm0\_.id **as** id1\_7\_0\_,

author2\_.id **as** id1\_0\_1\_,

bookfetchm0\_.isbn **as** isbn2\_7\_0\_,

bookfetchm0\_.publicationDate **as** publicat3\_7\_0\_,

bookfetchm0\_.title **as** title4\_7\_0\_,

author2\_.fullName **as** fullName2\_0\_1\_,

authors1\_.BookFetchModeJoin\_id **as** BookFetc1\_8\_0\_\_,

authors1\_.authors\_id **as** authors\_2\_8\_0\_\_

**from**

BookFetchModeJoin bookfetchm0\_

**inner** **join**

BookFetchModeJoin\_Author authors1\_

**on** bookfetchm0\_.id=authors1\_.BookFetchModeJoin\_id

**inner** **join**

Author author2\_

**on** authors1\_.authors\_id=author2\_.id

**select**

categories0\_.BookFetchModeJoin\_id **as** BookFetc1\_9\_0\_,

categories0\_.categories\_id **as** categori2\_9\_0\_,

category1\_.id **as** id1\_16\_1\_,

category1\_.description **as** descript2\_16\_1\_,

category1\_.name **as** name3\_16\_1\_

**from**

BookFetchModeJoin\_Category categories0\_

**inner** **join**

**Category** category1\_

**on** categories0\_.categories\_id=category1\_.id

**where**

categories0\_.BookFetchModeJoin\_id=?

**select**

categories0\_.BookFetchModeJoin\_id **as** BookFetc1\_9\_0\_,

categories0\_.categories\_id **as** categori2\_9\_0\_,

category1\_.id **as** id1\_16\_1\_,

category1\_.description **as** descript2\_16\_1\_,

category1\_.name **as** name3\_16\_1\_

**from**

BookFetchModeJoin\_Category categories0\_

**inner** **join**

**Category** category1\_

**on** categories0\_.categories\_id=category1\_.id

**where**

categories0\_.BookFetchModeJoin\_id=?

**select**

categories0\_.BookFetchModeJoin\_id **as** BookFetc1\_9\_0\_,

categories0\_.categories\_id **as** categori2\_9\_0\_,

category1\_.id **as** id1\_16\_1\_,

category1\_.description **as** descript2\_16\_1\_,

category1\_.name **as** name3\_16\_1\_

**from**

BookFetchModeJoin\_Category categories0\_

**inner** **join**

**Category** category1\_

**on** categories0\_.categories\_id=category1\_.id

**where**

categories0\_.BookFetchModeJoin\_id=?

**select**

categories0\_.BookFetchModeJoin\_id **as** BookFetc1\_9\_0\_,

categories0\_.categories\_id **as** categori2\_9\_0\_,

category1\_.id **as** id1\_16\_1\_,

category1\_.description **as** descript2\_16\_1\_,

category1\_.name **as** name3\_16\_1\_

**from**

BookFetchModeJoin\_Category categories0\_

**inner** **join**

**Category** category1\_

**on** categories0\_.categories\_id=category1\_.id

**where**

categories0\_.BookFetchModeJoin\_id=?

Hibernate Criteria запрос и FetchMode.JOIN

В Hibernate Criteria запросах связанные коллекции с режимом загрузки FetchMode.JOIN будут загружены сразу (тип загрузки EAGER) и в одном запросе с корневыми сущностями при помощи SQL оператора JOIN. Как уже рассматривалось [в прошлой статье](https://dou.ua/lenta/articles/jpa-fetch-types/), только одна коллекций, которая загружается со стратегией JOIN может быть типа java.util.List, остальные коллекции, которые загружаются стратегией JOIN должны быть типа java.util.Set, иначе будет выброшено исключение org.hibernate.loader.MultipleBagFetchException.

List books = getCurrentSession().createCriteria(BookFetchModeJoin.**class**).list();

Будет выброшено исключение:  
org.hibernate.loader.MultipleBagFetchException: cannot simultaneously fetch multiple bags

HQL и Hibernate Criteria запросы и FetchMode.SELECT с @BatchSize

@BatchSize устанавливает количество коллекций, которые должны быть загружены в одном SQL запросе. Если результат запроса содержит 4 сущности, каждая из которых имеет по связанной коллекции, при режим загрузки SELECT будет выполнено 5 запросов. Один запрос для загрузки всех корневых сущностей и по одному для загрузки связанной коллекции каждой из 4 корневых сущностей. @BatchSize(size = 2) указывает ORM загружать по 2 связанные коллекции в одном запросе. Таким образом, всего будет выполнено 3 запроса вместо 5. Один запрос для загрузки всех корневых сущностей и еще 2 запроса, каждый из которых загрузит по 2 связанные коллекции для 2 корневых сущностей.

**List** books = getCurrentSession().createQuery("select b from BookBatchSize b").**list**();

assertEquals(4, books.size());

Как уже отмечалось ранее, при режиме загрузки FetchMode.SELECT HQL и Hibernate Criteria запросы ведут себя одинаково. При использовании @BatchSize поведение также будет одинаковым.

List books = getCurrentSession().createCriteria(BookBatchSize.**class**).list();

assertEquals(4, books.size());

Сгенерированный SQL:

**select**

bookbatchs0\_.id **as** id1\_4\_,

bookbatchs0\_.isbn **as** isbn2\_4\_,

bookbatchs0\_.publicationDate **as** publicat3\_4\_,

bookbatchs0\_.title **as** title4\_4\_

**from**

BookBatchSize bookbatchs0\_

**select**

authors0\_.BookBatchSize\_id **as** BookBatc1\_5\_1\_,

authors0\_.authors\_id **as** authors\_2\_5\_1\_,

author1\_.id **as** id1\_0\_0\_,

author1\_.fullName **as** fullName2\_0\_0\_

**from**

BookBatchSize\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookBatchSize\_id **in** (

?, ?

)

**select**

authors0\_.BookBatchSize\_id **as** BookBatc1\_5\_1\_,

authors0\_.authors\_id **as** authors\_2\_5\_1\_,

author1\_.id **as** id1\_0\_0\_,

author1\_.fullName **as** fullName2\_0\_0\_

**from**

BookBatchSize\_Author authors0\_

**inner** **join**

Author author1\_

**on** authors0\_.authors\_id=author1\_.id

**where**

authors0\_.BookBatchSize\_id **in** (

?, ?

)

Выводы

Существует немало нюансов, связанных со стратегиями загрузки связанных коллекций в JPA и Hibernate, многие из которых были рассмотрены в этой и [предыдущей](https://dou.ua/lenta/articles/jpa-fetch-types/) статьях.

Не стоит использовать одну выбранную стратегию загрузки повсюду в приложении. Каждый случай надо проанализировать индивидуально и выбрать оптимальную стратегию загрузки. Как это часто бывает, в большинстве случаев стратегия загрузки по умолчанию будет оптимальным вариантом. Если приложение испытывает проблемы с производительностью, которые вызваны неправильным выбором стратегии загрузки, для анализа пригодятся следующие свойства Hibernate конфигурации: hibernate.show\_sql=true, hibernate.format\_sql=true и hibernate.use\_sql\_comments=true.

**[https://s.dou.ua/img/avatars/40x40_254652.jpg](https://dou.ua/users/igor-dmitriev-3/)**Engineer Manager в [SPD-Ukraine](https://jobs.dou.ua/companies/spd-ukraine/)[27.07.2016 14:05](https://dou.ua/lenta/articles/hibernate-fetch-types/#960796)

Как это часто бывает, в большинстве случаев стратегия загрузки по умолчанию будет оптимальным вариантом

Вы серьезно??? Если у вас приложение с дефолтными настройками и вы просто поставили аннотации и у вас будет все ок?? Я даю 99% если это так и у вас более менее не тривиальное CRUD приложение то оно работает медленно.

[Ответить](javascript:;)

[Поддержать](javascript:;)

[**[https://s.dou.ua/img/avatars/25x25_15032270_207028116404678_7812941592273916947_n.jpg](https://dou.ua/users/evgeniykhist/)Evgeniy Khyst**](https://dou.ua/users/evgeniykhist/) Java Solution Architect[27.07.2016 17:01](https://dou.ua/lenta/articles/hibernate-fetch-types/#960989)

99%? Почему не 99.999%? Это демагогия.  
Значения по-умолчанию берутся не с потолка, а чаще всего в результате анализа и так, чтобы удовлетворять наиболее частому сценарию использованию по мнению авторов.

Если у вас приложение с дефолтными настройками и вы просто поставили аннотации и у вас будет все ок??

Все зависит от модели данных и запросов, которые будут выполняться.  
В данной статье рассматриваются разные режими загрузки и отличия в SQL запросах, которые будут сгенерированны. На основании примеров читатели сами могут выбрать оптимальный для их конкретного случая набор настроек. Если говорить о производительности и проблемах с производительностью, которые взваны базой даных, то я более подробно рассматривал этот вопрос в своем докладе [www.slideshare.net/...-databaserelated-problems](http://www.slideshare.net/EvgeniyKhist/application-performance-databaserelated-problems)

[Ответить](javascript:;)

[Поддержать](javascript:;)

[**[https://s.dou.ua/img/avatars/40x40_254652.jpg](https://dou.ua/users/igor-dmitriev-3/)Igor Dmitriev**](https://dou.ua/users/igor-dmitriev-3/) Engineer Manager в [SPD-Ukraine](https://jobs.dou.ua/companies/spd-ukraine/)[27.07.2016 13:49](https://dou.ua/lenta/articles/hibernate-fetch-types/#960775)

Какой толк с вашей статьи если вы ничего нового от себя не написали, взяли материал с книги и перевели его на русский?? Более интересно когда вы показываете реальный опыт с жизни и даете советы как и что использовать основываясь на своем опыте. Вы написали теорию на поверхностном уровне. Где подводные камни и проблемы с которыми вы столкнулись? После прочтения статьи нет четкого понимания что и когда лучше использовать, какой batch size ставить и почему, когда subselect, а когда join и так далее.

Да и @BatchSize можно использовать над классом, а не только над коллекцией. И не понятно из статьи как этот batch size работает, какой алгоритм генерации запроса с IN? В вашем примере batchSize = 2  
Хибернейт сгенерирует два массива запросов с IN: in (?) и in(?,?). Алгоритм можно посмотреть здесь — org.hibernate.internal.util.collections.ArrayHelper getBatchSizes(int maxBatchSize).

Так же есть бага с FetchMode.SUBSELECT и pagination, в один прекрасный момент вы можете положить ваше приложение. И subselect работает на запоминании запроса и получается ситуация что сложно контролировать запросы в приложении и если запрос выполняется не очень быстро и он потом еще скопируется в IN секцию — не очень хороший перформенс будет.  
Я бы не советовал использовать Subselect вообще.

Вот бага — [hibernate.atlassian.net/browse/HHH-2666](https://hibernate.atlassian.net/browse/HHH-2666)  
Она открыта давно и известна, никто не планирует ее фиксить.  
Вы не стали советовать и не предостерегли их и не написали ситуации в какой лучше использовать тот или иной способ.